1.x 版的開發人員指南

AWS SDK for Java 1.

AWS SDK for Java 1.: 1.x 版的開發人員指南

Table of Contents

..... viii
AWS SDK for Java1. 1
已發行的 SDK 第 2 版 1
其他文件與資源 1
Eclipse IDE 支援 2
適用於 Android 的應用程序 2
檢視 SDK 的修訂歷程記錄 2
為早期 SDK 版本構建 Java 參考文檔 2
入門 3
基本設定 3
概要 3
AWS存取入口網站的登入功能 4
設定共用設定檔 4
安裝 Java 開發環境 5
如何獲得AWS SDK for Java 6
先決條件 6
使用構建工具 6
下載預建的罐子 6
從來源建置 7
使用建置工具 8
使用軟體開發套件搭配 Apache Maven 8
使用軟體開發套件搭配 Gradle 11
暫時登入資料 14
設定暫時性憑入 15
重新整理 IMDS 登入資料 15
設定AWS 區域 16
使用 AWS SDK for Java 18
與 AWS 開發的最佳做法 AWS SDK for Java 18
S3 18
建立服務用戶端 19
取得用戶端建置器 19
建立非同步用戶端 20
使用 DefaultClient 21
用戶端生命週期 21

提供臨時憑證 22
使用預設登入資料供應商鏈結 22
指定認證提供者或提供者鏈結 25
明確指定臨時認證 26
詳細資訊 26
AWS 區域 選擇 26
檢查區域中的服務可用性 26
選擇區域 27
選擇特定端點 27
從環境中自動確定區域 28
例外狀況處理 29
為什麼使用未檢查的例外狀況？ 29
AmazonServiceException （和子類別） 30
AmazonClientException 30
非同步程式設計 30
爪哇期貨 31
非同步回呼 32
最佳實務 34
記錄 AWS SDK for Java 呼叫 34
下載日誌 JAR 35
設定 Classpath 35
服務特定錯誤與警告 35
請求/回應摘要記錄 36
詳細連線記錄 37
延遲指標記錄 37
客戶端組態 38
代理組態 38
傳輸組態 38
TCP 通訊端緩衝區大小提示 40
存取控制政策 40
Amazon S3 例子 41
Amazon SQS 例子 41
Amazon SNS 示例 41
設定 DNS 名稱查詢的 JVM TTL 42
如何設定 JVM TTL 42
啟用指標 AWS SDK for Java 43

如何啟用 Java SDK 指標生成 43
可用的量度類型 44
詳細資訊 47
程式碼範例 48
AWS SDK for Java2.x 48
Amazon CloudWatch 範例 48
從 CloudWatch 取得指標 49
發佈自訂指標資料 50
使用 CloudWatch 警報 52
使用 CloudWatch 中的警示動作 55
傳送事件到 CloudWatch 56
Amazon DynamoDB 範例 59
處理資料表DynamoDB 59
在 中處理項目DynamoDB 66
Amazon EC2 範例 73
教學課程：建立 EC2 執行個體 73
使用 IAM 角色授予AWS上的資源Amazon EC2 78
教學課程：Amazon EC2Spot Instances 83
教學課程：進階Amazon EC2Spot 請求管理 93
管理 Amazon EC2 執行個體 109
在 Amazon EC2 中使用彈性 IP 地址 114
使用區域與可用區域 117
使用 Amazon EC2 金鑰對 120
在 Amazon EC2 中使用安全群組 122
AWS Identity and Access Management(IAM) 示例 126
管理 IAM 存取金鑰 126
管理 IAM 使用者 131
使用 IAM 帳戶別名 134
處理 IAM 政策 136
處理 IAM 伺服器憑證 141
亞馬遜Lambda範例 144
服務操作 145
Amazon Pinpoint 範例 148
在中創建和刪除應用Amazon Pinpoint 149
建立適用於的端點Amazon Pinpoint 150
在中建立客羣Amazon Pinpoint 152

建立行銷活動Amazon Pinpoint 154
更新頻道Amazon Pinpoint 156
Amazon S3 範例 157
創建、列出和刪除Amazon S3儲存貯體 157
對物件執行Amazon S3作業 162
管理Amazon S3存儲桶和對象的訪問權限 167
管理對 的存取Amazon S3使用儲存儲體政策 171
使用 TransferManager 為了Amazon S3操作 174
設定Amazon S3存儲桶即網站 186
使用Amazon S3用戶端加密 189
Amazon SQS 範例 195
使用Amazon SQS訊息佇列 195
傳送、接收和刪除Amazon SQS訊息 198
為啟用長輪詢Amazon SQS訊息 QueueRequest 200
設定可見性逾時Amazon SQS 203
在 Amazon SQS 中使用無效字母佇列 205
Amazon SWF 範例 207
SWF 基本概況 207
建立簡單的Amazon SWF應用程式 209
Lambda 任務 226
正常關閉活動和工作流工作人員 231
註冊網域 234
列出網域 234
SDK 附帶的代碼示例 235
如何獲取範例 235
使用命令行構建和運行示例 235
使用 Eclipse IDE 生成和運行示例 236
安全 238
資料保護 238
強制執行最低 TLS 版本 239
如何檢查 TLS 版本 239
強制執行最低 TLS 版本 240
身分和存取權管理 240
物件 240
使用身分驗證 241
使用政策管理存取權 243

如何 AWS 服務 使用 IAM 245
疑難排解 AWS 身分和存取 245
合規驗證 247
恢復能力 248
基礎設施安全性 248
S3 加密用戶端移轉 249
必要條件 249
移轉概觀 249
更新現有用戶端以讀取新格式 250
將加密和解密用戶端移轉至 V2 251
其他範例 253
開啟 PGP 金鑰 255
目前的金鑰 255
文件歷史記錄 257

我們宣布了即將推 end-of-support 出的 AWS SDK for Java (v1)。我們建議您移轉至 AWS SDK for Java v2。有關日期，其他詳細信息以及如何遷移的信息，請參閱鏈接的公告。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

開發人員指南-AWS SDK for Java 1.x

會針對AWS服務AWS SDK for Java提供一個 Java API。使用此開發套件，您可以輕鬆建置 Java 應用程式搭配 Amazon S3、Amazon EC2、DynamoDB 等服務使用。我們會定期新增新服務的支援到 AWS SDK for Java。如需每個 SDK 發行版本所包含的支援服務及其 API 版本的清單，請檢視您正在使用的版本的版本說明。

已發行的 SDK 第 2 版

看看新的AWS SDK for Java 2.x 網aws-sdk-java-v站。https://github.com/aws/ 它包括期待已久的功能，例如插入 HTTP 實現的方法。若要開始使用，請參閱 AWS SDK for Java2.x 開發人員指南。

其他文件與資源

除了本指南以外，以下是適用於 AWS SDK for Java 開發人員的寶貴線上資源：

- AWS SDK for Java API 參考
- Java 開發人員部落格
- Java 開發人員論壇
- GitHub:
  - 文件來源
  - 文件問題
  - 開發套件來源
  - SDK 問題
  - SDK 範例
  - 吉特通道
- AWS 程式碼範例目錄
- @awsforjava (Twitter)
- 版本備註

Eclipse IDE 支援

如果您使用 Eclipse IDE 開發程式碼，您可以使用將新增AWS Toolkit for EclipseAWS SDK for Java至現有的 Eclipse 專案或建立新AWS SDK for Java專案。此工具組也支援建立和上傳Lambda功能、啟動和監控Amazon EC2執行個體、管理IAM使用者和安全性群組、AWS CloudFormation範本編輯器等。

如需完整文件，請參閱AWS Toolkit for Eclipse使用者指南。

適用於 Android 的應用程序

如果您是 Android 開發人員，請發Amazon Web Services布專門為安卓開發而設計的 SDK：Amplify 安卓系統（適用於安卓系統的AWS移動 SDK）。

檢視 SDK 的修訂歷程記錄

若要檢視的發行歷程記錄AWS SDK for Java，包括每個 SDK 版本的變更和支援的服務，請參閱 SDK 的版本說明。

為早期 SDK 版本構建 Java 參考文檔

AWS SDK for JavaAPI 參考代表 SDK 1.x 版的最新版本。如果您使用的是先前版本的 1.x 版本，則可能需要訪問與您正在使用的版本匹配的 SDK 參考文檔。

構建文檔的最簡單方法是使用 Apache 的 Maven 構建工具。如果您的系統上還沒有 Maven，請先下載並安裝 Maven，然後使用以下說明構建參考文檔。

1. 在 SDK 存放庫的發行版本頁面上，找出並選取您正在使用的 SDK 版本GitHub。
2. 選擇`zip`（大多數平台，包括視窗）或`tar.gz`（Linux，macOS 或 Unix）鏈接將 SDK 下載到您的計算機。
3. 將封存項目項目項目項目項目項目項目項目
4. 在命令行上，導航到解壓縮歸檔的目錄，然後鍵入以下內容。

```
mvn javadoc:javadoc
```

5. 構建完成後，您將在`aws-java-sdk/target/site/apidocs/`目錄中找到生成的 HTML 文檔。

入門

本節提供如何安裝、設定和使用 AWS SDK for Java 的相關資訊。

主題

- 可使用的基本設定 AWS 服務
- 如何獲得AWS SDK for Java
- 使用建置工具
- 設置AWS臨時憑據並AWS 區域進行開發

可使用的基本設定 AWS 服務

概要

若要成功開發使用存取的應AWS 服務用程式AWS SDK for Java，需要下列條件：

- 您必須能夠登入中提供的AWS存取入口網站AWS IAM Identity Center。
- 為 SDK 設定的 IAM 角色權限必須允許存取應用程式所AWS 服務需的權限。
  與PowerUserAccessAWS受管理策略相關聯的權限足以滿足大多數開發需求。
- 具有下列元素的開發環境：
  - 以下列方式設定的共用組態檔案：
    - 檔`config`案包含指定的預設紀要AWS 區域。
    - `credentials`檔案包含作為預設設定檔一部分的臨時身份證明。
  - 一個合適的 Java 安裝。
  - 構建自動化工具，如 Maven 或搖籃。
  - 使用程式碼的文字編輯器。
  - （可選，但建議使用）IDE（集成開發環境），例如 IntelliJ IDEA，日食或 NetBeans.

    當您使用 IDE 時，您還可以集成 AWS 工具組 s 以更輕鬆地使用AWS 服務。AWS Toolkit for IntelliJ和AWS Toolkit for Eclipse是兩個可用於 Java 開發的工具組。

> **Important**
>
> 本設定區段中的指示假設您或組織使用 IAM 身分中心。如果您的組織使用獨立於 IAM 身分中心運作的外部身分識別提供者，請瞭解如何取得 SDK for Java 的臨時登入資料。請依照下列指示將暫時認證新增至`~/.aws/credentials`檔案。
> 如果您的身分識別提供者會自動將臨時認證新增至`~/.aws/credentials`檔案，請確定設定檔名稱為，這`[default]`樣您就不需要為 SDK 或提供設定檔名稱AWS CLI。

AWS存取入口網站的登入功能

AWS存取入口網站是您手動登入 IAM 身分中心的網路位置。網址的格式為`d-xxxxxxxxxx.awsapps.com/start`或*`your_subdomain`*`.awsapps.com/start`。

如果您不熟悉AWS存取入口網站，請按照 AWS SDK 和工具參考指南中 IAM 身分中心身份驗證主題的步驟 1 中的帳戶存取指引進行操作。請勿遵循步驟 2，因為 AWS SDK for Java 1.x 不支援自動權杖重新整理，以及自動擷取步驟 2 所描述之 SDK 的臨時認證。

設定共用設定檔

共用設定檔位於您的開發工作站上，並包含所有 AWS SDK 和 AWS Command Line Interface (CLI) 使用的基本設定。共用設定檔可以包含許多設定，但這些指示會設定使用 SDK 所需的基本元素。

設定共享`config`檔案

下列範例顯示共用`config`檔案的內容。

```
[default]
region=us-east-1
output=json
```

出於開發目的，請使用最AWS 區域接近您計劃運行代碼的位置。有關要在`config`文件中使用的區域代碼的列表，請參閱指Amazon Web Services 一般參考南。輸出格式的`json`設定是數個可能值之一。

請遵循本節中的指導來建立`config`檔案。

設定 SDK 的臨時認證

透過存取入口網站存取AWS 帳戶和 IAM 角色後，請使用要AWS存取 SDK 的臨時登入資料來設定您的開發環境。

使用臨時身份證明設置本地**credentials**文件的步驟

1. 建立共用credentials檔案。
2. 在credentials檔案中，貼上下列預留位置文字，直到您貼上工作中的暫時認證為止。

```
[default]
aws_access_key_id=<value from AWS access portal>
aws_secret_access_key=<value from AWS access portal>
aws_session_token=<value from AWS access portal>
```

3. 儲存檔案。檔案現在~/.aws/credentials應該存在於您的本機開發系統上。如果未指定特定的具名描述檔，則此檔案包含 SDK for Java 使用的 [預設] 設定檔。
4. 登入 AWS 存取入口網站。
5. 依照手動憑證重新整理標題下的指示，從AWS存取入口網站複製 IAM 角色憑證。
   a. 針對連結指示中的步驟 4，請選擇針對開發需求授予存取權的 IAM 角色名稱。此角色通常具有類似PowerUserAccess或開發人員的名稱。
   b. 針對步驟 7，請選取手動將設定檔案新增至AWS憑證檔案。
6. 將複製的認證貼到您的本機credentials檔案中，並移除貼上的任何設定檔名稱。您的檔案應類似以下內容：

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONG
```

7. 儲存檔credentials案

Java SDK 在建立服務用戶端並用於每個要求時，會存取這些臨時憑證。在步驟 5a 中選擇的 IAM 角色設定會決定臨時憑證的有效時長。最長持續時間為 12 小時。

臨時憑證過期後，請重複步驟 4 至 7。

安裝 Java 開發環境

AWS SDK for Java需要 J2SE 開發套件 6.0 或更新版本。您可以從 http://www.oracle.com/technetwork/java/javase/downloads/ 下載最新的 Java 軟體。

> **Important**
>
> Java 1.6 版 (JS2E 6.0) 並未內建對 SHA256 簽署的 SSL 憑證的支援，這些憑證在 2015 年 9 月 30 日AWS之後的所有 HTTPS 連線都需要這些憑證。
> Java 1.7 或更新版本已使用更新的憑證封裝，不受此問題影響。

選擇 JVM

為使伺服器式應用程式搭配AWS SDK for Java可獲得最佳效能，建議您使用 64 位元版本的 Java 虛擬機器 (JVM)。即使您在執行時間指定 `-Client` 選項，這個 JVM 仍只在伺服器模式下執行。

在執行階段使用 32 位元版本的 JVM 與`-Server`選項應該可提供與 64 位元 JVM 相比的效能。

如何獲得AWS SDK for Java

先決條件

若要使用 AWS SDK for Java，您必須擁有：

- 您必須能夠登入中提供的AWS存取入口網站AWS IAM Identity Center。
- 一個合適的 Java 安裝。
- 在您的本機共用`credentials`檔案中設定的臨時認證。

有關如何設置使用 Java SDK 的說明，請參閱the section called “基本設定”主題。

使用構建工具來管理 SDK for Java 的依賴關係

我們建議您使用阿帕奇 Maven 或搖籃與您的項目來訪問 SDK for Java 所需的依賴關係。 本節說明如何使用這些工具。

下載並解壓縮 GSpS/ 解壓縮 GSpS/

我們建議您使用構建工具來訪問項目的 SDK，但是您可以下載最新版本的 SDK 的預構建 jar。

> **Note**
>
> 如需如何下載及建置先前版本 SDK 的詳細資訊，請參閱安裝 SDK 的舊版。

1. 從下載 SDK://sdk-for-java. 亞馬遜網站服aws-java-sdk務.
2. 下載 SDK 後，將內容解壓縮到本機目錄中。

SDK 包含以下目錄：

- `documentation`-包含 API 文檔（也可在網絡上獲得：AWS SDK for JavaAPI 參考）。
- `lib`-包含 SDK.jar 文件。
- `samples`-包含示範如何使用 SDK 的工作範例程式碼。
- `third-party/lib`-包含 SDK 所使用的第三方程式庫，例如 Apache 公共資源記錄、AspectJ 和 Spring 框架。

要使用 SDK，請將`lib`和`third-party`目錄的完整路徑添加到構建文件中的依賴項中，然後將它們添加`CLASSPATH`到 Java 中以運行代碼。

從原始碼建置先前版本的 SDK (不建議使用)

只有完整 SDK 的最新版本以預先構建的形式提供作為可下載的 jar。但是，您可以使用 Apache Maven（開源）構建 SDK 的以前版本。Maven 將下載所有必要的依賴關係，在一個步驟中構建和安裝 SDK。如需安裝說明和更多資訊，請造訪 http://maven.apache.org/。

1. 轉到 SDK 的 GitHub 頁面：AWS SDK for Java（GitHub）。
2. 選擇與所需 SDK 版本號碼對應的標籤。例如：`1.6.10`。
3. 按一下 [下載 ZIP] 按鈕以下載您選取的 SDK 版本。
4. 將檔案解壓縮至開發系統上的目錄。在許多系統上，您可以使用圖形文件管理器來執行此操作，或在終端機窗口中使用該`unzip`實用程序。
5. 在終端機視窗中，導覽至您將 SDK 來源解壓縮的目錄所在位置。
6. 使用以下命令構建並安裝 SDK（需要 Maven）：

   ```
   mvn clean install -Dgpg.skip=true
   ```

   產生的 `.jar` 檔案內建至 `target` 目錄。
7. (選擇性) 使用下列命令來建置 API 參考文件：

   ```
   mvn javadoc:javadoc
   ```

文件內建於目`target/site/apidocs/`錄中。

使用建置工具

構建工具的使用有助於管理 Java 項目的開發。有幾種構建工具可用，但我們展示瞭如何使用兩種流行的構建工具-Maven 和 Gradle 啟動和運行。本主題說明如何使用這些建置工具來管理專案所需之 Java 相依性的 SDK。

主題

- 使用軟體開發套件搭配 Apache Maven
- 使用軟體開發套件搭配 Gradle

使用軟體開發套件搭配 Apache Maven

您可以使用 Apache Maven 設定及建立AWS SDK for Java的專案或建立開發套件本身。

> **Note**
>
> 您必須已安裝 Maven 才能使用本主題中的指導方針。如果尚未安裝，請造訪 http://maven.apache.org/ 進行下載和安裝。

建立新 Maven 軟體開發套件

要創建一個基本的 Maven 包，打開一個終端（命令行）窗口並運行：

```
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=org.example.basicapp \
  -DartifactId=myapp
```

將 org.example.basicapp 替換為應用程序的完整包名稱空間，並將 myapp 替換為項目名稱（這將成為項目的目錄名稱）。

根據預設，會使用快速入門原型為您建立專案範本，這對於許多專案來說都是一個很好的起點。還有更多可用的原型；請訪問 Maven 原型頁面以獲取打包的原型列表。您可以將 `-DarchetypeArtifactId` 引數新增到 `archetype:generate` 命令，選擇使用特定原型。例如：

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-webapp \
  -DgroupId=org.example.webapp \
  -DartifactId=mywebapp
```

> **Note**
>
> 有關創建和配置項目的更多信息在 Maven 入門指南中提供。

將軟體開發套件作為 Maven 依存項目

要在項目AWS SDK for Java中使用，您需要將其聲明為項目pom.xml文件中的依賴項。從版本 1.9.0 開始，您可以導入單個組件或整個 SDK。

指定個別的 SDK 模組

要選擇AWS SDK for Java單獨的 SDK 模塊，請使用 Maven 的材料清單（BOM），這將確保您指定的模塊使用相同版本的 SDK，並且它們彼此兼容。

要使用 BOM，請將<dependencyManagement>部分添加到應用程序的pom.xml文件中，添加aws-java-sdk-bom為依賴項並指定要使用的 SDK 版本：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.1000</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

若要檢視 Maven 中央提供的最新AWS SDK for Java BOM 版本，請造訪：https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-bom。您也可以使用此頁面來查看 BOM 管理哪些模組 (相依性)，您可以在專pom.xml案檔案的<dependencies>區段中包含這些模組 (相依性)。

您現在可以從應用程式中使用的 SDK 中選取個別模組。由於您已經在 BOM 中宣告開發套件版本，所以不需要指定每個元件的版本編號。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-s3</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-dynamodb</artifactId>
  </dependency>
</dependencies>
```

您也可以參考，瞭解指AWS 程式碼範例目錄定的相依性要使用哪些相依性AWS 服務。請參閱特定服務範例下的 POM 檔案。例如，如果您對AWS S3 服務的相依性感興趣，請參閱上的完整範例 GitHub。（看看 /java/例子代碼/s3 下的 POM）。

匯入所有 SDK 模組

如果您想將整個 SDK 作為依賴項提取，請不要使用 BOM 方法，而只需在pom.xml如下所示中聲明它：

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk</artifactId>
    <version>1.11.1000</version>
  </dependency>
</dependencies>
```

建立專案

一旦你有你的項目設置，你可以使用 Maven 的package命令來構建它：

```
mvn package
```

這將在target目錄中創建您的0jar文件。

使用 Maven 軟體開發套件

您可以使用 Apache Maven 從來源建置軟體開發套件從來源。為此，請從下載 SDK 代碼 GitHub，在本地解壓縮，然後執行以下 Maven 命令：

```
mvn clean install
```

使用軟體開發套件搭配 Gradle

若要管理您的 SDK 相依性搖籃專案中，匯入的 Maven 物料清單AWS SDK for Java進入應用程序的`build.gradle`文件。

> **Note**
>
> 在下列範例中，取代*1.12.529*在具有有效版本的構建文件中AWS SDK for Java。在「」中尋找最新版本Maven 中央存儲庫。

搖籃 4.6 或更高版本的項目設置

自搖籃 4.6 以來，您可以使用 Gradle 的改進 POM 支持功能通過聲明對 BOM 的依賴關係來導入材料單（BOM）文件。

1. 如果您使用的是搖籃 5.0 或更高版本，請跳到步驟 2。否則，請啟用改進了支持功能中的`settings.gradle`文件。

   ```
   enableFeaturePreview('IMPROVED_POM_SUPPORT')
   ```

2. 將材料表加入至依賴應用程式的部分`build.gradle`文件。

   ```
   ...
   dependencies {
       implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')

       // Declare individual SDK dependencies without version
       ...
   }
   ```

3. 指定要在 dependencies (相依性) 區段中使用的開發套件模組。例如，下列項目包含的相依性 Amazon Simple Storage Service(Amazon S3).

```
...
dependencies {
    implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')
    implementation 'com.amazonaws:aws-java-sdk-s3'
    ...
}
```

Gradle 會使用 BOM 的資訊，自動解析您開發套件相依性的正確版本。

以下是包含 Amazon S3 相依性的完整 `build.gradle` 檔案範例。

```
group 'aws.test'
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.12.529')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

> **Note**
>
> 在上一個範例中，取代的相依性Amazon S3與的依賴關係AWS您將在專案中使用的服務。由管理的模塊（依賴關係）AWS SDK for Java材料表列示於Maven 中央存儲庫。

搖籃版本早於 4.6 的項目設置

早於 4.6 的搖籃版本缺少本地 BOM 支持。若要管理AWS SDK for Java您的項目的依賴關係，使用 Spring 的依賴管理插件讓搖籃導入 SDK 的 Maven BOM。

1. 將依賴管理插件添加到您的應用程序`build.gradle`文件。

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath "io.spring.gradle:dependency-management-plugin:1.0.9.RELEASE"
    }
}

apply plugin: "io.spring.dependency-management"
```

2. 新增 BOM 到檔案的 dependencyManagement 區段。

```
dependencyManagement {
    imports {
        mavenBom 'com.amazonaws:aws-java-sdk-bom:1.12.529'
    }
}
```

3. 指定您將在中使用的 SDK 模組依賴部分。例如，以下內容包含 Amazon S3 的相依性。

```
dependencies {
    compile 'com.amazonaws:aws-java-sdk-s3'
}
```

Gradle 會使用 BOM 的資訊，自動解析您開發套件相依性的正確版本。

以下是包含 Amazon S3 相依性的完整 `build.gradle` 檔案範例。

```
group 'aws.test'
version '1.0'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
  mavenCentral()
}

buildscript {
```

```
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath "io.spring.gradle:dependency-management-plugin:1.0.9.RELEASE"
  }
}

apply plugin: "io.spring.dependency-management"

dependencyManagement {
  imports {
    mavenBom 'com.amazonaws:aws-java-sdk-bom:1.12.529'
  }
}

dependencies {
  compile 'com.amazonaws:aws-java-sdk-s3'
  testCompile group: 'junit', name: 'junit', version: '4.11'
}
```

> **Note**
>
> 在上一個範例中，取代的相依性Amazon S3與的依賴關係AWS您將在項目中使用的服務。由管理的模塊（依賴關係）AWS SDK for Java材料表列示於Maven 中央存儲庫。

如需有關使用 BOM 指定 SDK 相依性的詳細資訊，請參閱使用 SDK 與阿帕奇 Maven。

設置AWS臨時憑據並AWS 區域進行開發

若要使用連線到任何支援的服務AWS SDK for Java，您必須提供AWS暫時認證。AWSSDK 和 CLI 使用提供者鏈結在許多不同的地方尋找AWS暫時登入資料，包括系統/使用者環境變數以及本機AWS組態檔案。

本主題提供有關使用設定AWS暫時認證以進行本機應用程式開發的基本資訊AWS SDK for Java。如果您需要設定憑證以在 EC2 執行個體中使用，或者您使用 Eclipse IDE 進行開發，請參閱下列主題：

- 使用 EC2 執行個體時，請建立 IAM 角色，然後將該角色授予 EC2 執行個體存取權，如在上使用 IAM 角色授予AWS資源存取權中所示Amazon EC2。

- 在 Eclipse 中使用設定AWS認證AWS Toolkit for Eclipse。如需詳細資訊，請參閱AWS Toolkit for Eclipse使用者指南中的設定AWS身分證明。

設定暫時性憑入

您可以透過數種方AWS SDK for Java式設定臨時認證，但建議的方法如下：

- 在您本機系統的AWS認證設定檔檔案中設定臨時身份證明，位於：
  - `~/.aws/credentials`在 Linux、macOS 或 Unix
  - Windows 上的 `C:\Users\USERNAME\.aws\credentials`

  如需如何取得暫時登入資料的指示，請參閱本指南the section called “設定 SDK 的臨時認證”中的內容。
- 設定`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`、和`AWS_SESSION_TOKEN`環境變數。

  若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 ：

  ```
  export AWS_ACCESS_KEY_ID=your_access_key_id
  export AWS_SECRET_ACCESS_KEY=your_secret_access_key
  export AWS_SESSION_TOKEN=your_session_token
  ```

  若要在 Windows 上設定這些變數，請使用 ：

  ```
  set AWS_ACCESS_KEY_ID=your_access_key_id
  set AWS_SECRET_ACCESS_KEY=your_secret_access_key
  set AWS_SESSION_TOKEN=your_session_token
  ```
- 對於 EC2 執行個體，請指定 IAM 角色然後提供您的 EC2 執行個體存取權給該角色。如需其運作方式的詳細討論，請參閱 Linux 執行個體Amazon EC2使用者指南Amazon EC2中的 IAM 角色。

使用其中一種方法設定AWS暫時認證後，會使用預設認證提供者鏈結自動載入它們。AWS SDK for Java如需有關在 Java 應用程式中使用AWS認證的詳細資訊，請參閱使用AWS認證。

重新整理 IMDS 登入資料

無論登入資料過期時間為何，AWS SDK for Java 支援每 1 分鐘在背景中重新整理 IMDS 登入資料。這可讓您更頻繁地重新整理認證，並減少未到達 IMDS 影響感知AWS可用性的機會。

```
1. // Refresh credentials using a background thread, automatically every minute. This
will log an error if IMDS is down during
2. // a refresh, but your service calls will continue using the cached credentials
until the credentials are refreshed
3. // again one minute later.
4.
5. InstanceProfileCredentialsProvider credentials =
6.     InstanceProfileCredentialsProvider.createAsyncRefreshingProvider(true);
7.
8. AmazonS3Client.builder()
9.              .withCredentials(credentials)
10.              .build();
11.
12. // This is new: When you are done with the credentials provider, you must close it
to release the background thread.
13. credentials.close();
```

設定AWS 區域

您應該設定一個預設值AWS 區域，該預設值將用於存取AWS服務AWS SDK for Java。為了獲得最佳的網路效能，請選擇地理位置上靠近您 (或您的客戶) 的區域。如需每個服務的區域清單，請參閱Amazon Web Services一般參考中的區域和端點。

> **Note**
>
> 如果您沒有選擇一個區域，那麼 us-east-1 將默認使用。

您可以使用類似的方法來設定認證來設定預設AWS區域：

- AWS 區域在本地系統上的AWS配置文件中設置，位於：
  - Linux、macOS 或 Unix
  - C:\Users\USERNAME\ .aws\ 在視窗上配置

  此檔案應該包含下列格式的行：

  +

```
[default]
region = your_aws_region
```

+

將您想要的AWS 區域 (例如 , 「美國東部 -1」) 替換為您的 _aws_區域。

- 設定 AWS_REGION 環境變數。

  若為 Linux、macOS 或 Unix，使用 Linux、macOS 或 Unix

  ```
  export AWS_REGION=your_aws_region
  ```

  在 Windows 上，使用 ：

  ```
  set AWS_REGION=your_aws_region
  ```

  您的 _aws_區域是所需的AWS 區域名稱。

使用 AWS SDK for Java

本節提供有關使用程式設計的重要一般資訊，這 AWS SDK for Java 些資訊適用於您可能與 SDK 搭配使用的所有服務。

如需服務特定程式設計資訊和範例 (適用於 Amazon EC2 Amazon S3 Amazon SWF、等等)，請參閱程式AWS SDK for Java 碼範例。

主題

- 與 AWS 開發的最佳做法 AWS SDK for Java
- 建立服務用戶端
- 提供臨時登入資料給 AWS SDK for Java
- AWS 區域 選擇
- 例外狀況處理
- 非同步程式設計
- 記錄 AWS SDK for Java 呼叫
- 客戶端組態
- 存取控制政策
- 設定 DNS 名稱查詢的 JVM TTL
- 啟用指標 AWS SDK for Java

與 AWS 開發的最佳做法 AWS SDK for Java

下列最佳作法可協助您避免在開發 AWS 應用程式時發生問題或發生問題 AWS SDK for Java。我們已經按服務組織了最佳實踐。

S3

避免 ResetExceptions

當您使用串流 (透 Amazon S3 過用`AmazonS3`戶端或`TransferManager`) 將物件上傳至時，可能會遇到網路連線或逾時問題。根據預設， AWS SDK for Java 嘗試重試失敗的傳輸，方法是在傳輸開始之前標記輸入串流，然後在重試之前將其重設。

如果串流不支援標記和重設，當發生暫ResetException時性失敗並啟用重試時，SDK 會擲回一個。

最佳做法

建議您使用支援標記和重設作業的串流。

避免 a 的最可靠的方法ResetException是通過使用 File 或提供數據 FileInputStream，該文件 AWS SDK for Java 可以處理而不受標記和復位限制的限制。

如果串流不是FileInputStream但支援標記和重設，您可以使用的`setReadLimit`方法來設定標記限制RequestClientOptions。其預設值為 128 KB。將讀取限制值設置為大於流大小的一個字節將可靠地避免 ResetException.

例如，如果串流的預期大小上限為 100,000 個位元組，請將讀取限制設定為 100,001 (100,000 + 1) 位元組。標記和重置將始終適用於 100,000 個字節或更少。請注意，這可能會導致某些流緩衝該字節數量到內存中。

建立服務用戶端

若要向其發出要求 Amazon Web Services，請先建立服務用戶端物件。推薦的方法是使用服務客戶端構建器。

每個人都 AWS 服務 有一個服務接口，其中包含服務 API 中每個操作的方法。例如，DynamoDB 的服務介面名AmazonDynamo為「資料庫用戶端」。每個服務接口都有一個相應的客戶端構建器，您可以使用它來構建服務接口的實現。的用戶端建置器類別名稱 DynamoDB 為 AmazonDynamoDB ClientBuilder。

取得用戶端建置器

若要取得用戶端建置器的執行個體，請使用靜態工廠方法`standard`，如下列範例所示。

```
AmazonDynamoDBClientBuilder builder = AmazonDynamoDBClientBuilder.standard();
```

擁有構建器後，您可以通過在構建器 API 中使用許多流利的 setter 來自定義客戶端的屬性。例如，您可以設定自訂區域和自訂認證提供者，如下所示。

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.standard()
                        .withRegion(Regions.US_WEST_2)
                        .withCredentials(new ProfileCredentialsProvider("myProfile"))
                        .build();
```

> **Note**
>
> 流暢的`withXXX`方法返回對`builder`象，以便您可以鏈接方法調用以方便並獲得更易讀的代碼。設定您要的屬性後，您可以呼叫 `build` 方法來建立用戶端。一旦客戶端被創建，它是不可變的，任何調用`setRegion`或`setEndpoint`將失敗。

生成器可以創建具有相同配置的多個客戶端。在編寫應用程序時，請注意構建器是可變的，而不是線程安全的。

下面的代碼使用生成器作為客戶端實例的工廠。

```
public class DynamoDBClientFactory {
    private final AmazonDynamoDBClientBuilder builder =
        AmazonDynamoDBClientBuilder.standard()
            .withRegion(Regions.US_WEST_2)
            .withCredentials(new ProfileCredentialsProvider("myProfile"));

    public AmazonDynamoDB createClient() {
        return builder.build();
    }
}
```

構建器還公開了流利的RequestHandler設置者ClientConfiguration和 RequestMetricCollector，以及 2 的自定義列表。

以下是覆寫所有可配置屬性的完整範例。

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCredentials(new ProfileCredentialsProvider("myProfile"))
        .withClientConfiguration(new ClientConfiguration().withRequestTimeout(5000))
        .withMetricsCollector(new MyCustomMetricsCollector())
        .withRequestHandlers(new MyCustomRequestHandler(), new
 MyOtherCustomRequestHandler)
        .build();
```

建立非同步用戶端

每個服務（除外）都 AWS SDK for Java 具有異步（或異步）客戶端，並為 Amazon S3每個服務提供相應的異步客戶端構建器。

若要使用預設值建立非同步 DynamoDB 用戶端 ExecutorService

```
AmazonDynamoDBAsync ddbAsync = AmazonDynamoDBAsyncClientBuilder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCredentials(new ProfileCredentialsProvider("myProfile"))
        .build();
```

除了同步 (或同步處理) 用戶端產生器支援的設定選項之外，非同步用戶端可讓您設定自訂ExecutorFactory來變更非同步用戶端使用ExecutorService的設定選項。 ExecutorFactory是一個函數接口，因此它與 Java 8 lambda 表達式和方法引用進行交互操作。

使用自訂執行程式建立非同步用戶端

```
AmazonDynamoDBAsync ddbAsync = AmazonDynamoDBAsyncClientBuilder.standard()
            .withExecutorFactory(() -> Executors.newFixedThreadPool(10))
            .build();
```

使用 DefaultClient

同步和異步客戶端構建器都有另一個名為defaultClient的工廠方法。此方法會建立具有預設組態的服務用戶端，並使用預設提供者鏈結來載入認證和 AWS 區域. 如果無法從應用程式執行的環境判斷登入資料或區域，則對 defaultClient 的呼叫失敗。請參閱使用 AWS 認證和AWS 區域 選擇，以取得有關如何確定認證和區域的詳細資訊。

建立預設服務用戶端

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
```

用戶端生命週期

SDK 中的服務客戶端是線程安全的，為了獲得最佳性能，您應該將它們視為長壽命對象。每個用戶端都有自己的連線集區資源。在不再需要用戶端時，明確關閉用戶端，以避免資源洩漏。

若要明確關閉用戶端，請呼叫方shutdown法。呼叫之後shutdown，所有用戶端資源都會釋放，且用戶端無法使用。

關閉用戶端

```
AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
```

```
ddb.shutdown();
// Client is now unusable
```

提供臨時登入資料給 AWS SDK for Java

若要向其提出要求 Amazon Web Services，您必須提供 AWS 暫時登入資料， AWS SDK for Java 以便在呼叫服務時使用。您可採用以下方式：

- 使用預設登入資料供應者鏈結 (建議)。
- 使用特定的登入資料供應者或供應者鏈結 (或建立您自己的項目)。
- 在代碼中自己提供臨時憑據。

使用預設登入資料供應商鏈結

當您在未提供任何引數的情況下初始化新的服務用戶端時，會 AWS SDK for Java 嘗試使用 D efault AWSCredentialsProviderChain 類別實作的預設認證提供者鏈來尋找暫時認證。預設登入資料供應者鏈結會依以下順序尋找登入資料：

1. 環境變數-AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY、、和AWS_SESSION_TOKEN。會 AWS SDK for Java 使用EnvironmentVariableCredentialsProvider類別來載入這些認證。
2. Java 系統屬性-aws.accessKeyId aws.secretKey、和aws.sessionToken。使 AWS SDK for Java 用載SystemPropertiesCredentialsProvider入這些認證。
3. 來自環境或容器的 Web 身份權杖認證。
4. 默認憑據配置文件文件-通常位於~/.aws/credentials（位置可能因平台而異），並由許多 AWS SDK 共享。 AWS CLI使 AWS SDK for Java 用載ProfileCredentialsProvider入這些認證。

   您可以使用提供的aws configure指令建立認證檔案 AWS CLI，也可以使用文字編輯器編輯檔案來建立認證檔案。如需認證檔案格式的相關資訊，請參閱AWS 認證檔案格式。
5. Amazon ECS 容器登入資料-如果設定了環境變數AWS_CONTAINER_CREDENTIALS_RELATIVE_URI，則會從 Amazon ECS 載入。使 AWS SDK for Java 用載ContainerCredentialsProvider入這些認證。您可以指定此值的 IP 位址。
6. 執行個體設定檔登入資料-用於 EC2 執行個體，並透過 Amazon EC2 中繼資料服務提供。使 AWS SDK for Java 用載InstanceProfileCredentialsProvider入這些認證。您可以指定此值的 IP 位址。

> **Note**
>
> 只有在未設定執行個體設定檔證明資料時才AWS_CONTAINER_CREDENTIALS_RELATIVE_URI會使用。如需詳細ContainerCredentialsProviderWrapper資訊，請參閱 EC2。

設定臨時認證

為了能夠使用 AWS 臨時身份證明，它們必須至少設置在先前的位置之一。如需設定登入資料的相關資訊，請參閱下列主題：

- 若要在環境中或預設認證設定檔檔案中指定認證，請參閱the section called “設定暫時性憑入”。
- 若要設定 Java 系統屬性，請參閱官方 Java 教學課程網站的系統屬性教學課程。
- 若要在 EC2 執行個體上設定和使用執行個體設定檔登入資料，請參閱在上使用 IAM 角色授予 AWS 資源存取權 Amazon EC2。

設定替代認證設定檔

依預設 AWS SDK for Java 會使用預設設定檔，但有多種方法可以自訂來自認證檔案的設定檔。

您可以使用 AWS 設定檔環境變數來變更 SDK 載入的設定檔。

例如，在 Linux、macOS 或 Unix 上，您可以執行下列指令，將設定檔變更為「myProfile」。

```
export AWS_PROFILE="myProfile"
```

在視窗上，您可以使用以下內容。

```
set AWS_PROFILE="myProfile"
```

設定AWS_PROFILE環境變數會影響所有官方支援的 AWS SDK 和工具 (包括 AWS CLI 和 AWS Tools for Windows PowerShell) 的認證載入。若只要變更 Java 應用程式的設定檔，您可以aws.profile改用 system 屬性。

> **Note**
>
> 環境變數會優先於系統屬性。

設定替代認證檔案位置

會從預設認證檔案位置自動 AWS SDK for Java 載入 AWS 臨時認證。不過，您也可以設定 AWS_CREDENTIAL_PROFILES_FILE 環境變數搭配登入資料檔案的完整路徑，來指定位置。

您可以使用此功能暫時變更 AWS SDK for Java 尋找認證檔案的位置 (例如，透過使用指令行設定此變數)。或者，您也可以在使用者或系統環境中設定環境變數，來為該使用者進行變更或是進行全系統變更。

若要覆寫預設登入資料檔案位置

- 將AWS_CREDENTIAL_PROFILES_FILE環境變數設定為 AWS 認證檔案的位置。
  - 在 Linux、macOS 系統或 Unix 上，使用：

    ```
    export AWS_CREDENTIAL_PROFILES_FILE=path/to/credentials_file
    ```

  - 在視窗上，使用：

    ```
    set AWS_CREDENTIAL_PROFILES_FILE=path/to/credentials_file
    ```

Credentials檔案格式

遵循本指南基本設定中的指示，您的認證檔案應具有下列基本格式。

```
[default]
aws_access_key_id=<value from AWS access portal>
aws_secret_access_key=<value from AWS access portal>
aws_session_token=<value from AWS access portal>

[profile2]
aws_access_key_id=<value from AWS access portal>
aws_secret_access_key=<value from AWS access portal>
aws_session_token=<value from AWS access portal>
```

設定檔名稱在方括號中指定 (例如 [default])，後接該設定檔的可設定欄位，以索引鍵值組形式表示。您的檔案中可以有多個設定credentials檔，您可以使用選取aws configure --profile *PROFILE_NAME* 要設定的設定檔來新增或編輯這些設定檔。

您可以指定其他欄位，例如metadata_service_timeout、和metadata_service_num_attempts。這些檔案無法透過 CLI 進行配置 — 如果您想要使用它們，您必須手動編輯檔案。如需有關規劃檔及其可用欄位的詳細資訊，請參閱《 AWS Command Line Interface 使用指南》 AWS Command Line Interface中的〈配置〉。

載入認證

設定臨時認證之後，SDK 會使用預設認證提供者鏈結載入它們。

要做到這一點，你實例化一個 AWS 服務 客戶端，而不明確提供認證給生成器，如下所示。

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withRegion(Regions.US_WEST_2)
                        .build();
```

指定認證提供者或提供者鏈結

您可以使用用戶端建置器，指定與預設登入資料供應者鏈結不同的登入資料供應者。

您可以將認證提供者或提供者鏈結的執行個體提供給用戶端產生器，以接AWSCredentialsProvider口作為輸入。下列範例說明如何具體使用環境登入資料。

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withCredentials(new EnvironmentVariableCredentialsProvider())
                        .build();
```

如需提供的憑證 AWS SDK for Java提供者和提供者鏈結的完整清單，請參閱.
AWSCredentialsProvider

> **Note**
>
> 您可以使用此技巧來提供認證提供者或提供者鏈結，方法是使用您自己實作AWSCredentialsProvider介面的認證提供者或子類別化的認證提供者。AWSCredentialsProviderChain

明確指定臨時認證

如果預設認證鏈或特定或自訂提供者或提供者鏈結不適用於您的程式碼，您可以設定明確提供的認證。如果您已使用擷取暫時認證 AWS STS，請使用此方法指定要 AWS 存取的認證。

1. 實例化BasicSessionCredentials類，並提供 SDK 將用於連接的 AWS 訪問 AWS 密鑰，秘密密鑰和會 AWS 話令牌。
2. AWSStaticCredentialsProvider使用AWSCredentials對象創建一個。
3. 使用 AWSStaticCredentialsProvider 設定用戶端建置器並建置用戶端。

以下是範例。

```
BasicSessionCredentials awsCreds = new BasicSessionCredentials("access_key_id",
 "secret_key_id", "session_token");
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                        .withCredentials(new AWSStaticCredentialsProvider(awsCreds))
                        .build();
```

詳細資訊

- 註冊 AWS 並建立 IAM 使用者
- 設定 AWS 認證和開發區域
- 使用 IAM 角色授予以下 AWS 資源的存取權 Amazon EC2

AWS 區域 選擇

區域可讓您存取實際位於特定地理區域的 AWS 服務。這對於備援以及讓您的資料和應用程式在靠近您和您的使用者存取位置附近執行，都很有用。

檢查區域中的服務可用性

若要查看某個特定地區 AWS 服務 是否有提供，請在您要使用的地區上使用該isServiceSupported方法。

```
Region.getRegion(Regions.US_WEST_2)
    .isServiceSupported(AmazonDynamoDB.ENDPOINT_PREFIX);
```

請參閱 Region 類別說明文件，瞭解您可以指定的區域，並使用服務的端點前置詞進行查詢。每個服務的端點前綴都在服務界面中定義。例如， DynamoDB 端點前綴在 AmazonDynamoDB 中定義。

選擇區域

從 1.4 版開始 AWS SDK for Java，您可以指定區域名稱，SDK 會自動為您選擇適當的端點。若要自行選擇端點，請參閱選擇特定端點。

若要明確設定區域，我們建議您使用 Region 列舉。這是所有公開可用區域的列舉。要從枚舉中創建具有區域的客戶端，請使用以下代碼。

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withRegion(Regions.US_WEST_2)
                    .build();
```

如果您嘗試使用的區域不在Regions枚舉中，則可以使用代表區域名稱的字符串設置區域。

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withRegion("{region_api_default}")
                    .build();
```

> **Note**
>
> 使用建置器建置用戶端後，這是不可變的，而且區域無法變更。如果您為相同的服務使用多個 AWS 區域 用戶端，您應該建立多個用戶端 — 每個區域一個。

選擇特定端點

建立用 AWS 戶端時呼叫withEndpointConfiguration方法，可將每個用戶端設定為使用區域內的特定端點。

例如，若要設定用 Amazon S3 戶端使用歐洲 (愛爾蘭) 區域，請使用下列程式碼。

```
AmazonS3 s3 = AmazonS3ClientBuilder.standard()
     .withEndpointConfiguration(new EndpointConfiguration(
          "https://s3.eu-west-1.amazonaws.com",
          "eu-west-1"))
     .withCredentials(CREDENTIALS_PROVIDER)
     .build();
```

如需所有 AWS 服務的目前區域清單及其對應端點，請參閱區域和端點。

從環境中自動確定區域

> **Important**
>
> 本節僅在使用客戶端構建器訪問 AWS 服務時適用。 AWS 使用客戶端構造函數創建的客戶端不會自動從環境中確定區域，而是使用默認的 SDK 區域（UseAst1）。

在 Amazon EC2 或 Lambda 上執行時，您可能想要將用戶端設定為使用與執行程式碼相同的區域。這會讓您的程式碼與其執行環境分離，也更容易將應用程式部署到多個區域，以降低延遲或提供備援。

您必須使用用戶端建置器，讓 SDK 自動偵測您的程式碼執行所在的區域。

若要使用預設的登入資料/區域供應者鏈結來從環境判斷區域，請使用用戶端建置器的 `defaultClient` 方法。

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
```

這與使用`standard`後跟的相同`build`。

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .build();
```

如果您未使用這些`withRegion`方法明確設定區域，SDK 會諮詢預設的區域提供者鏈結，以嘗試決定要使用的區域。

預設區域供應者鏈結

以下是區域查詢程序：

1. 通過使用`withRegion`或構建器本身設置`setRegion`的任何明確區域都優先於其他任何內容。
2. 檢查 `AWS_REGION` 環境變數。如果有設定，會使用該區域來設定用戶端。

> **Note**
>
> 此環境變數由 Lambda 容器設定。

3. SDK 會檢查 AWS 共用設定檔案 (通常位於~/.aws/config)。如果 region 屬性存在，開發套件會予以使用。
   - AWS_CONFIG_FILE 環境變數可用於自訂共用組態檔的位置。
   - AWS_PROFILE環境變數或系aws.profile統屬性可用來自訂 SDK 載入的設定檔。
4. SDK 會嘗試使用 Amazon EC2 執行個體中繼資料服務來判斷目前執行中 Amazon EC2 執行個體的區域。
5. 如果開發套件在這個時候仍找不到區域，用戶端建立會失敗，並出現例外狀況。

在開發 AWS 應用程式時，常見的方法是使用共用組態檔案 (如使用預設認證提供者鏈結中所述) 來設定本機開發的區域，並依賴預設的區域提供者鏈來決定在 AWS 基礎結構上執行時的區域。這可大幅簡化用戶端建立並讓您的應用程式保持可攜式。

例外狀況處理

了解 AWS SDK for Java 拋出異常的方式和時間對於使用 SDK 構建高質量的應用程序非常重要。以下章節說明開發套件會擲回的不同例外狀況案例，以及如何正確處理這些狀況。

為什麼使用未檢查的例外狀況？

由於以下原因， AWS SDK for Java 使用運行時（或未選中）異常而不是檢查異常：

- 為了讓開發人員能夠更精確的控制他們想處理的錯誤，而非強制他們處理不在乎的例外情況 (因而使得程式碼過於冗長)
- 為了避免大型應用程式中已檢查例外狀況的固有擴展性問題

一般而言，已檢查例外狀況在小規模上可運作良好，但會隨著應用程式增長且更複雜而變得棘手。

如需有關使用已勾選和未核取例外狀況的詳細資訊，請參閱：

- 未核取的例外狀況-爭議
- 檢查異常的麻煩
- Java 檢查的異常是一個錯誤（這是我想要做的）

AmazonServiceException （和子類別）

AmazonServiceException是最常見的例外狀況，您在使用 AWS SDK for Java. 此例外表示來自的錯誤回應 AWS 服務。例如，如果您嘗試終止不存在的 Amazon EC2 實例，EC2 將返回一個錯誤響應，並且該錯誤響應的所有詳細信息將包含在拋出AmazonServiceException的實例中。在某些情況下，會擲回 AmazonServiceException 的子類別以讓開發人員透過 catch 區塊更精確的控制錯誤情況處理。

當您遇到時AmazonServiceException，您知道您的請求已成功發送到， AWS 服務 但無法成功處理。這可能是因為請求參數中的錯誤，或因為服務端的問題。

AmazonServiceException 為您提供資訊，例如：

- 傳回 HTTP 狀態碼
- 傳回 AWS 錯誤碼
- 來自該服務的詳細錯誤訊息
- AWS 失敗要求的要求識別碼

AmazonServiceException還包括有關失敗請求是調用者的錯誤（具有非法值的請求）還 AWS 服務是錯誤（內部服務錯誤）的信息。

AmazonClientException

AmazonClientException表示 Java 客戶端代碼中發生了一個問題，無論是在嘗試向其發送請求 AWS 或嘗試解析響應時 AWS。A 通AmazonClientException常比一個更嚴重AmazonServiceException，並且表示阻止用戶端對服務進行服務呼叫的主要問題。 AWS 例AmazonClientException如，當您嘗試在其中一個用戶端上呼叫作業時， AWS SDK for Java 會擲回如果沒有網路連線可用。

非同步程式設計

您可以使用同步或非同步方法來呼叫 AWS 服務上的作業。同步方法會封鎖您的執行緒執行，直到用戶端收到服務的回應。非同步方法會立即傳回，將控制權回歸給呼叫端執行緒，無需等待回應。

由於非同步方法會在有可用回應之前傳回，您需要一個方法在回應準備好時取得回應。 AWS SDK for Java 提供了兩種方式：未來的對象和回調方法。

爪哇期貨

異步方法在 AWS SDK for Java 返回一個包含 future 異步操作結果的 Future 對象。

調用該FutureisDone()方法以查看服務是否提供了響應對象。當響應準備就緒時，您可以通過調用該Futureget()方法獲取響應對象。您可以使用此機制來定期輪詢非同步作業的結果，而您的應用程式會繼續處理其他項目。

下面是一個異步操作的例子，它調用一個 Lambda 函數，接收一個可Future以容納一個InvokeResult對象的。只有在是之後才會擷取InvokeResult物isDone()件true。

```
import com.amazonaws.services.lambda.AWSLambdaAsyncClient;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import java.nio.ByteBuffer;
import java.util.concurrent.Future;
import java.util.concurrent.ExecutionException;

public class InvokeLambdaFunctionAsync
{
    public static void main(String[] args)
    {
        String function_name = "HelloFunction";
        String function_input = "{\"who\":\"SDK for Java\"}";

        AWSLambdaAsync lambda = AWSLambdaAsyncClientBuilder.defaultClient();
        InvokeRequest req = new InvokeRequest()
            .withFunctionName(function_name)
            .withPayload(ByteBuffer.wrap(function_input.getBytes()));

        Future<InvokeResult> future_res = lambda.invokeAsync(req);

        System.out.print("Waiting for future");
        while (future_res.isDone() == false) {
            System.out.print(".");
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                System.err.println("\nThread.sleep() was interrupted!");
                System.exit(1);
            }
        }
```

```
        try {
            InvokeResult res = future_res.get();
            if (res.getStatusCode() == 200) {
                System.out.println("\nLambda function returned:");
                ByteBuffer response_payload = res.getPayload();
                System.out.println(new String(response_payload.array()));
            }
            else {
                System.out.format("Received a non-OK response from {AWS}: %d\n",
                        res.getStatusCode());
            }
        }
        catch (InterruptedException | ExecutionException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        System.exit(0);
    }
}
```

非同步回呼

除了使用 Java Future 物件監視非同步要求的狀態之外，SDK 也可讓您實作使用AsyncHandler介面的類別。 AsyncHandler根據要求完成的方式，提供兩種呼叫的方法：onSuccess和onError。

回調接口方法的主要優點是，它使您無需輪詢對Future象以了解請求何時完成。相反，您的代碼可以立即開始其下一個活動，並依靠 SDK 在正確的時間調用您的處理程序。

```
import com.amazonaws.services.lambda.AWSLambdaAsync;
import com.amazonaws.services.lambda.AWSLambdaAsyncClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.amazonaws.handlers.AsyncHandler;
import java.nio.ByteBuffer;
import java.util.concurrent.Future;

public class InvokeLambdaFunctionCallback
{
    private class AsyncLambdaHandler implements AsyncHandler<InvokeRequest,
 InvokeResult>
    {
```

```
        public void onSuccess(InvokeRequest req, InvokeResult res) {
            System.out.println("\nLambda function returned:");
            ByteBuffer response_payload = res.getPayload();
            System.out.println(new String(response_payload.array()));
            System.exit(0);
        }

        public void onError(Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void main(String[] args)
    {
        String function_name = "HelloFunction";
        String function_input = "{\"who\":\"SDK for Java\"}";

        AWSLambdaAsync lambda = AWSLambdaAsyncClientBuilder.defaultClient();
        InvokeRequest req = new InvokeRequest()
            .withFunctionName(function_name)
            .withPayload(ByteBuffer.wrap(function_input.getBytes()));

        Future<InvokeResult> future_res = lambda.invokeAsync(req, new
 AsyncLambdaHandler());

        System.out.print("Waiting for async callback");
        while (!future_res.isDone() && !future_res.isCancelled()) {
            // perform some other tasks...
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                System.err.println("Thread.sleep() was interrupted!");
                System.exit(0);
            }
            System.out.print(".");
        }
    }
}
```

最佳實務

回呼執行

您的AsyncHandler實作會在非同步用戶端所擁有的執行緒集區內執行。簡短，快速執行的代碼在您的AsyncHandler實現中是最合適的。處理常式方法中的長時間執行或封鎖程式碼可能會對非同步用戶端所使用的執行緒集區造成爭用，而且可能會阻止用戶端執行要求。如果您有需要從回調開始的長時間運行任務，請讓回調在新線程或應用程序管理的線程池中運行其任務。

執行緒集區組態

中的非同步用戶端 AWS SDK for Java 提供預設執行緒集區，該執行緒集區應適用於大多數應用程式您可以實作自訂ExecutorService並將其傳遞給 AWS SDK for Java 非同步用戶端，以進一步控制執行緒集區的管理方式。

例如，您可以提供使用 Custom ThreadFactory來控制集區中執行緒的命名方式的ExecutorService實作，或記錄執行緒使用情況的其他相關資訊。

非同步存取

SDK 中的TransferManager類別為使用提供非同步支援 Amazon S3。 TransferManager管理異步上傳和下載，提供有關傳輸的詳細進度報告，並支持對不同事件的回調。

記錄 AWS SDK for Java 呼叫

與 Apache 的 AWS SDK for Java 共享資源日誌，這是一個抽象層，使得在運行時使用多個日誌系統中的任何一個進行檢測。

支援的記錄系統包括 Java Logging Framework 和 Apache Log4j 等等。本主題說明如何使用 Log4j。您可以使用開發套件的記錄功能，而無需更改您的應用程式程式碼。

若要進一步了解 Log4j，請參閱 Apache 網站。

> **Note**
>
> 本主題著重於 Log4j 的 1.x。Log4j2 不直接支持阿帕奇共享資源日誌記錄，但提供了一個適配器，使用 Apache 共享資源日誌記錄接口自動將調用記錄定向 Log4j2。如需詳細資訊，請參閱 Log4j2 文件中的維基資源記錄橋接器。

下載日誌 JAR

要使用 Log4j 的與 SDK，你需要從阿帕奇網站下載 Log4j 的 JAR。開發套件不包含 JAR。將 JAR 檔案複製到類別路徑上的位置。

Log4j 的使用一個配置文件，log4j. 屬性。範例組態檔案如下所示。將此配置文件複製到類路徑上的目錄中。Log4j 的 JAR 和 log4j. 屬性檔案不一定要位於相同的目錄中。

log4j.properties 組態檔會指定內容，例如記錄層級、將記錄輸出傳送到檔案或主控台的位置，以及輸出的格式。記錄層級是記錄器所產生輸出的精細度。Log4j 支援多個記錄階層的概念。每個階層的記錄層級都是獨立設定。以下兩個記錄階層可用於 AWS SDK for Java：

- 日誌. 日誌. 日誌.
- 日誌. 記錄器.

設定 Classpath

Log4j JAR 和 log4j. 屬性檔案都必須位於您的類別路徑中。如果您使用的是 Apache Ant，請在 Ant 文件中的`path`元素中設置類路徑。以下示例顯示了來自 Ant 文件的路徑元素，用於 SDK 中包含的示 Amazon S3 例。

```
<path id="aws.java.sdk.classpath">
  <fileset dir="../../third-party" includes="**/*.jar"/>
  <fileset dir="../../lib" includes="**/*.jar"/>
  <pathelement location="."/>
</path>
```

如果您是使用 Eclipse IDE，可以開啟選單並導覽到 Project (專案) | Properties (屬性) | Java Build Path (Java 建置路徑) 來設定 classpath。

服務特定錯誤與警告

我們建議您始終將「com.amazonaws」記錄器層次結構設置為「WARN」，以 catch 客戶端庫中的任何重要消息。例如，如果用 Amazon S3 戶端偵測到您的應用程式未正確關閉`InputStream`並且可能洩漏資源，則 S3 用戶端會透過警告訊息向記錄檔報告。這也可確保在用戶端處理請求或回應發生任何問題時，會記錄訊息。

下面的 log4j. 屬性文件設置rootLogger為警告，這會導致來自「com.amazonaws」層次結構中的所有記錄器的警告和錯誤消息被包含在內。或者，您可以明確地將 com.amazonaws 記錄器設置為警告。

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Or you can explicitly enable WARN and ERROR messages for the {AWS} Java clients
log4j.logger.com.amazonaws=WARN
```

請求/回應摘要記錄

對 a 的每個請求都 AWS 服務 會生成一個唯一的 AWS 請求 ID，如果您遇到有關如何處理請求的問題，該請求將 AWS 服務 非常有用。 AWS 要求識別碼可透過 SDK 中的例外狀況物件以程式設計方式存取，以程式設計方式存取任何失敗的服務呼叫，也可以透過「com.amazonaws.request」記錄程式中的 DEBUG 記錄檔層級進行報告。

下列 log4j. 屬性檔案會啟用要求和回應的摘要，包括 AWS 要求識別碼。

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Turn on DEBUG logging in com.amazonaws.request to log
# a summary of requests/responses with {AWS} request IDs
log4j.logger.com.amazonaws.request=DEBUG
```

以下為日誌輸出的範例。

```
2009-12-17 09:53:04,269 [main] DEBUG com.amazonaws.request - Sending
Request: POST https://rds.amazonaws.com / Parameters: (MaxRecords: 20,
Action: DescribeEngineDefaultParameters, SignatureMethod: HmacSHA256,
AWSAccessKeyId: ACCESSKEYID, Version: 2009-10-16, SignatureVersion: 2,
Engine: mysql5.1, Timestamp: 2009-12-17T17:53:04.267Z, Signature:
q963XH63Lcovl5Rr71APlzlye99rmWwT9DfuQaNznkD, ) 2009-12-17 09:53:04,464
[main] DEBUG com.amazonaws.request - Received successful response: 200, {AWS}
Request ID: 694d1242-cee0-c85e-f31f-5dab1ea18bc6 2009-12-17 09:53:04,469
[main] DEBUG com.amazonaws.request - Sending Request: POST
https://rds.amazonaws.com / Parameters: (ResetAllParameters: true, Action:
```

```
ResetDBParameterGroup, SignatureMethod: HmacSHA256, DBParameterGroupName:
java-integ-test-param-group-0000000000000, AWSAccessKeyId: ACCESSKEYID,
Version: 2009-10-16, SignatureVersion: 2, Timestamp:
2009-12-17T17:53:04.467Z, Signature:
9WcgfPwTobvLVcpyhbrdN7P7l3uH0oviYQ4yZ+TQjsQ=, )

2009-12-17 09:53:04,646 [main] DEBUG com.amazonaws.request - Received
successful response: 200, {AWS} Request ID:
694d1242-cee0-c85e-f31f-5dab1ea18bc6
```

詳細連線記錄

在某些情況下，查看 AWS SDK for Java 發送和接收的確切請求和響應可能很有用。您不應該在生產系統中啟用此日誌記錄，因為寫出大型請求（例如，正在上傳的文件 Amazon S3）或響應可能會顯著降低應用程序的速度。如果您確實需要訪問此信息，則可以通過 Apache HttpClient 4 記錄器臨時啟用它。啟用 `org.apache.http.wire` 記錄器的 DEBUG 層級，可以啟用所有請求和回應資料的記錄。

下列 log4j.properties 檔案會在 Apache HttpClient 4 中開啟完整的電線記錄，並且只能暫時開啟，因為它可能會對您的應用程式產生重大的效能影響。

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
# Log all HTTP content (headers, parameters, content, etc)  for
# all requests and responses. Use caution with this since it can
# be very expensive to log such verbose data!
log4j.logger.org.apache.http.wire=DEBUG
```

延遲指標記錄

如果您正在進行故障排除，並希望查看指標，例如哪個進程花費最多時間，或者服務器或客戶端是否具有更大的延遲，則延遲記錄器可能會有所幫助。將`com.amazonaws.latency`記錄器設置為 DEBUG 以啟用此記錄器。

> **Note**
>
> 只有在啟用 SDK 指標時，才能使用此記錄器。若要深入瞭解 SDK 度量套件，請參閱[啟用 AWS SDK for Java](#).

```
log4j.rootLogger=WARN, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c -  %m%n
log4j.logger.com.amazonaws.latency=DEBUG
```

以下為日誌輸出的範例。

```
com.amazonaws.latency - ServiceName=[{S3}], StatusCode=[200],
ServiceEndpoint=[https://list-objects-integ-test-test.s3.amazonaws.com],
RequestType=[ListObjectsV2Request], AWSRequestID=[REQUESTID],
 HttpClientPoolPendingCount=0,
RetryCapacityConsumed=0, HttpClientPoolAvailableCount=0, RequestCount=1,
HttpClientPoolLeasedCount=0, ResponseProcessingTime=[52.154],
 ClientExecuteTime=[487.041],
HttpClientSendRequestTime=[192.931], HttpRequestTime=[431.652],
 RequestSigningTime=[0.357],
CredentialsRequestTime=[0.011, 0.001], HttpClientReceiveResponseTime=[146.272]
```

客戶端組態

AWS SDK for Java 可讓您變更預設用戶端組態，這在您想要執行下列作業時很有幫助：

- 透過代理伺服器 Connect 網際網路
- 變更 HTTP 傳輸設定，例如連線逾時和要求重試
- 指定 TCP 通訊端緩衝區大小提示

代理組態

構建客戶端對象時，您可以傳入可選ClientConfiguration對象以自定義客戶端的配置。

如果您透過 Proxy 伺服器連線到網際網路，您需要透過物件設定 Proxy 伺服器設定 (Proxy 主機、連接埠和使用者名稱/密碼)。`ClientConfiguration`

傳輸組態

您可以使用ClientConfiguration物件來設定數個 HTTP 傳輸選項。偶爾會新增新選項；若要查看可擷取或設定的完整選項清單，請參閱 AWS SDK for Java API 參考。

> **Note**
>
> 每個可設定值都有一個由常數定義的預設值。如需的常數值清單`ClientConfiguration`，請參閱 AWS SDK for Java API 參考中的常數欄位值。

連線數上限

您可以使用設定允許的開啟 HTTP 連線數目上限ClientConfiguration。 setMaxConnections方法。

> **Important**
>
> 設定連線數上限到並行交易，避免連線失敗和效能不佳。有關默認最大連接值，請參閱 AWS SDK for Java API 參考中的常量字段值。

逾時和錯誤處理

您可以設定與 HTTP 連線的逾時和處理錯誤相關的選項。

- 連線逾時

  連線逾時是 HTTP 連線在放棄之前等待建立連線的時間量 (以毫秒為單位)。預設值為 10,000 毫秒。

  若要自行設定此值，請使用 ClientConfiguration. setConnectionTimeout方法。
- 即時連線時間 (TTL)

  根據預設，SDK 會盡可能長時間嘗試重複使用 HTTP 連線。在失敗的情況下，建立連線到已經退出服務的伺服器，具有有限的 TTL 可以協助應用程式復原。例如，設定 15 分鐘 TTL 可確保即使您已建立連線到發生問題的伺服器，也可以在 15 分鐘內重新建立與新伺服器的連線。

  若要設定 HTTP 連線 TTL，請使用 ClientConfiguration.set 連線 TTL 方法。
- 錯誤重試次數上限

  可重新擷取錯誤的預設重試次數上限為 3。您可以使用設定不同的值ClientConfiguration。 setMaxError重試方法。

本地地址

若要設定 HTTP 用戶端將繫結的本機位址，請使用ClientConfiguration。 setLocalAddress。

TCP 通訊端緩衝區大小提示

想要調整低階 TCP 參數的進階使用者可以透過ClientConfiguration物件另外設定 TCP 緩衝區大小提示。大多數用戶永遠不需要調整這些值，但它們是為高級用戶提供的。

應用程式的最佳 TCP 緩衝區大小高度取決於網路和作業系統的組態和功能。例如，大多數現代作業系統都針對 TCP 緩衝區大小提供自動調整邏輯。這對 TCP 連線的效能有很大的影響，因為這些連線保持開放時間足以讓自動調整以最佳化緩衝區大小。

較大的緩衝區大小（例如 2 MB）允許操作系統在內存中緩衝更多數據，而無需遠程服務器確認收到該信息，因此當網絡具有高延遲時特別有用。

這只是一個提示，操作系統可能不遵守它。使用此選項時，用戶應始終檢查操作系統的配置限制和默認值。大多數作業系統都設定了 TCP 緩衝區大小上限，並且除非您明確提高最大 TCP 緩衝區大小限制，否則不會讓您超出該限制。

許多資源可協助您設定 TCP 緩衝區大小和作業系統特定 TCP 設定，包括下列項目：

- 主機調整

存取控制政策

AWS 存取控制原則可讓您針對 AWS 資源指定精細的存取控制。存取控制原則是由陳述式集合所組成，這些陳述式採用下列格式：

帳號 A 有權在適用條件 D 的資源 C 上執行動作 B。

其中：

- A 是主體-正在 AWS 帳戶 提出要求以存取或修改其中一個 AWS 資源的主體。
- B 是動作-訪問或修改 AWS 資源的方式，例如向 Amazon SQS 隊列發送消息或將對象存儲在存儲 Amazon S3 桶中。
- C 是資源-主 AWS 體想要訪問的實體，例如 Amazon SQS 隊列或存儲在其中的對象 Amazon S3。
- D 是一組條件-可選限制，用於指定何時允許或拒絕主參與者存取您的資源的存取權限。許多表現條件都可用，有些特定於每個服務。例如，您可以使用日期條件，只允許在特定時間之後或之前存取資源。

Amazon S3 例子

下列範例示範一項政策，允許任何人存取讀取值區中的所有物件，但將上傳物件至該值區的存取權限限制為兩個特定 AWS 帳戶 s (除了值區擁有者的帳戶之外)。

```
Statement allowPublicReadStatement = new Statement(Effect.Allow)
    .withPrincipals(Principal.AllUsers)
    .withActions(S3Actions.GetObject)
    .withResources(new S3ObjectResource(myBucketName, "*"));
Statement allowRestrictedWriteStatement = new Statement(Effect.Allow)
    .withPrincipals(new Principal("123456789"), new Principal("876543210"))
    .withActions(S3Actions.PutObject)
    .withResources(new S3ObjectResource(myBucketName, "*"));

Policy policy = new Policy()
    .withStatements(allowPublicReadStatement, allowRestrictedWriteStatement);

AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();
s3.setBucketPolicy(myBucketName, policy.toJson());
```

Amazon SQS 例子

政策的一個常見用途是授權 Amazon SQS 佇列接收來自 Amazon SNS 主題的訊息。

```
Policy policy = new Policy().withStatements(
    new Statement(Effect.Allow)
        .withPrincipals(Principal.AllUsers)
        .withActions(SQSActions.SendMessage)
        .withConditions(ConditionFactory.newSourceArnCondition(myTopicArn)));

Map queueAttributes = new HashMap();
queueAttributes.put(QueueAttributeName.Policy.toString(), policy.toJson());

AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.setQueueAttributes(new SetQueueAttributesRequest(myQueueUrl, queueAttributes));
```

Amazon SNS 示例

某些服務提供額外的條件，可以在政策中使用。Amazon SNS 根據訂閱主題的通訊協定 (例如電子郵件、HTTP、HTTPS Amazon SQS) 和端點 (例如電子郵件地址、URL、 Amazon SQS ARN)，提供允許或拒絕訂閱 SNS 主題的條件。

```
Condition endpointCondition =
    SNSConditionFactory.newEndpointCondition("*@mycompany.com");

Policy policy = new Policy().withStatements(
    new Statement(Effect.Allow)
        .withPrincipals(Principal.AllUsers)
        .withActions(SNSActions.Subscribe)
        .withConditions(endpointCondition));

AmazonSNS sns = AmazonSNSClientBuilder.defaultClient();
sns.setTopicAttributes(
    new SetTopicAttributesRequest(myTopicArn, "Policy", policy.toJson()));
```

設定 DNS 名稱查詢的 JVM TTL

Java 虛擬機器 (JVM) 會快取 DNS 名稱查詢。當 JVM 將主機名稱解析為 IP 位址時，會將 IP 位址快取一段指定的時間段，稱為 time-to-live(TTL)。

由於 AWS 資源使用偶爾會變更的 DNS 名稱項目，因此建議您將 JVM 設定為不超過 60 秒的 TTL 值。這可確保當資源的 IP 位址變更時，您的應用程式將可透過重新查詢 DNS 來接收並使用資源的新 IP 位址。

在一些 Java 組態上，JVM 的預設 TTL 會如此設定，在重新啟動 JVM 之前，「絕不」重新整理 DNS 項目。因此，如果 AWS 資源的 IP 位址在應用程式仍在執行時發生變更，則除非您手動重新啟動 JVM 並重新整理快取的 IP 資訊，否則該資源將無法使用該資源。在此情況下，設定 JVM 的 TTL 至為關鍵，以便其定期重新整理快取的 IP 資訊。

> **Note**
>
> 預設 TTL 會隨 JVM 版本及是否安裝安全管理員而異。許多 JVM 提供的預設 TTL 少於 60 秒。如果您使用的是此類 JVM，而不是安全管理員，您可忽略本主題的其餘內容。

如何設定 JVM TTL

若要修改 JVM 的 TTL，請設定 networkaddress.cache.ttl 屬性值。根據您的需求，使用下列其中一種方法：

- 適用全體使用 JVM 的應用程式。networkaddress.cache.ttl在 Java 8 或更高版本的$JAVA_HOME/jre/lib/security/java.security文件中設置為 Java 11 或更高版本的$JAVA_HOME/conf/security/java.security文件：

```
networkaddress.cache.ttl=60
```

- 為您的應用程序，在應用程式的初始化程式碼中設定 networkaddress.cache.ttl：

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
```

啟用指標 AWS SDK for Java

AWS SDK for Java 可以生成指標 CloudWatch，用於可視化和監控與 Amazon 衡量：

- 訪問時應用程序的性能 AWS
- 搭配使用時 JVM 的效能 AWS
- 運行時環境詳細信息，例如堆內存，線程數和打開的文件描述符

如何啟用 Java SDK 指標生成

您需要添加以下 Maven 依賴項，以使 SDK 將指標發送到 CloudWatch。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.12.490*</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-cloudwatchmetrics</artifactId>
    <scope>provided</scope>
  </dependency>
```

```
  <!-- Other SDK dependencies. -->
</dependencies>
```

* 將版本號替換為 Maven 中央提供的最新版本的 SDK。

AWS SDK for Java 預設會停用量度。若要為您的本機開發環境啟用此功能，請在啟動 JVM 時加入指向 AWS 安全認證檔案的系統屬性。例如：

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/aws.properties
```

您需要指定認證檔案的路徑，以便 SDK 可以將收集的資料點上傳到以 CloudWatch 供日後分析。

> **Note**
>
> 如果您使用 Amazon EC2 執行個體中繼資料服務 AWS 從 Amazon EC2 執行個體存取，則不需要指定認證檔案。在這種情況下，您只需要指定：
>
> ```
> -Dcom.amazonaws.sdk.enableDefaultMetrics
> ```

擷取的所有量度 AWS SDK for Java 都位於命名空間 AWSSDK/Java 之下，並會上傳至 CloudWatch 預設區域 (us-ea st-1)。若要變更區域，請使用系統`cloudwatchRegion`屬性中的屬性來指定區域。例如，若要將 CloudWatch 區域設定為 us-east-1，請使用：

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/
aws.properties,cloudwatchRegion={region_api_default}
```

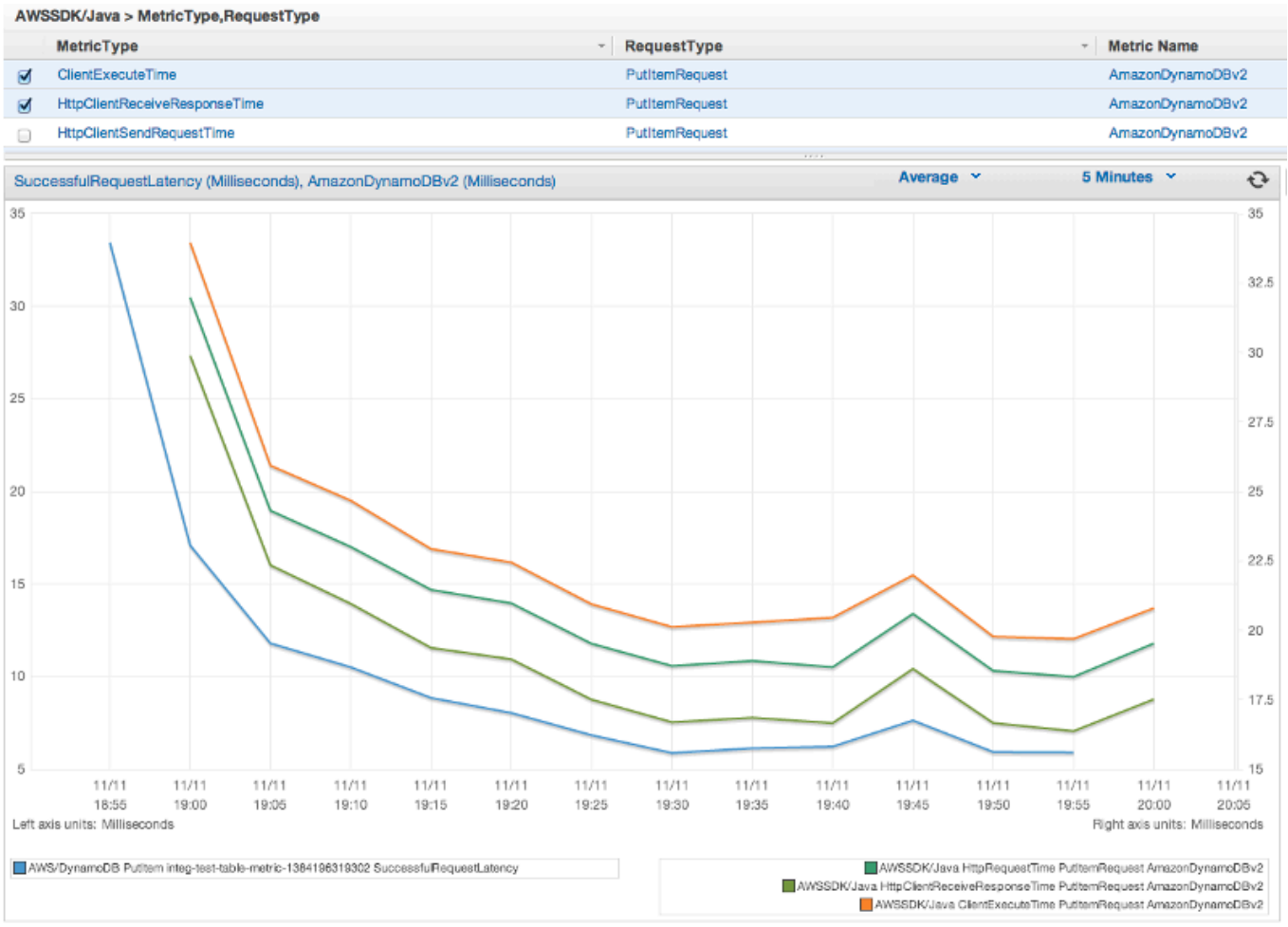
啟用此功能後，每當有服務要求來 AWS 自時，就會產生測量結果資料點 AWS SDK for Java、排入佇列以取得統計摘要，並以非同步方式上傳至大 CloudWatch 約每分鐘一次。上傳量度後，您可以使用以視覺化方式呈現，AWS Management Console並針對潛在問題 (例如記憶體洩漏、檔案描述元洩漏等) 設定警示。

可用的量度類型

預設量度集分為三個主要類別：

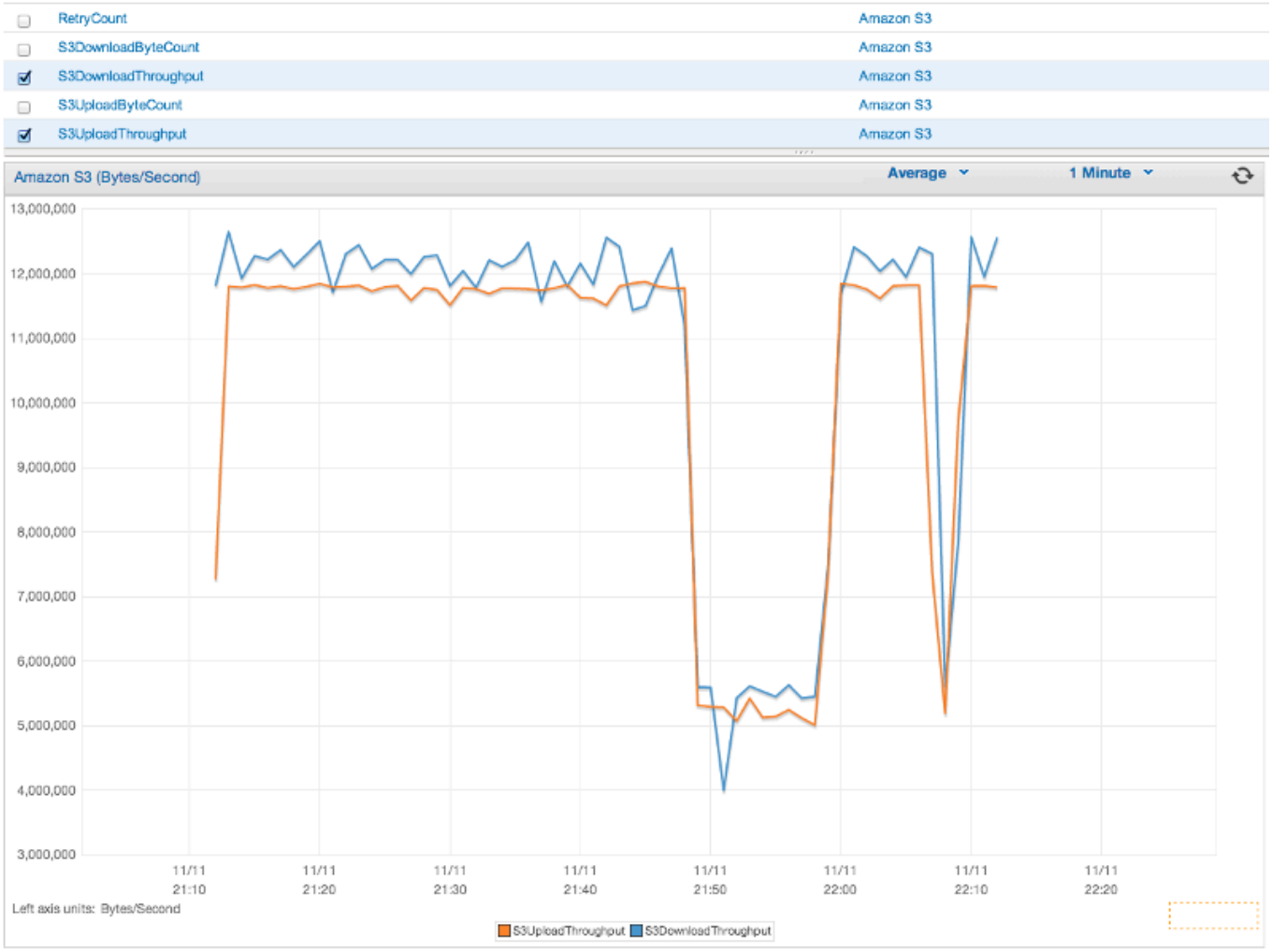
AWS 要求量度

- 涵蓋 HTTP 要求/回應的延遲、要求數目、例外狀況和重試等區域。



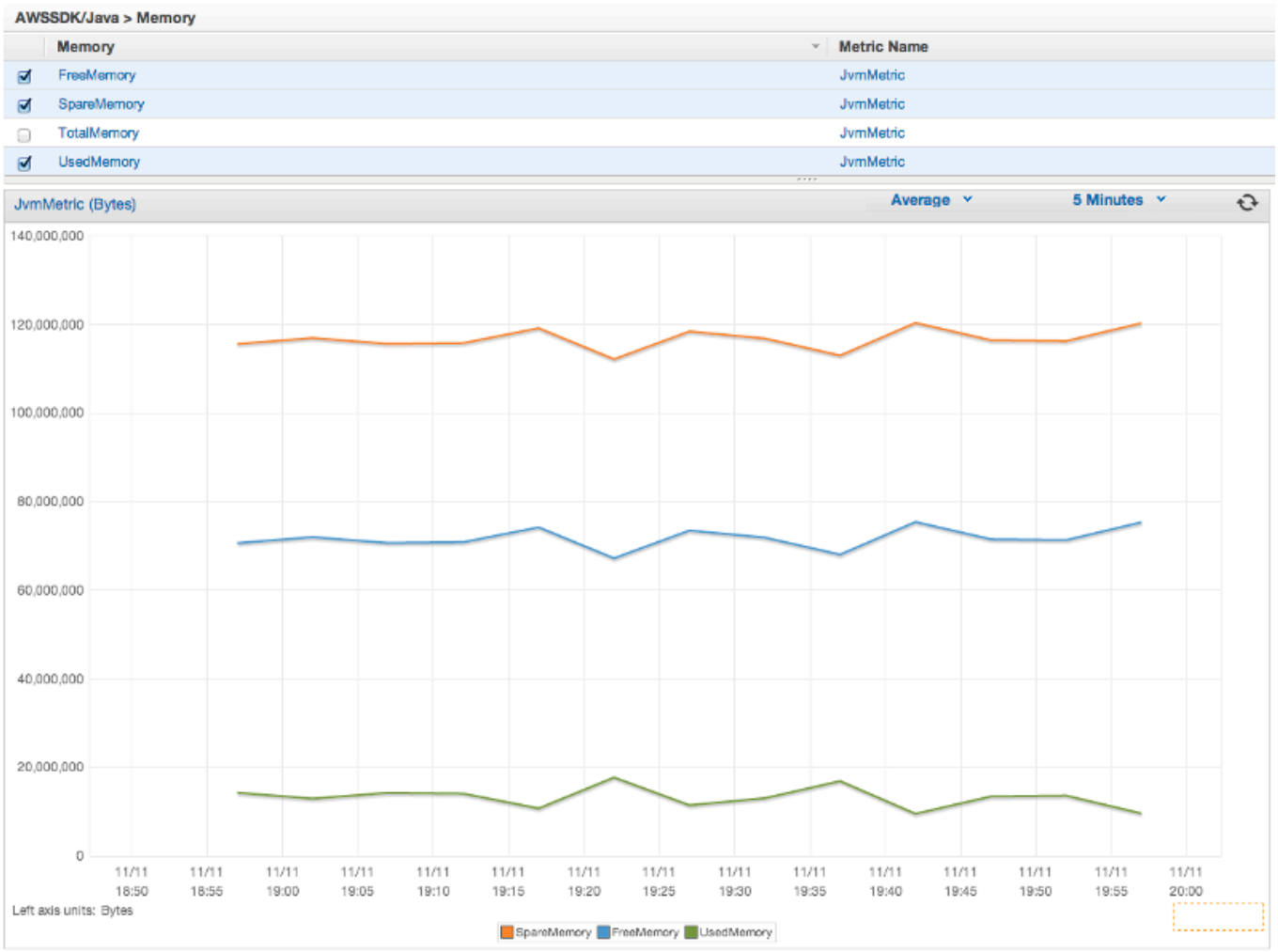
AWS 服務 度量

- 包含 AWS 服務特定資料，例如 S3 上傳和下載的輸送量和位元組計數。



機器度量

- 涵蓋運行時環境，包括堆內存，線程數和打開文件描述符。



如果您想要排除機器度量，請新增`excludeMachineMetrics`至系統內容：

```
-Dcom.amazonaws.sdk.enableDefaultMetrics=credentialFile=/path/
aws.properties,excludeMachineMetrics
```

詳細資訊

- 如需預先定義的核心指標類型的完整清單，請參閱 amazonaws/度量套件摘要。
- 瞭解如何 CloudWatch 使用範CloudWatch 例 AWS SDK for Java 中的使用 AWS SDK for Java。
- 若要進一步了解效能調整，請參閱調整 AWS SDK for Java 以改善彈性部落格文章。

AWS SDK for Java 程式碼範例

本節提供使用AWS SDK for Java v1 對AWS服務進行程式設計的教學課程和範例。

在上的AWS文檔代碼示例存儲庫中找到這些示例和其他示例的源代碼GitHub。

若要向 AWS 文件團隊提議可考慮產生的新程式碼範例，請建立新請求。該團隊想要產生比僅涵蓋個別 API 呼叫之簡易程式碼更為廣泛的程式碼範例，以涵蓋更為廣泛的案例和使用案例。如需指示，請參閱上的程式碼範例儲存庫中的「貢獻」指南GitHub。

AWS SDK for Java2.x

在 2018 年，AWS發布了 AWS SDK for Java 2.x. 本指南包含使用最新 Java SDK 的說明以及範例程式碼。

> **Note**
>
> 如需AWS SDK for Java開發人員可用的更多範例和其他資源，請參閱其他文件和資源！

使用AWS SDK for Java

本節提供使用AWS SDK for Java編寫 CloudWatch 程式的範例。

亞馬遜 CloudWatch 可監控Amazon Web Services(AWS) 資源和您在上執行的應用程式AWS即時。您可以使用 CloudWatch 收集和追蹤指標，這些是您可以為您的資源和應用程式測量的變數。CloudWatch 警示會根據您定義的規則來發送通知，或自動對您監控的資源進行變更。

如需有關 CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 從 CloudWatch 取得指標
- 發佈自訂指標資料
- 使用 CloudWatch 警報
- 使用 CloudWatch 中的警示動作
- 傳送事件到 CloudWatch

從 CloudWatch 取得指標

列出指標

要清單 CloudWatch 衡量指標，請創建ListMetricsRequest，並致電卓越亞馬遜關注客户端的`listMetrics`方法。您可以使用 `ListMetricsRequest` 來根據命名空間、指標名稱或維度，篩選傳回的指標。

> **Note**
>
> 發佈的指標和維度的列表AWS服務可以在 {HTTPS-Docs-AWS-AWS-AWS-AMAZON-亞馬遜-亞馬遜雲監視-最新監控-CW 支持-AWS-html} [亞馬遜 CloudWatch 指標和維度參考]Amazon CloudWatch使用者指南。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.ListMetricsRequest;
import com.amazonaws.services.cloudwatch.model.ListMetricsResult;
import com.amazonaws.services.cloudwatch.model.Metric;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

ListMetricsRequest request = new ListMetricsRequest()
        .withMetricName(name)
        .withNamespace(namespace);
```

```
boolean done = false;

while(!done) {
    ListMetricsResult response = cw.listMetrics(request);

    for(Metric metric : response.getMetrics()) {
        System.out.printf(
            "Retrieved metric %s", metric.getMetricName());
    }

    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
}
```

指標會在ListMetricsResult通過調用其getMetrics方法。結果可能會分頁。要檢索下一批次的結果，請呼叫setNextToken在原始請求對象上，返回值為ListMetricsResult物件的getNextToken方法，並傳遞修改後的請求物件傳回listMetrics。

詳細資訊

- ListMetrics中的Amazon CloudWatchAPI 參考。

發佈自訂指標資料

的一些AWS服務發佈他們自己的指標在以」開頭的命名空間中AWS 「您也可以使用自己的命名空間來發佈自訂指標資料（只要開頭不是」AWS 「）。

發佈自訂指標資料

若要發佈您自己的指標資料，請呼叫putMetricData方法並搭配PutMetricDataRequest詢。PutMetricDataRequest 必須在 MetricDatum 物件中包含用於資料的自訂命名空間，以及資料點本身的相關資訊。

> **Note**
>
> 您不能指定開頭為」AWS 「。以」開頭的命名空間AWS 保留以供日後使用Amazon Web Services產品。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.Dimension;
import com.amazonaws.services.cloudwatch.model.MetricDatum;
import com.amazonaws.services.cloudwatch.model.PutMetricDataRequest;
import com.amazonaws.services.cloudwatch.model.PutMetricDataResult;
import com.amazonaws.services.cloudwatch.model.StandardUnit;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

Dimension dimension = new Dimension()
    .withName("UNIQUE_PAGES")
    .withValue("URLS");

MetricDatum datum = new MetricDatum()
    .withMetricName("PAGES_VISITED")
    .withUnit(StandardUnit.None)
    .withValue(data_point)
    .withDimensions(dimension);

PutMetricDataRequest request = new PutMetricDataRequest()
    .withNamespace("SITE/TRAFFIC")
    .withMetricData(datum);

PutMetricDataResult response = cw.putMetricData(request);
```

詳細資訊

- 使用Amazon CloudWatch指標中的Amazon CloudWatch使用者指南。
- AWS命名空間中的Amazon CloudWatch使用者指南。
- PutMetricData中的Amazon CloudWatchAPI 參考。

使用 CloudWatch 警報

建立警示

若要根據 CloudWatch 指標，請調用卓越亞馬遜關注客户端的`putMetricAlarm`方法並搭配PutMetricAlarmRequest充滿了報警條件。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.ComparisonOperator;
import com.amazonaws.services.cloudwatch.model.Dimension;
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmRequest;
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmResult;
import com.amazonaws.services.cloudwatch.model.StandardUnit;
import com.amazonaws.services.cloudwatch.model.Statistic;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

Dimension dimension = new Dimension()
    .withName("InstanceId")
    .withValue(instanceId);

PutMetricAlarmRequest request = new PutMetricAlarmRequest()
    .withAlarmName(alarmName)
    .withComparisonOperator(
        ComparisonOperator.GreaterThanThreshold)
    .withEvaluationPeriods(1)
    .withMetricName("CPUUtilization")
    .withNamespace("{AWS}/EC2")
    .withPeriod(60)
    .withStatistic(Statistic.Average)
    .withThreshold(70.0)
    .withActionsEnabled(false)
    .withAlarmDescription(
        "Alarm when server CPU utilization exceeds 70%")
    .withUnit(StandardUnit.Seconds)
    .withDimensions(dimension);
```

```
PutMetricAlarmResult response = cw.putMetricAlarm(request);
```

列出警示

若要列出 CloudWatch 警示，請呼叫卓越亞馬遜 CloudWatch 客户端的`describeAlarms`方法並搭配DescribeAlarmsRequest，您可以使用它來設置結果的選項。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DescribeAlarmsRequest;
import com.amazonaws.services.cloudwatch.model.DescribeAlarmsResult;
import com.amazonaws.services.cloudwatch.model.MetricAlarm;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

boolean done = false;
DescribeAlarmsRequest request = new DescribeAlarmsRequest();

while(!done) {

    DescribeAlarmsResult response = cw.describeAlarms(request);

    for(MetricAlarm alarm : response.getMetricAlarms()) {
        System.out.printf("Retrieved alarm %s", alarm.getAlarmName());
    }

    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
}
```

警示清單可以通過呼叫`getMetricAlarms`在DescribeAlarmsResult，會傳回`describeAlarms`。

結果可能會分頁。若要檢索下一批次的結果，請呼叫setNextToken在原始請求對象上，返回值為DescribeAlarmsResult物件的getNextToken方法，並將修改的請求物件傳回另一個describeAlarms。

> **Note**
>
> 您也可以使用卓越亞馬遜 CloudWatch 客戶端的describeAlarmsForMetric方法。其用法類似於 describeAlarms。

刪除警示

要刪除 CloudWatch 警報，請調用卓越亞馬遜關注客戶端的deleteAlarms方法並搭配DeleteAlarmsRequest，其包含一個或多個您所要刪除警示的名稱。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DeleteAlarmsRequest;
import com.amazonaws.services.cloudwatch.model.DeleteAlarmsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

DeleteAlarmsRequest request = new DeleteAlarmsRequest()
    .withAlarmNames(alarm_name);

DeleteAlarmsResult response = cw.deleteAlarms(request);
```

詳細資訊

- 正在建立Amazon CloudWatch警報中的Amazon CloudWatch使用者指南
- PutMetricAlarm中的Amazon CloudWatchAPI 參考
- DescribeAlarms中的Amazon CloudWatchAPI 參考
- DeleteAlarms中的Amazon CloudWatchAPI 參考

使用 CloudWatch 中的警示動作

使用 CloudWatch 警示動作，您可以建立警示來執行動作，例如自動停止、終止、重新啟動或復原 Amazon EC2實例。

> **Note**
>
> 在建立警示時，可以使用 `setAlarmActions`PutMetricAlarmRequest 的 方法，將警示動作新增到警示。

啟用警示動作

要啟用警報操作 CloudWatch 警報，請致電卓越亞馬遜關注客戶端的`enableAlarmActions`具有EnableAlarmActionsRequest包含一個或多個您想要啟用其動作的警示名稱。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.EnableAlarmActionsRequest;
import com.amazonaws.services.cloudwatch.model.EnableAlarmActionsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

EnableAlarmActionsRequest request = new EnableAlarmActionsRequest()
    .withAlarmNames(alarm);

EnableAlarmActionsResult response = cw.enableAlarmActions(request);
```

停用警示動作

若要禁用警報操作 CloudWatch 警報，請致電卓越亞馬遜關注客戶端的`disableAlarmActions`具有DisableAlarmActionsRequest包含一個或多個您想要停用其動作的警示名稱。

匯入

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.DisableAlarmActionsRequest;
import com.amazonaws.services.cloudwatch.model.DisableAlarmActionsResult;
```

Code

```
final AmazonCloudWatch cw =
    AmazonCloudWatchClientBuilder.defaultClient();

DisableAlarmActionsRequest request = new DisableAlarmActionsRequest()
    .withAlarmNames(alarmName);

DisableAlarmActionsResult response = cw.disableAlarmActions(request);
```

詳細資訊

- 建立警示以停止、終止、重新啟動或復原執行個體中的Amazon CloudWatch使用者指南
- PutMetricAlarm中的Amazon CloudWatchAPI 參考
- EnableAlarmActions中的Amazon CloudWatchAPI 參考
- DisableAlarmActions中的Amazon CloudWatchAPI 參考

傳送事件到 CloudWatch

CloudWatch事件可傳送近乎即時的系統事件串流，描述AWS資源Amazon EC2實例,Lambda函數,Kinesis串流,Amazon ECS任務,Step Functions狀態機器,Amazon SNS主題,Amazon SQS隊列或內置目標。您可以使用簡單的規則，來比對事件，並將這些事件轉傳到一或多個目標函數或串流。

新增事件

新增自訂CloudWatch事件，請致電卓越亞馬遜客户端的`putEvents`方法搭配PutEventsRequest對象包含一或多個PutEventsRequestEntry對象，提供有關每個事件的詳細信息。您可以指定項目的多個參數，例如事件的來源和類型、與事件相關聯的資源等等。

> **Note**
>
> 對 `putEvents` 的每個呼叫最多可以指定 10 個事件。

匯入

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutEventsRequest;
import com.amazonaws.services.cloudwatchevents.model.PutEventsRequestEntry;
import com.amazonaws.services.cloudwatchevents.model.PutEventsResult;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

final String EVENT_DETAILS =
    "{ \"key1\": \"value1\", \"key2\": \"value2\" }";

PutEventsRequestEntry request_entry = new PutEventsRequestEntry()
    .withDetail(EVENT_DETAILS)
    .withDetailType("sampleSubmitted")
    .withResources(resource_arn)
    .withSource("aws-sdk-java-cloudwatch-example");

PutEventsRequest request = new PutEventsRequest()
    .withEntries(request_entry);

PutEventsResult response = cwe.putEvents(request);
```

新增規則

若要建立或更新規則，請呼叫卓越管理程序客戶端的putRule方法搭配完全 PutRuleRequest與規則的名稱和可選參數（例如事件模式、IAM角色與規則關聯，以及排程表達式，描述了規則的運行頻率。

匯入

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutRuleRequest;
import com.amazonaws.services.cloudwatchevents.model.PutRuleResult;
import com.amazonaws.services.cloudwatchevents.model.RuleState;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

PutRuleRequest request = new PutRuleRequest()
    .withName(rule_name)
    .withRoleArn(role_arn)
    .withScheduleExpression("rate(5 minutes)")
    .withState(RuleState.ENABLED);

PutRuleResult response = cwe.putRule(request);
```

新增目標

目標是觸發規則時叫用的資源。目標的例子包括 Amazon EC2 執行個體、Lambda 函數、Kinesis 串流、Amazon ECS 任務、Step Functions 狀態機器和內建目標。

若要新增目標到規則，請呼叫卓越管理程序客户端的`putTargets`方法搭配[PutTargetsRequest]中包含要更新的規則和要新增至規則的目標清單。

匯入

```
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEvents;
import com.amazonaws.services.cloudwatchevents.AmazonCloudWatchEventsClientBuilder;
import com.amazonaws.services.cloudwatchevents.model.PutTargetsRequest;
import com.amazonaws.services.cloudwatchevents.model.PutTargetsResult;
import com.amazonaws.services.cloudwatchevents.model.Target;
```

Code

```
final AmazonCloudWatchEvents cwe =
    AmazonCloudWatchEventsClientBuilder.defaultClient();

Target target = new Target()
    .withArn(function_arn)
    .withId(target_id);

PutTargetsRequest request = new PutTargetsRequest()
    .withTargets(target)
    .withRule(rule_name);

PutTargetsResult response = cwe.putTargets(request);
```

詳細資訊

- 使用 PutEvents 新增事件中的Amazon CloudWatch Events使用者指南
- 排程規則運算式中的Amazon CloudWatch Events使用者指南
- 事件類型CloudWatch活動中的Amazon CloudWatch Events使用者指南
- 事件和事件模式中的Amazon CloudWatch Events使用者指南
- PutEvents中的Amazon CloudWatch EventsAPI 參考
- PutTargets中的Amazon CloudWatch EventsAPI 參考
- PutRule中的Amazon CloudWatch EventsAPI 參考

使用 AWS SDK for Java 的 DynamoDB 範例

本節提供使用AWS SDK for Java編寫 DynamoDB 程式的範例。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 處理資料表DynamoDB
- 在 中處理項目DynamoDB

處理資料表DynamoDB

資料表是 DynamoDB 資料庫中所有項目的容器。您必須先建立資料表，然後才可以從 DynamoDB 新增或移除資料。

對於每個資料表，您必須定義：

- 對於您的帳戶和區域都具有獨一性的資料表名稱。
- 每個值的主索引鍵都必須獨一無二，資料表中任兩個項目不能有相同的主索引鍵值。

  主索引鍵可以是簡單的，包含單一分割區 (HASH) 索引鍵；也可以是複合的，包含分割區和排序 (RANGE) 索引鍵。

每個鍵值都有一個關聯的數據類型，由ScalarAttributeType類枚舉。索引鍵值可以是二進位 (B)、數值 (N)、或字串 (S)。如需詳細資訊，請參閱Amazon DynamoDB開發人員指南中的命名規則和資料類型。

- 佈建的輸送量值，用於定義表格的保留讀取/寫入容量單位數目。

> **Note**
>
> Amazon DynamoDB定價是根據您在表格上設定的佈建輸送量值，因此請只保留您認為表格需要的容量。

您可以隨時修改表格的佈建輸送量，因此您可以在需求變更時調整容量。

建立資料表

使用用DynamoDB戶端的`createTable`方法建立新DynamoDB資料表。您需要建構資料表屬性和資料表結構描述，這兩項都會用來識別資料表的主索引鍵。您也必須提供初始佈建的輸送量值和資料表名稱。僅在建立資料表時定義關鍵資DynamoDB料表屬性。

> **Note**
>
> 如果具有您選擇的名稱的資料表已存在，AmazonServiceException就會擲回一個。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
```

使用簡單主索引鍵建立資料表

此程式碼會使用簡單主索引鍵 ("Name") 來建立資料表。

Code

```
CreateTableRequest request = new CreateTableRequest()
    .withAttributeDefinitions(new AttributeDefinition(
            "Name", ScalarAttributeType.S))
    .withKeySchema(new KeySchemaElement("Name", KeyType.HASH))
    .withProvisionedThroughput(new ProvisionedThroughput(
            new Long(10), new Long(10)))
    .withTableName(table_name);

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    CreateTableResult result = ddb.createTable(request);
    System.out.println(result.getTableDescription().getTableName());
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

使用複合主索引鍵建立資料表

將另一個AttributeDefinition和新增KeySchemaElement至CreateTableRequest。

Code

```
CreateTableRequest request = new CreateTableRequest()
    .withAttributeDefinitions(
          new AttributeDefinition("Language", ScalarAttributeType.S),
          new AttributeDefinition("Greeting", ScalarAttributeType.S))
    .withKeySchema(
          new KeySchemaElement("Language", KeyType.HASH),
          new KeySchemaElement("Greeting", KeyType.RANGE))
    .withProvisionedThroughput(
          new ProvisionedThroughput(new Long(10), new Long(10)))
    .withTableName(table_name);
```

請參閱（詳見）的完整實例GitHub。

列出資料表

您可以通過調DynamoDB用客戶端的`listTables`方法列出特定區域中的表。

> **Note**
>
> 如果您的帳戶和區域不存在具名資料表，ResourceNotFoundException就會擲回 a。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.ListTablesRequest;
import com.amazonaws.services.dynamodbv2.model.ListTablesResult;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

ListTablesRequest request;

boolean more_tables = true;
String last_name = null;

while(more_tables) {
    try {
        if (last_name == null) {
         request = new ListTablesRequest().withLimit(10);
        }
        else {
         request = new ListTablesRequest()
            .withLimit(10)
            .withExclusiveStartTableName(last_name);
        }

        ListTablesResult table_list = ddb.listTables(request);
        List<String> table_names = table_list.getTableNames();

        if (table_names.size() > 0) {
            for (String cur_name : table_names) {
```

```
                System.out.format("* %s\n", cur_name);
            }
        } else {
            System.out.println("No tables found!");
            System.exit(0);
        }

        last_name = table_list.getLastEvaluatedTableName();
        if (last_name == null) {
            more_tables = false;
        }
```

根據預設，每次呼叫最多會傳回 100 個表格 — 用`getLastEvaluatedTableName`於傳回的ListTablesResult物件來取得最後一個評估的資料表。您可以使用這個值，在前次列表最後傳回值之後開始列表。

請參閱（詳見）的完整實例GitHub。

說明資料表 (取得相關資訊)

調DynamoDB用客戶端的`describeTable`方法。

> **Note**
>
> 如果您的帳戶和區域不存在具名資料表，ResourceNotFoundException就會擲回 a。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputDescription;
import com.amazonaws.services.dynamodbv2.model.TableDescription;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
```

```
try {
    TableDescription table_info =
       ddb.describeTable(table_name).getTable();

    if (table_info != null) {
        System.out.format("Table name  : %s\n",
              table_info.getTableName());
        System.out.format("Table ARN   : %s\n",
              table_info.getTableArn());
        System.out.format("Status      : %s\n",
              table_info.getTableStatus());
        System.out.format("Item count  : %d\n",
              table_info.getItemCount().longValue());
        System.out.format("Size (bytes): %d\n",
              table_info.getTableSizeBytes().longValue());

        ProvisionedThroughputDescription throughput_info =
           table_info.getProvisionedThroughput();
        System.out.println("Throughput");
        System.out.format("  Read Capacity : %d\n",
              throughput_info.getReadCapacityUnits().longValue());
        System.out.format("  Write Capacity: %d\n",
              throughput_info.getWriteCapacityUnits().longValue());

        List<AttributeDefinition> attributes =
           table_info.getAttributeDefinitions();
        System.out.println("Attributes");
        for (AttributeDefinition a : attributes) {
            System.out.format("  %s (%s)\n",
                  a.getAttributeName(), a.getAttributeType());
        }
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

修改 (更新) 資料表

您可以透過呼叫DynamoDB用戶端的updateTable方法，隨時修改表格的佈建輸送量值。

> **Note**
>
> 如果您的帳戶和區域不存在具名資料表，ResourceNotFoundException就會擲回 a。

匯入

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.AmazonServiceException;
```

Code

```
ProvisionedThroughput table_throughput = new ProvisionedThroughput(
      read_capacity, write_capacity);

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.updateTable(table_name, table_throughput);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

刪除資料表

調用DynamoDB客戶端的`deleteTable`方法並將其傳遞給表的名稱。

> **Note**
>
> 如果您的帳戶和區域不存在具名資料表，ResourceNotFoundException就會擲回 a。

匯入

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
```

Code

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.deleteTable(table_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

詳細資訊

- Amazon DynamoDB開發人員指南中使用表格的準則
- 使用Amazon DynamoDB開發人員指南DynamoDB中的表格

在 中處理項目DynamoDB

在 DynamoDB 中，項目是屬性的集合，每個屬性都有名稱和值。屬性值可以是純量、集合或文件類型。如需詳細資訊，請參閱「」命名規則與資料類型中的Amazon DynamoDB開發人員指南。

從資料表擷取 (取得) 項目

致電卓越亞馬遜動態數據庫`getItem`方法並將其傳遞一個GetItemRequest物件，搭配所要項目的資料表名稱和主索引鍵值。其會傳回GetItemResult物件。

您可以使用所傳回 `GetItemResult` 物件的 `getItem()` 方法來擷取與項目關聯之索引鍵 (字串) 和值的對應 (AttributeValue) 組。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
```

```
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
```

Code

```
HashMap<String,AttributeValue> key_to_get =
    new HashMap<String,AttributeValue>();

key_to_get.put("DATABASE_NAME", new AttributeValue(name));

GetItemRequest request = null;
if (projection_expression != null) {
    request = new GetItemRequest()
        .withKey(key_to_get)
        .withTableName(table_name)
        .withProjectionExpression(projection_expression);
} else {
    request = new GetItemRequest()
        .withKey(key_to_get)
        .withTableName(table_name);
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    Map<String,AttributeValue> returned_item =
       ddb.getItem(request).getItem();
    if (returned_item != null) {
        Set<String> keys = returned_item.keySet();
        for (String key : keys) {
            System.out.format("%s: %s\n",
                    key, returned_item.get(key).toString());
        }
    } else {
        System.out.format("No item found with the key %s!\n", name);
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
```

請參閱 GitHub 上的完整範例。

新增項目到資料表

建立代表項目屬性的索引鍵值組對應。這些項目必須包含資料表主索引鍵欄位的值。如果主索引鍵識別的項目已存在，其欄位會透過請求更新。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，會擲出 ResourceNotFoundException。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

Code

```
HashMap<String,AttributeValue> item_values =
    new HashMap<String,AttributeValue>();

item_values.put("Name", new AttributeValue(name));

for (String[] field : extra_fields) {
    item_values.put(field[0], new AttributeValue(field[1]));
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
    ddb.putItem(table_name, item_values);
} catch (ResourceNotFoundException e) {
    System.err.format("Error: The table \"%s\" can't be found.\n", table_name);
    System.err.println("Be sure that it exists and that you've typed its name
 correctly!");
    System.exit(1);
} catch (AmazonServiceException e) {
    System.err.println(e.getMessage());
    System.exit(1);
```

請參閱 GitHub 上的完整範例。

更新資料表中的現有項目

您可以使用卓越亞馬遜 DynamoDB 的`updateItem`方法，提供資料表名稱、主索引鍵值和要更新的欄位對應。

> **Note**
>
> 如果您的帳戶和區域不存在指定的資料表，或者如果您傳遞的主索引鍵所識別的項目不存在，會擲出 ResourceNotFoundException。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import java.util.ArrayList;
```

Code

```
HashMap<String,AttributeValue> item_key =
   new HashMap<String,AttributeValue>();

item_key.put("Name", new AttributeValue(name));

HashMap<String,AttributeValueUpdate> updated_values =
    new HashMap<String,AttributeValueUpdate>();

for (String[] field : extra_fields) {
    updated_values.put(field[0], new AttributeValueUpdate(
                new AttributeValue(field[1]), AttributeAction.PUT));
}

final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();

try {
```

```
    ddb.updateItem(table_name, item_key, updated_values);
} catch (ResourceNotFoundException e) {
    System.err.println(e.getMessage());
    System.exit(1);
} catch (AmazonServiceException e) {
    System.err.println(e.getMessage());
    System.exit(1);
```

請參閱 GitHub 上的完整範例。

使用 DynamoDBMapper Class

所以此AWS SDK for Java提供DynamoDBMapper類別，可讓您將用户端類別映射至Amazon DynamoDB資料表。若要使用 AWS for WordPressDynamoDBMapper類別，請定義DynamoDB資料表及其對應物件執行個體（如下列程式碼範例所示）。所以此DynamoDBMapper類別可讓您存取資料表、執行各種建立、讀取、更新和刪除 (CRUD) 操作，以及執行查詢。

> **Note**
>
> 所以此DynamoDBMapper類別不允許您建立、更新或刪除資料表。

匯入

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException;
```

Code

下列 Java 程式碼範例示範如何將內容添加至音樂表中使用DynamoDBMapper類別。將內容添加到表中後，請注意，項目是通過使用分區和Sort鍵。然後獎項項目已更新。有關創建音樂表中，請參閱建立資料表中的Amazon DynamoDB開發人員指南。

```
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        MusicItems items = new MusicItems();
```

```
        try{
            // Add new content to the Music table
            items.setArtist(artist);
            items.setSongTitle(songTitle);
            items.setAlbumTitle(albumTitle);
            items.setAwards(Integer.parseInt(awards)); //convert to an int

            // Save the item
            DynamoDBMapper mapper = new DynamoDBMapper(client);
            mapper.save(items);

            // Load an item based on the Partition Key and Sort Key
            // Both values need to be passed to the mapper.load method
            String artistName = artist;
            String songQueryTitle = songTitle;

            // Retrieve the item
            MusicItems itemRetrieved = mapper.load(MusicItems.class, artistName,
songQueryTitle);
            System.out.println("Item retrieved:");
            System.out.println(itemRetrieved);

            // Modify the Award value
            itemRetrieved.setAwards(2);
            mapper.save(itemRetrieved);
            System.out.println("Item updated:");
            System.out.println(itemRetrieved);

            System.out.print("Done");
        } catch (AmazonDynamoDBException e) {
            e.getStackTrace();
        }
    }

    @DynamoDBTable(tableName="Music")
    public static class MusicItems {

        //Set up Data Members that correspond to columns in the Music table
        private String artist;
        private String songTitle;
        private String albumTitle;
        private int awards;

        @DynamoDBHashKey(attributeName="Artist")
```

```
        public String getArtist() {
            return this.artist;
        }

        public void setArtist(String artist) {
            this.artist = artist;
        }

        @DynamoDBRangeKey(attributeName="SongTitle")
        public String getSongTitle() {
            return this.songTitle;
        }

        public void setSongTitle(String title) {
            this.songTitle = title;
        }

        @DynamoDBAttribute(attributeName="AlbumTitle")
        public String getAlbumTitle() {
            return this.albumTitle;
        }

        public void setAlbumTitle(String title) {
            this.albumTitle = title;
        }

        @DynamoDBAttribute(attributeName="Awards")
        public int getAwards() {
            return this.awards;
        }

        public void setAwards(int awards) {
            this.awards = awards;
        }
    }
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 處理項目的指導方針中的Amazon DynamoDB開發人員指南
- 在 中處理項目DynamoDB中的Amazon DynamoDB開發人員指南

使用 AWS SDK for Java 的 Amazon EC2 範例

本節提供編寫程式的範例Amazon EC2與AWS SDK for Java。

主題

- 教學課程：建立 EC2 執行個體
- 使用 IAM 角色授予AWS上的資源Amazon EC2
- 教學課程：Amazon EC2Spot Instances
- 教學課程：進階Amazon EC2Spot 請求管理
- 管理 Amazon EC2 執行個體
- 在 Amazon EC2 中使用彈性 IP 地址
- 使用區域與可用區域
- 使用 Amazon EC2 金鑰對
- 在 Amazon EC2 中使用安全群組

教學課程：建立 EC2 執行個體

本指南示範如何使用AWS SDK for Java啟動 EC2 執行個體。

主題

- 先決條件
- 建立Amazon EC2安全群組
- 建立金鑰對
- 執行Amazon EC2執行個體

先決條件

開始之前，請務必先創建「」AWS 帳戶，並且您已設定AWS登入資料。如需詳細資訊，請參閱 入門。

建立Amazon EC2安全群組

EC2-Classic 正在淘汰

> **Warning**
>
> 我們將於 2022 年 8 月 15 日淘汰 EC2-Classic。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱「」從 EC2-Classic 遷移至 VPC中的Amazon EC2 Linux 執行個體使用者指南或Amazon EC2 Windows 執行個體使用者指南。另請參閱部落格文章EC2-Classic 網路正在淘汰-本文介紹如何準備。

建立安全群組，做為一種虛擬防火牆，控制一個或一個以上的 EC2 執行個體流量。在預設情況下，Amazon EC2 將您的執行個體與不允許輸入流量的安全群組相關聯。您可以建立允許您的 EC2 執行個體接受特定連接的安全群組。例如，如果您需要連接到 Linux 執行個體，則必須設定安全群組以允許 SSH 流量。您可以使用建立安全群組Amazon EC2主控台或AWS SDK for Java。

您可以建立安全群組，提供於 EC2-Classic 或 EC2-VPC 使用。如需 EC2-Classic 和 EC2-VPC 的詳細資訊，請參閱。支援的平台中的Amazon EC2Linux 執行個體使用者指南。

如需使用建立安全群組的詳細資訊Amazon EC2主控台，請參閱。Amazon EC2安全群組中的Amazon EC2Linux 執行個體使用者指南。

1. 建立和初始化 aCreateSecurityGroupRequest實例。使用withGroupName設定安全群組名稱的方法，以及withDescription設置安全組描述的方法，如下所示：

   ```
   CreateSecurityGroupRequest csgr = new CreateSecurityGroupRequest();
   csgr.withGroupName("JavaSecurityGroup").withDescription("My security group");
   ```

   安全群組名稱在AWS您初始化的區域Amazon EC2用戶端。您必須使用 US-ASCII 字元作為安全性群組名稱和說明。

2. 將請求物件做為參數傳遞給createSecurityGroup方法。該方式傳回 aCreateSecurityGroupResult物件，如下所示：

   ```
   CreateSecurityGroupResult createSecurityGroupResult =
       amazonEC2Client.createSecurityGroup(csgr);
   ```

   如果您嘗試以現有的安全群組名稱建立另一個安全群組，`createSecurityGroup`擲出例外狀況。

根據預設，新的安全性群組不允許任何輸入流量傳送到您的Amazon EC2實例。若要允許輸入流量，您必須明確授權安全性群組輸入。您可以針對個別 IP 位址、某範圍的 IP 位址、特定通訊協定以及 TCP/UDP 連接埠授權輸入。

1. 建立和初始化IpPermission實例。使用有 4 範圍設定要授權輸入的 IP 位址範圍的方法，並使用withIpProtocol設置 IP 協議的方法。使用withFromPort和withToPort指定要授權輸入之連接埠範圍的方法，如下所示：

```
IpPermission ipPermission =
    new IpPermission();

IpRange ipRange1 = new IpRange().withCidrIp("111.111.111.111/32");
IpRange ipRange2 = new IpRange().withCidrIp("150.150.150.150/32");

ipPermission.withIpv4Ranges(Arrays.asList(new IpRange[] {ipRange1, ipRange2}))
            .withIpProtocol("tcp")
            .withFromPort(22)
            .withToPort(22);
```

   您在中指定的所有條件IpPermission必須符合物件，才能允許輸入。

   使用 CIDR 標記法指定 IP 位址。如果您將通訊協定指定為 TCP/UDP，則必須提供來源連接埠和目的地連接埠。只有在指定 TCP 或 UDP 時，才能授權連接埠。

2. 建立和初始化AuthorizeSecurityGroupIngressRequest實例。使用withGroupName方法以指定安全群組名稱，並傳遞給IpPermission您之前初始化為withIpPermissions方法，如下：

```
AuthorizeSecurityGroupIngressRequest authorizeSecurityGroupIngressRequest =
    new AuthorizeSecurityGroupIngressRequest();

authorizeSecurityGroupIngressRequest.withGroupName("JavaSecurityGroup")
                                    .withIpPermissions(ipPermission);
```

3. 將請求物件傳遞給authorizeSecurityGroup傳入方法，如下：

```
amazonEC2Client.authorizeSecurityGroupIngress(authorizeSecurityGroupIngressRequest);
```

   如果你打電話authorizeSecurityGroupIngress對於已經授權輸入的 IP 位址，此方法會擲回例外狀況。建立和初始化新的IpPermission物件在呼叫前授權不同 IP、連接埠和通訊協定的輸入AuthorizeSecurityGroupIngress。

每當你打電話給authorizeSecurityGroup傳入或者authorizeSecurityGroup傳出方法中，規則會新增至您的安全性群組。

建立金鑰對

當您啟動 EC2 執行個體時，您必須指定金 key pair，然後在您連接到執行個體時，也必須指定金 key pair 的私有金鑰。您可以建立金 key pair，或使用啟動其他執行個體時已使用過的金 key pair 對。如需詳細資訊，請參閱「」Amazon EC2金鑰對中的Amazon EC2Linux 執行個體使用者指南。

1. 建立和初始化CreateKeyPairRequest實例。使用withKeyName方法設置 key pair 名稱，如下所示：

```
CreateKeyPairRequest createKeyPairRequest = new CreateKeyPairRequest();

createKeyPairRequest.withKeyName(keyName);
```

> **Important**
>
> 金鑰對名稱必須是唯一的。如果您嘗試創建與現有 key pair 具有相同密鑰名稱的 key pair，則會出現異常。

2. 傳遞請求物件給createKeyPair方法。該方式傳回CreateKeyPairResult實例，如下所示：

```
CreateKeyPairResult createKeyPairResult =
  amazonEC2Client.createKeyPair(createKeyPairRequest);
```

3. 調用結果對象的getKeyPair方法來獲取KeyPair物件。呼叫KeyPair物件的getKeyMaterial方法，獲取未加密 PEM 編碼私有金鑰，如下所示：

```
KeyPair keyPair = new KeyPair();

keyPair = createKeyPairResult.getKeyPair();

String privateKey = keyPair.getKeyMaterial();
```

執行Amazon EC2執行個體

使用以下步驟來啟動來自同一個的 Amazon 機器映像 (AMI) 中的一或多個配置完全相同的 EC2 執行個體。建立 EC2 執行個體後，您可以查看他們的狀態。EC2 執行個體正在執行之後，您便可以與它們連線。

1. 建立和初始化RunInstancesRequest實例。請確定 AMI、key pair 和安全性組，該安全組是您建立用户端物件時所指定存在於特定的區域中。

```
RunInstancesRequest runInstancesRequest =
   new RunInstancesRequest();

runInstancesRequest.withImageId("ami-a9d09ed1")
                   .withInstanceType(InstanceType.T1Micro)
                   .withMinCount(1)
                   .withMaxCount(1)
                   .withKeyName("my-key-pair")
                   .withSecurityGroups("my-security-group");
```

withImageId

- AMI 的 ID。要瞭解如何查找亞馬遜提供的公共 AMI 或創建您自己的 AMI，請參閱Amazon Machine Image (AMI)。

withInstanceType

- 執行個體類型與所指定的 AMI 相容。如需詳細資訊，請參閱「」執行個體類型中的Amazon EC2Linux 執行個體使用者指南。

withMinCount

- 要啟動執行個體的最少數量。如果這比 Amazon EC2 有更多的執行個體可以在目標可用區域啟動，Amazon EC2 啟動 0 個執行個體。

withMaxCount

- 要啟動執行個體的最大數量。如果這比 Amazon EC2 有更多的執行個體可以在目標可用區域啟動， Amazon EC2 盡可能啟動超過 `MinCount` 數量的執行個體。您可以啟動的範圍數量介於 1 到執行個體類型允許的執行個體最大數量。如需詳細資訊，請參我可以在Amazon EC2中的Amazon EC2常見問答集

withKeyName

- EC2 金鑰對的名稱。如果您未指定金鑰對而啟動執行個體，則就無法與它連線。如需詳細資訊，請參閱建立金鑰對。

withSecurityGroups

- 一個或多個安全群組。如需詳細資訊，請參閱「」建立Amazon EC2安全群組。

2. 啟動執行個體，方法是傳遞請求物件給runInstances方法。該方式傳回RunInstancesResult物件，如下所示：

```
RunInstancesResult result = amazonEC2Client.runInstances(
```

```
                                    runInstancesRequest);
```

執行個體之後，您便可以使用您的 key pair 來連接到該執行個體。如需詳細資訊，請參閱「」連線至您的 Linux 執行個體。在Amazon EC2Linux 執行個體使用者指南。

使用 IAM 角色授予AWS上的資源Amazon EC2

所有請求Amazon Web Services(AWS) 必須使用由AWS。您可以使用IAM 角色以便方便地授予AWS來自您的資源Amazon EC2實例。

本主題提供如何使用 IAM 角色搭配上執行之 Java SDK 應用程式的相關資訊Amazon EC2。如需 IAM 執行個體的詳細資訊，請參的 IAM 角色Amazon EC2中的Amazon EC2Linux 執行個體使用者指南。

默認提供程序鏈和 EC2 實例配置文件

如果您的應用程序創建AWS客户端使用默認構造函數，則客户端將使用默認憑據提供程序鏈(按照以下順序)：

1. 在 Java 系統屬性中：`aws.accessKeyId` 和 `aws.secretKey`。
2. 在系統環境變數中：`AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY`。
3. 在預設登入資料檔案中 (此檔案的位置因平台而異)。
4. 通過Amazon EC2容器服務（如果使用`AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`環境變量，並且安全管理器具有訪問該變量的權限。
5. 在執行個體描述檔登入資料中，這存在於與 EC2 執行個體的 IAM 角色關聯的執行個體中繼資料內。
6. 來自環境或容器的 Web 標識令牌憑據。

所以此執行個體配置文件步驟僅適用於在Amazon EC2執行個體，但在使用Amazon EC2實例。您也可以直接傳遞 InstanceProfileCredentialsProvider 執行個體給用戶端建構函數來取得執行個體描述檔登入資料，而無須繼續進行整個預設供應者鏈結。

例如：

```
AmazonS3 s3 = AmazonS3ClientBuilder.standard()
              .withCredentials(new InstanceProfileCredentialsProvider(false))
              .build();
```

使用此方法時，SDK 會檢索臨時AWS登入資料，此登入資料擁有與與Amazon EC2執行個體描述檔。雖然這些證書是臨時性的, 最終將過期,`InstanceProfileCredentialsProvider`定期重新整理，讓取得的登入資料持續允許存取AWS。

> **Important**
>
> 發生自動憑據刷新只要當你使用默認的客户端構造函數時，它會創建它自己的`InstanceProfileCredentialsProvider`作為默認提供程序鏈的一部分，或者當您傳遞`InstanceProfileCredentialsProvider`實例直接添加到客户端構造函數中。如果您使用其他方法獲取或傳遞實例配置文件憑據，則需要負責檢查和刷新過期的憑據。

如果客户端構造函數無法使用憑據提供程序鏈找到憑據，則會拋出AmazonClientException。

逐步解說：使用 IAM 角色適用於 EC2 執行個體

以下的逐步解說明如何從Amazon S3使用 IAM 角色管理存取權。

建立 IAM 角色

建立 IAM 角色，授與 Amazon S3 的唯讀存取權。

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇角色，然後建立新的角色。
3. 輸入角色的名稱，然後選擇 Next Step (下一步)。請記住此名稱，因為當您啟動Amazon EC2實例。
4. 在選取角色類型頁面，在 AWS 服務角色，選擇 Amazon EC2 。
5. 在設定許可頁面，在選取政策範本，選擇 Amazon S3唯讀存取，然後後續步驟。
6. 在檢閱頁面上，選擇建立角色。

啟動 EC2 執行個體時並指定 IAM 角色

您可以啟動Amazon EC2使用 IAM 角色使用Amazon EC2主控台或AWS SDK for Java。

- 要啟動Amazon EC2執行個體，請遵循入門Amazon EC2Linux 執行個體中的Amazon EC2Linux 執行個體使用者指南。

  當您到達 Review Instance Launch (檢閱執行個體啟動) 頁面時，選取 Edit instance details (編輯執行個體詳細資訊)。InIAM 角色下，選取先前建立的 IAM 角色。依照指示完成程序。

> **Note**
>
> 您需要建立或使用現有的安全群組與金鑰對，以連接到執行個體。

- 要啟動Amazon EC2使用 IAM 角色使用AWS SDK for Java，請參執行Amazon EC2執行個體。

創建您的應用程序

讓我們構建要在 EC2 實例上運行的示例應用程序。首先，創建一個可用於保存教程文件的目錄（例如`GetS3ObjectApp`。

接下來，複製AWS SDK for Java庫添加到新創建的目錄中。如果您已下載AWS SDK for Java到您的`~/Downloads`目錄中，您可以使用下列命令複製它們：

```
cp -r ~/Downloads/aws-java-sdk-{1.7.5}/lib .
cp -r ~/Downloads/aws-java-sdk-{1.7.5}/third-party .
```

打開一個新文件，稱之為`GetS3Object.java`，並新增下列程式碼：

```
import java.io.*;

import com.amazonaws.auth.*;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;

public class GetS3Object {
  private static final String bucketName = "text-content";
  private static final String key = "text-object.txt";

  public static void main(String[] args) throws IOException
  {
    AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

    try {
      System.out.println("Downloading an object");
      S3Object s3object = s3Client.getObject(
          new GetObjectRequest(bucketName, key));
      displayTextInputStream(s3object.getObjectContent());
```

```
    }
    catch(AmazonServiceException ase) {
      System.err.println("Exception was thrown by the service");
    }
    catch(AmazonClientException ace) {
      System.err.println("Exception was thrown by the client");
    }
  }

  private static void displayTextInputStream(InputStream input) throws IOException
  {
    // Read one text line at a time and display.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
    while(true)
    {
      String line = reader.readLine();
      if(line == null) break;
      System.out.println( "    " + line );
    }
    System.out.println();
  }
}
```

打開一個新文件，稱之為build.xml，並新增下列幾行：

```
<project name="Get {S3} Object" default="run" basedir=".">
  <path id="aws.java.sdk.classpath">
    <fileset dir="./lib" includes="**/*.jar"/>
    <fileset dir="./third-party" includes="**/*.jar"/>
    <pathelement location="lib"/>
    <pathelement location="."/>
  </path>

  <target name="build">
  <javac debug="true"
    includeantruntime="false"
    srcdir="."
    destdir="."
    classpathref="aws.java.sdk.classpath"/>
  </target>

  <target name="run" depends="build">
    <java classname="GetS3Object" classpathref="aws.java.sdk.classpath" fork="true"/>
```

```
  </target>
</project>
```

建置並執行修改後的程式。請注意，程序中沒有存儲憑據。因此，除非您有AWS憑據，則代碼將拋出AmazonServiceException。例如：

```
$ ant
Buildfile: /path/to/my/GetS3ObjectApp/build.xml

build:
  [javac] Compiling 1 source file to /path/to/my/GetS3ObjectApp

run:
   [java] Downloading an object
   [java] AmazonServiceException

BUILD SUCCESSFUL
```

將已編譯的程序轉移到 EC2 實例

將該計劃轉移到您的Amazon EC2實例使用安全副本（ ），以及AWS SDK for Java程式庫。命令序列類似如下所示。

```
scp -p -i {my-key-pair}.pem GetS3Object.class ec2-user@{public_dns}:GetS3Object.class
scp -p -i {my-key-pair}.pem build.xml ec2-user@{public_dns}:build.xml
scp -r -p -i {my-key-pair}.pem lib ec2-user@{public_dns}:lib
scp -r -p -i {my-key-pair}.pem third-party ec2-user@{public_dns}:third-party
```

> **Note**
>
> 根據您使用的 Linux 發行版，使用者名稱可能是「ec2 用户」，「根」或「Ubuntu」。若要獲取執行個體的公有 DNS 名稱，請開啟EC2 主控台並查找公有 DNS值 (位於)描述選項卡（例如ec2-198-51-100-1.compute-1.amazonaws.com。

在上述命令中：

- GetS3Object.class是你編譯的程序
- build.xml是使用於建置和執行程式的 ant 文件
- 該lib和third-party目錄是相應的庫文件夾來自AWS SDK for Java。

- 所以此`-r`切換指示`scp`應該執行遞歸副本的所有內容`library`和`third-party`目錄AWS SDK for Java分發。
- 所以此`-p`切換指示`scp`應該在源文件複製到目標時保留源文件的權限。

> **Note**
>
> 所以此`-p`切換僅適用於 Linux、macOS 或 Unix。如果要從 Windows 複製文件，則可能需要使用以下命令修復實例上的文件權限：

```
chmod -R u+rwx GetS3Object.class build.xml lib third-party
```

在 EC2 執行個體上執行範例程式

若要執行程式，請連接至Amazon EC2實例。如需詳細資訊，請參閱「」連線至您的 Linux 執行個體中的Amazon EC2Linux 執行個體使用者指南。

如果 **`ant`** 在您的執行個體上不可用，請使用下列命令進行安裝：

```
sudo yum install ant
```

然後，使用`ant`命中率：

```
ant run
```

該程序將寫入Amazon S3對象拖入命令視窗。

教學課程：Amazon EC2Spot Instances

概觀

競價型實例使您能夠對未使用的出價Amazon Elastic Compute Cloud(Amazon EC2) 容量與按需實例價格相比，容量不超過 90%，並且只要您的出價超過當前Spot 價格。Amazon EC2根據供需狀況定期更改 Spot 價格，且出價符合或超過 Spot 價格的顧客可存取可用的 Spot 執行個體。像隨需隨需執行個體和預留執行個體，Spot 執行個體提供另一個取得更多運算容量的方式。

競價型實例可以顯著降低您的Amazon EC2成本用於批次處理、科學研究、影像處理、影片編碼、數據和網路爬取、財務分析和測試。此外，Spot 執行個體讓您可以存取大量額外容量，可讓您存取不急需此容量的情況。

若要使用 Spot 執行個體，您可以提出 Spot 執行個體請求，並指定您願意支付每一個小時使用一個執行個體的最高預算；此為您的出價。如果您出價符合或超出目前競價型價格，則會履行您的請求，您的執行個體將會執行，直到選擇終止或競價型價格增加到高於出價 (以先到者為準)。

重要的是要注意：

- 通常您所付的費用會低於您的出價。Amazon EC2 定期依照需求及供應量調整 Spot 價格。無論出價多高，在該期間每個人支付相同的 Spot 價格。因此，您支付的費用可能會低於您的出價，但永遠不會超過您的出價。
- 如果您正在執行 Spot 執行個體且您的出價將不符合、也將不超過目前的 Spot 價格，則您的執行個體將被終止。這表示您會想要確保您的工作負載和應用程式具備足夠的彈性，以利用這個機會性容量。

Spot 執行個體的執行動作完全類似於其他 Amazon EC2 執行個體，也像其他 Amazon EC2 執行個體，Spot 執行個體在您不再需要它們時，可被終止。如果您終止了您的執行個體，不足一小時則按一小時支付費用，就像使用 (如您為隨需執行個體或預留執行個體所做的那樣)。然而，如果 Spot 價格上漲超過您的出價，並且您的執行個體由Amazon EC2，您將不會被收取任何部分使用小時的費用。

此教學課程示範如何使用AWS SDK for Java執行下列動作：

- 提交 Spot 請求
- 判斷 Spot 請求何時完成
- 取消 Spot 請求
- 終止關聯的執行個體

先決條件

若要使用本教學課程，您必須具有AWS SDK for Java，並且已滿足其基本的安裝前提條件。請參閱設定AWS SDK for Java如需詳細資訊，請參

步驟 1：建立您的登入資料

若要開始使用此程式碼範例，您需要設定AWS登入資料。請參閱設定AWS全權證書和區域促進發展如需如何執行此動作的詳細資訊。

> **Note**
>
> 我們建議您使用 IAM 用户的證書來提供這些值。如需詳細資訊，請參閱「」註冊AWS並建立 IAM 使用者。

現在您已設定您的設定，您可以開始使用示範中的程式碼。

步驟 2：設置安全組

一個安全群組扮演防火牆的角色，可控制允許進出一組執行個體的流量。默認情況下，啟動實例時沒有任何安全組，這意味着任何 TCP 端口上的所有傳入 IP 流量都將被拒絕。因此，在提交競價請求之前，我們將設置一個允許必要的網絡流量的安全組。為了本教程的目的，我們將創建一個名為「GettingStarted Stwork」的新安全組，該安全組允許來自運行應用程序的 IP 地址的安全外殼 (SSH) 流量。要設置新的安全組，您需要包含或運行以下以編程方式設置安全組的代碼示例。

在我們創建一個AmazonEC2客户端對象，我們創建一個CreateSecurityGroupRequest物件的名稱、「GettingStarted」和安全性羣組的描述。然後我們調用ec2.createSecurityGroupAPI 創建組。

要啟用對組的訪問，我們創建一個ipPermission對象的 IP 地址範圍設置為本地計算機子網的 CIDR 表示；IP 地址上的「/10」後綴表示指定 IP 地址的子網。我們還配置ipPermission物件與 TCP 協定和連接口 22 (SSH)。最後步驟是調用ec2.authorizeSecurityGroupIngress與我們的安全組的名稱和ipPermission物件。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Create a new security group.
try {
    CreateSecurityGroupRequest securityGroupRequest = new
 CreateSecurityGroupRequest("GettingStartedGroup", "Getting Started Security Group");
    ec2.createSecurityGroup(securityGroupRequest);
} catch (AmazonServiceException ase) {
    // Likely this means that the group is already created, so ignore.
    System.out.println(ase.getMessage());
}

String ipAddr = "0.0.0.0/0";

// Get the IP of the current host, so that we can limit the Security
```

```
// Group by default to the ip range associated with your subnet.
try {
    InetAddress addr = InetAddress.getLocalHost();

    // Get IP Address
    ipAddr = addr.getHostAddress()+"/10";
} catch (UnknownHostException e) {
}

// Create a range that you would like to populate.
ArrayList<String> ipRanges = new ArrayList<String>();
ipRanges.add(ipAddr);

// Open up port 22 for TCP traffic to the associated IP
// from above (e.g. ssh traffic).
ArrayList<IpPermission> ipPermissions = new ArrayList<IpPermission> ();
IpPermission ipPermission = new IpPermission();
ipPermission.setIpProtocol("tcp");
ipPermission.setFromPort(new Integer(22));
ipPermission.setToPort(new Integer(22));
ipPermission.setIpRanges(ipRanges);
ipPermissions.add(ipPermission);

try {
    // Authorize the ports to the used.
    AuthorizeSecurityGroupIngressRequest ingressRequest =
        new AuthorizeSecurityGroupIngressRequest("GettingStartedGroup",ipPermissions);
    ec2.authorizeSecurityGroupIngress(ingressRequest);
} catch (AmazonServiceException ase) {
    // Ignore because this likely means the zone has
    // already been authorized.
    System.out.println(ase.getMessage());
}
```

請注意，您只需運行此程序一次，即可建立新的安全羣組。

您也可以使用AWS Toolkit for Eclipse。請參閱從管理安全羣組AWS Cost Explorer如需詳細資訊，請參

步驟 3：提交您的 Sport 請求

若要提交 Spot 請求，您首先需要判斷執行個體的類型、Amazon 機器映像 (AMI)，以及您想要使用的最高金額。您還必須有我們先前配置的安全羣組，可讓您登入想要的執行個體。

有多種執行個體類型可供選擇；轉到Amazon EC2完整列表的執行個體類型。在本教程中，我們將使用 t1.micro，這是可用的最便宜的實例類型。接下來，我們將確定想要使用的 AMI 類型。我們將使用 AMI-a9d09ed1，這是我們編寫本教程時可用的最新的亞馬遜 Linux AMI。最新的 AMI 可能會隨着時間的推移而改變，但您始終可以通過以下步驟確定最新版本的 AMI：

1. 開啟 Amazon EC2 主控台。
2. 選擇啟動執行個體按鈕。
3. 第一個窗口顯示可用的 AMI。AMI ID 列在每個 AMI 標題旁邊。或者，您也可以使用`DescribeImages`API，但是利用該命令已超出本教學課程的範圍。

Spot 執行個體的出價有多種方式；若要取得各種方法的概觀，您應該查看競價型執行個體的出價影片。不過，在開始前，我們將描述三種常見的策略：出價競標，以確保成本低於隨需價格；根據計算結果的價值而出價競標；儘快出價競標越快取得運算容量。

- 降低低於隨需執行個體您可以批次處理任務，需要花費數小時或數天的時間完成。然而，您在開始時間和完成時都有彈性。您想要查看是否能以低於按需執行個體的成本完成此操作。您可以使用 AWS Management Console 或 Amazon EC2 API 檢查執行個體類型的 Spot 歷史價格。如需詳細資訊，請至 查閱 Spot 歷史價格。於一個給定的可用區域內分析您所想要的執行個體類型的歷史價格後，您的出價有兩個替代方式：
  - 您可以出價此 Spot 價格範圍的上限 (仍低於隨需執行個體的價格)、期待您的一次性 Spot 要求可以履行並執行，有一段足夠的連續時間完成工作。
  - 或者，您可以指定願意為 Spot 執行個體支付的金額, 以隨需執行個體價格的百分比表示, 並計畫如何透過一個持久性的請求聯合數個已長期啟動的執行個體。如果超過指定的價格，則 Spot 執行個體將會終止。(我們將說明如何使這個任務自動進行。)
- 不為結果支付超過其價值的金額您有一個要執行的資料正在處理任務。您很了解此任務結果的價值，換言之您知道成本為多少。在您分析執行個體類型的 Spot 歷史價格後，您選擇出價的運算時間成本金額不超過此任務結果的價值。您建立可以長久出價的方式，且允許它間歇性地執行，當 Spot 價格出現波動，或 Spot 價格低於您的出價時。
- 快速獲得運算容量出乎意料，您需要短期的額外容量，無法透過隨需執行個體取得。在您分析執行個體類型的 Spot 歷史價格後，您選擇出價高於歷史價格的金額，期望能快速履行您的出價，並持續運算直到完成為止。

在您選擇您的出價金額後，您可以要求 Spot 執行個體了。在此教學課程中，我們將出價競標隨需價格 (0.03 USD)，以最大化履行出價的機率。您可以判斷可用執行個體的類型和執行個體的隨需價格，方式是轉到Amazon EC2定價頁面。在 Spot 執行個體執行的這段時間，您將持續支付生

效的 Spot 價格。Spot 執行個體的價格由 Amazon EC2 制定，然後根據 Spot 執行個體容量的長期供需趨勢逐漸調整。您也可以指定願意為 Spot 執行個體支付的金額, 以隨需執行個體價格的百分比表示。若要請求 Spot 執行個體，您只需要使用您之前選擇的參數生成您的出價。我們從建立`RequestSpotInstanceRequest`物件。請求物件需要您想啟動的執行個體數量和出價。此外，您需要設定`LaunchSpecification`，其中含有執行個體類型、AMI ID 及想要使用的安全羣組。填充請求後，您可以調用`requestSpotInstances`方法`AmazonEC2Client`物件。以下範例示範如何請求 Spot 執行個體。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Setup the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specifications to the request.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
```

運行此程式碼將啟動新的 Spot 執行個體請求。還有其他方式可設定您的 Spot 請求。如需詳細資訊，請訪問教學課程：進階Amazon EC2Spot 請求管理或RequestSpotInstances類別中的AWS SDK for JavaAPI 參考。

> **Note**
>
> 您將需要為任何執行個體執行的 Spot 執行個體支付費用，因此確定您取消所有請求和終止您啟動的所有執行個體，以降低相關的費用。

步驟 4：判斷您的 Spot 價格之狀態

接著，我們想要創建代碼，等候直到 Spot 請求成為「活動」狀態，再進行最後一個步驟。要確定競價請求的狀態，我們將輪詢describeSpotInstanceRequests方法，以取得想要監控狀態的 Spot 請求 ID。

在步驟 2 中創建的請求 ID 嵌入到我們的`requestSpotInstances`請求。以下範例程式碼示範如何從`requestSpotInstances`響應並使用它們填充`ArrayList`。

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
List<SpotInstanceRequest> requestResponses = requestResult.getSpotInstanceRequests();

// Setup an arraylist to collect all of the request ids we want to
// watch hit the running state.
ArrayList<String> spotInstanceRequestIds = new ArrayList<String>();

// Add all of the request ids to the hashset, so we can determine when they hit the
// active state.
for (SpotInstanceRequest requestResponse : requestResponses) {
    System.out.println("Created Spot Request:
 "+requestResponse.getSpotInstanceRequestId());
    spotInstanceRequestIds.add(requestResponse.getSpotInstanceRequestId());
}
```

要監控您的請求 ID，請調用`describeSpotInstanceRequests`方法，以確定請求的狀態。然後循環直到請求不處於「打開」狀態。請注意，我們監視的狀態不是「打開」，而是「活動」狀態，因為如果您的請求參數存在問題，請求可以直接進入「關閉」。下面的代碼示例提供了有關如何完成此任務的詳細信息。

```
// Create a variable that will track whether there are any
// requests still in the open state.
boolean anyOpen;

do {
    // Create the describeRequest object with all of the request ids
```

```
    // to monitor (e.g. that we started).
    DescribeSpotInstanceRequestsRequest describeRequest = new
 DescribeSpotInstanceRequestsRequest();
    describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);

    // Initialize the anyOpen variable to false - which assumes there
    // are no requests open unless we find one that is still open.
    anyOpen=false;

    try {
        // Retrieve all of the requests we want to monitor.
        DescribeSpotInstanceRequestsResult describeResult =
 ec2.describeSpotInstanceRequests(describeRequest);
        List<SpotInstanceRequest> describeResponses =
 describeResult.getSpotInstanceRequests();

        // Look through each request and determine if they are all in
        // the active state.
        for (SpotInstanceRequest describeResponse : describeResponses) {
            // If the state is open, it hasn't changed since we attempted
            // to request it. There is the potential for it to transition
            // almost immediately to closed or cancelled so we compare
            // against open instead of active.
        if (describeResponse.getState().equals("open")) {
            anyOpen = true;
            break;
        }
    }
} catch (AmazonServiceException e) {
      // If we have an exception, ensure we don't break out of
      // the loop. This prevents the scenario where there was
      // blip on the wire.
      anyOpen = true;
    }

    try {
        // Sleep for 60 seconds.
        Thread.sleep(60*1000);
    } catch (Exception e) {
        // Do nothing because it woke up early.
    }
} while (anyOpen);
```

運行此代碼後，您的競價型實例請求將已完成或失敗，錯誤將輸出到屏幕。在這兩種情況中，我們都可以繼續執行下一個步驟，以清理所有活動中的請求和終止所有執行中的執行個體。

步驟 5：清除您的 Spot 請求和 Spot 執行個體請

最後，我們需要清除我們的請求和執行個體。取消任何未完成請求非常重要和終止任何執行個體。只取消請求並不會終止您的執行個體，也就是說您將繼續為它們付費。如果您終止您的執行個體，您的 Spot 請求可能會取消，但在部分情況中，像是如果您是使用長久出價的方式，那麼只終止您的執行個體還不足以停止重新履行您的出價。因此，最好的方式是取消所有作用中的競價和終止所有執行中的執行個體。

以下程式碼示範如何取消您的請求。

```
try {
    // Cancel requests.
    CancelSpotInstanceRequestsRequest cancelRequest =
       new CancelSpotInstanceRequestsRequest(spotInstanceRequestIds);
    ec2.cancelSpotInstanceRequests(cancelRequest);
} catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error cancelling instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

要終止任何未完成的實例，您需要與啟動它們的請求相關聯的實例 ID。下面的代碼示例採用我們的原始代碼來監控實例，並添加了一個`ArrayList`，我們在其中存儲與`describeInstance`回應。

```
// Create a variable that will track whether there are any requests
// still in the open state.
boolean anyOpen;
// Initialize variables.
ArrayList<String> instanceIds = new ArrayList<String>();

do {
   // Create the describeRequest with all of the request ids to
   // monitor (e.g. that we started).
   DescribeSpotInstanceRequestsRequest describeRequest = new
 DescribeSpotInstanceRequestsRequest();
   describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);
```

```
   // Initialize the anyOpen variable to false, which assumes there
   // are no requests open unless we find one that is still open.
   anyOpen = false;

   try {
         // Retrieve all of the requests we want to monitor.
         DescribeSpotInstanceRequestsResult describeResult =
            ec2.describeSpotInstanceRequests(describeRequest);

         List<SpotInstanceRequest> describeResponses =
            describeResult.getSpotInstanceRequests();

         // Look through each request and determine if they are all
         // in the active state.
         for (SpotInstanceRequest describeResponse : describeResponses) {
           // If the state is open, it hasn't changed since we
           // attempted to request it. There is the potential for
           // it to transition almost immediately to closed or
           // cancelled so we compare against open instead of active.
           if (describeResponse.getState().equals("open")) {
              anyOpen = true; break;
           }
           // Add the instance id to the list we will
           // eventually terminate.
           instanceIds.add(describeResponse.getInstanceId());
         }
   } catch (AmazonServiceException e) {
      // If we have an exception, ensure we don't break out
      // of the loop. This prevents the scenario where there
      // was blip on the wire.
      anyOpen = true;
   }

    try {
        // Sleep for 60 seconds.
        Thread.sleep(60*1000);
    } catch (Exception e) {
        // Do nothing because it woke up early.
    }
} while (anyOpen);
```

使用存儲在ArrayList使用下列程式碼片段終止所有正在運行的執行個體。

```
try {
    // Terminate instances.
    TerminateInstancesRequest terminateRequest = new
 TerminateInstancesRequest(instanceIds);
    ec2.terminateInstances(terminateRequest);
} catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

整合練習

為了整合這一切，我們提供了一種更面向對象的方法，它結合了我們展示的前面步驟：初始化 EC2 客戶端、提交競價請求、確定競價請求何時不再處於打開狀態，以及清理任何延遲的競價請求和相關實例。我們創建了一個名為`Requests`執行這些動作。

我們還創建了一個`GettingStartedApp`類，它有一個主方法，我們執行高級函數調用。具體來說，我們初始化`Requests`物件。我們提交 Spot 執行個體請求。然後，我們等待競價請求達到「活動」狀態。最後，我們清理請求和實例。

此範例的完整源程式碼可在GitHub。

恭喜您！您剛剛完成了開發競價型實例軟件的入門教程。AWS SDK for Java。

後續步驟

繼續進行教學課程：進階Amazon EC2Spot 請求管理。

教學課程：進階Amazon EC2Spot 請求管理

Amazon EC2競價型實例允許您對未使用的出價Amazon EC2容量並運行這些實例，只要您的出價超過當前Spot 價格。Amazon EC2根據供應和需求定期更改現貨價格。如需 Spot 執行個體的詳細資訊，請參競價型執行個體中的Amazon EC2Linux 執行個體使用者指南。

先決條件

若要使用本教學課程，您必須具有AWS SDK for Java，並且已滿足其基本的安裝前提條件。請參閱設定AWS SDK for Java以了解詳細資訊。

設置您的憑據

若要開始使用此程式碼範例，您需要設定AWS登入資料。請參閱設定AWS全權證書和區域促進發展如需如何執行此動作的詳細資訊。

> **Note**
>
> 建議您使用IAM用户提供這些值。如需詳細資訊，請參閱「」註冊AWS並建立IAM使用者。

現在您已配置好自己的設置，可開始使用示例中的代碼。

設定安全羣組

安全組可作為防火牆的角色，可控制允許進出執行個體的流量。默認情況下，啟動實例時沒有任何安全組，這意味着任何 TCP 端口上的所有傳入 IP 流量都將被拒絕。因此，在提交競價請求之前，我們將設置一個允許必要的網絡流量的安全組。在本教程中，我們將創建一個名為「GettingStarted Start」的新安全組，該安全組允許來自運行應用程序的 IP 地址的安全外殼 (SSH) 流量。要設置新的安全組，您需要包含或運行以下以編程方式設置安全組的代碼示例。

在我們創建AmazonEC2客户端對象，我們創建一個CreateSecurityGroupRequest物件的名稱、「GettingStarted」和安全性羣組的描述。然後我們調用ec2.createSecurityGroupAPI 來創建組。

要啟用對該組的訪問，我們創建一個ipPermission對象的 IP 地址範圍設置為本地計算機子網的 CIDR 表示；IP 地址上的「/10」後綴表示指定 IP 地址的子網。我們還配置ipPermission物件與 TCP 協定和端口 22 (SSH) 建立。最後步驟是呼叫ec2 .authorizeSecurityGroupIngress與我們的安全組的名稱和ipPermission物件。

（以下代碼與我們在第一個教程中使用的代碼相同。）

```
// Create the AmazonEC2Client object so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
                    .withCredentials(credentials)
                    .build();

// Create a new security group.
try {
    CreateSecurityGroupRequest securityGroupRequest =
        new CreateSecurityGroupRequest("GettingStartedGroup",
        "Getting Started Security Group");
```

```
    ec2.createSecurityGroup(securityGroupRequest);
} catch (AmazonServiceException ase) {
    // Likely this means that the group is already created, so ignore.
    System.out.println(ase.getMessage());
}

String ipAddr = "0.0.0.0/0";

// Get the IP of the current host, so that we can limit the Security Group
// by default to the ip range associated with your subnet.
try {
    // Get IP Address
    InetAddress addr = InetAddress.getLocalHost();
    ipAddr = addr.getHostAddress()+"/10";
}
catch (UnknownHostException e) {
    // Fail here...
}

// Create a range that you would like to populate.
ArrayList<String> ipRanges = new ArrayList<String>();
ipRanges.add(ipAddr);

// Open up port 22 for TCP traffic to the associated IP from
// above (e.g. ssh traffic).
ArrayList<IpPermission> ipPermissions = new ArrayList<IpPermission> ();
IpPermission ipPermission = new IpPermission();
ipPermission.setIpProtocol("tcp");
ipPermission.setFromPort(new Integer(22));
ipPermission.setToPort(new Integer(22));
ipPermission.setIpRanges(ipRanges);
ipPermissions.add(ipPermission);

try {
    // Authorize the ports to the used.
    AuthorizeSecurityGroupIngressRequest ingressRequest =
        new AuthorizeSecurityGroupIngressRequest(
            "GettingStartedGroup",ipPermissions);
    ec2.authorizeSecurityGroupIngress(ingressRequest);
}
catch (AmazonServiceException ase) {
    // Ignore because this likely means the zone has already
    // been authorized.
    System.out.println(ase.getMessage());
```

```
}
```

您可以查看整個代碼示例在advanced.CreateSecurityGroupApp.java程式碼範例。請注意，您只需運行此應用程式，即可建立新的安全羣組。

> **Note**
>
> 您也可以使用AWS Toolkit for Eclipse。請參閱從管理安全羣組AWS Cost Explorer中的AWS Toolkit for Eclipse用户指南瞭解更多信息。

詳細 Spot 執行個體請求建立選項

正如我們在教學課程：Amazon EC2競價型執行個體此外，您需要使用執行個體類型、Amazon 機器映像 (AMI)、以及最高金額生成請求。

讓我們首先創建一個RequestSpotInstanceRequest物件。請求物件需要您想要的執行個體數量和出價。此外，我們需要設置LaunchSpecification（其中含有執行個體的類型、AMI ID 及想要執行個體使用的安全組）。填充請求後，我們調用requestSpotInstances方法AmazonEC2Client物件。以下是如何請求 Spot 執行個體的範例。

（以下代碼與我們在第一個教程中使用的代碼相同。）

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
```

```
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

持久性請求與一次性請求

構建競價請求時，您可以指定多個可選參數。首先是您的請求是僅一次性還是持久性請求。默認情況下，它是一次性請求。一次性請求只能完成一次，在請求的實例終止後，請求將被關閉。只要沒有針對同一請求運行的競價型實例，就會考慮執行持久性請求。要指定請求的類型，您只需要在競價請求上設置類型。這可以透過下列程式碼來完成。

```
// Retrieves the credentials from an AWSCredentials.properties file.
AWSCredentials credentials = null;
try {
    credentials = new PropertiesCredentials(
        GettingStartedApp.class.getResourceAsStream("AwsCredentials.properties"));
}
catch (IOException e1) {
    System.out.println(
        "Credentials were not properly entered into AwsCredentials.properties.");
    System.out.println(e1.getMessage());
    System.exit(-1);
}

// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest =
    new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set the type of the bid to persistent.
```

```
requestRequest.setType("persistent");

// Set up the specifications of the launch. This includes the
// instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

限制請求的持續時間

您還可以選擇指定請求保持有效的時間長度。您可以為此期間指定開始時間和結束時間。默認情況下，競價請求從創建的那一刻起，將考慮執行競價請求，直到您完成或取消該請求。但是，如果需要，您可以限制有效期。下列程式碼中顯示了如何指定期限的範例。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set the valid start time to be two minutes from now.
Calendar cal = Calendar.getInstance();
cal.add(Calendar.MINUTE, 2);
requestRequest.setValidFrom(cal.getTime());
```

```
// Set the valid end time to be two minutes and two hours from now.
cal.add(Calendar.HOUR, 2);
requestRequest.setValidUntil(cal.getTime());

// Set up the specifications of the launch. This includes
// the instance type (e.g. t1.micro)

// and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon
// Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType("t1.micro");

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
```

分組您的Amazon EC2Spot 執行個體請求

您可以選擇以多種不同的方式對競價型實例請求進行分組。我們將介紹使用啟動組、可用區組和置放羣組的好處。

如果您希望確保您的競價型實例一起啟動和終止，則您可以選擇利用啟動組。啟動組是將一組出價分組在一起的標籤。啟動群組中的所有執行個體會同時啟動和終止。請注意，如果啟動組中的實例已完成，則不能保證使用同一啟動組啟動的新實例也將完成。下列程式碼範例中顯示如何設定啟動羣組的範例。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 5 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(5));
```

```
// Set the launch group.
requestRequest.setLaunchGroup("ADVANCED-DEMO-LAUNCH-GROUP");

// Set up the specifications of the launch. This includes
// the instance type (e.g. t1.micro) and the latest Amazon Linux
// AMI id available. Note, you should always use the latest
// Amazon Linux AMI id or another of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

如果您希望確保請求中的所有實例都在同一個可用區中啟動，並且您不關心哪個實例，則可以利用可用區組。可用區組是一個標籤，它將一組實例分組在同一可用區中。共享一個可用區組並同時履行的所有實例都將在同一可用區中啟動。以下是如何設定可用區域羣組的範例。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 5 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(5));

// Set the availability zone group.
requestRequest.setAvailabilityZoneGroup("ADVANCED-DEMO-AZ-GROUP");

// Set up the specifications of the launch.  This includes the instance
// type (e.g.  t1.micro) and the latest Amazon Linux AMI id available.
```

```
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

您可以為 Spot 執行個體指定您想要的可用區。下列程式碼範例示範如何設定可用區。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);
```

```
// Set up the availability zone to use. Note we could retrieve the
// availability zones using the ec2.describeAvailabilityZones() API. For
// this demo we will just use us-east-1a.
SpotPlacement placement = new SpotPlacement("us-east-1b");
launchSpecification.setPlacement(placement);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

最後，您可以指定置放羣組如果您使用的是高性能計算 (HPC) 競價型實例，如集羣計算實例或集羣 GPU 實例。置放羣組可在執行個體之間，提供低延遲和高頻寬連線能力。以下是如何設定置放羣組的範例。

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.

LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Set up the placement group to use with whatever name you desire.
// For this demo we will just use "ADVANCED-DEMO-PLACEMENT-GROUP".
```

```
SpotPlacement placement = new SpotPlacement();
placement.setGroupName("ADVANCED-DEMO-PLACEMENT-GROUP");
launchSpecification.setPlacement(placement);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
    ec2.requestSpotInstances(requestRequest);
```

本節中顯示的所有參數均為可選參數。同樣重要的是要認識到，這些參數中的大多數參數（您的出價是一次性出價還是持久性出價除外）都可以降低出價履行的可能性。因此，只有在需要時才利用這些選項是非常重要的。前面的所有代碼示例都被合併為一個長代碼示例，該示例可在`com.amazonaws.codesamples.advanced.InlineGettingStartedCodeSampleApp.java`類別。

如何在中斷或終止後保留根分區

管理競價型實例中斷的最簡單方法之一是確保您的數據被檢查到 Amazon 彈性塊存儲（AmazonAmazon EBS）在常規節奏上的音量。通過定期檢查點，如果出現中斷，則只會丟失自上次檢查點以來創建的數據（假設兩者之間沒有執行其他非冪等操作）。為了使此過程更加簡單，您可以配置競價請求，以確保您的根分區不會在中斷或終止時被刪除。我們在以下示例中插入了新代碼，說明如何啟用此方案。

在添加的代碼中，我們創建一個`BlockDeviceMapping`對象並設置其關聯的Amazon Elastic Block Store(Amazon EBS）設置為Amazon EBS對象，我們已配置為`not`如果競價型實例終止，則刪除。然後我們添加這個`BlockDeviceMapping`添加到我們在啟動規範中包含的映射的陣列列表。

```
// Retrieves the credentials from an AWSCredentials.properties file.
AWSCredentials credentials = null;
try {
    credentials = new PropertiesCredentials(
        GettingStartedApp.class.getResourceAsStream("AwsCredentials.properties"));
}
catch (IOException e1) {
    System.out.println(
        "Credentials were not properly entered into AwsCredentials.properties.");
    System.out.println(e1.getMessage());
    System.exit(-1);
}
```

```
// Create the AmazonEC2 client so we can call various APIs.
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

// Initializes a Spot Instance Request
RequestSpotInstancesRequest requestRequest = new RequestSpotInstancesRequest();

// Request 1 x t1.micro instance with a bid price of $0.03.
requestRequest.setSpotPrice("0.03");
requestRequest.setInstanceCount(Integer.valueOf(1));

// Set up the specifications of the launch. This includes the instance
// type (e.g. t1.micro) and the latest Amazon Linux AMI id available.
// Note, you should always use the latest Amazon Linux AMI id or another
// of your choosing.
LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.setImageId("ami-a9d09ed1");
launchSpecification.setInstanceType(InstanceType.T1Micro);

// Add the security group to the request.
ArrayList<String> securityGroups = new ArrayList<String>();
securityGroups.add("GettingStartedGroup");
launchSpecification.setSecurityGroups(securityGroups);

// Create the block device mapping to describe the root partition.
BlockDeviceMapping blockDeviceMapping = new BlockDeviceMapping();
blockDeviceMapping.setDeviceName("/dev/sda1");

// Set the delete on termination flag to false.
EbsBlockDevice ebs = new EbsBlockDevice();
ebs.setDeleteOnTermination(Boolean.FALSE);
blockDeviceMapping.setEbs(ebs);

// Add the block device mapping to the block list.
ArrayList<BlockDeviceMapping> blockList = new ArrayList<BlockDeviceMapping>();
blockList.add(blockDeviceMapping);

// Set the block device mapping configuration in the launch specifications.
launchSpecification.setBlockDeviceMappings(blockList);

// Add the launch specification.
requestRequest.setLaunchSpecification(launchSpecification);

// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult =
```

```
    ec2.requestSpotInstances(requestRequest);
```

假設您希望在啟動時將此卷重新連接到您的實例，您還可以使用塊儲存設備映射設置。或者，如果您附加了非根分區，則可以指定 AmazonAmazon EBS您希望在競價型實例恢復後連接到競價型實例的卷。您只需通過在`EbsBlockDevice`和備用設備名稱`BlockDeviceMapping`物件。通過利用塊儲存設備映射，可以更輕鬆地引導實例。

使用根分區檢查關鍵數據是管理實例中斷可能性的好方法。有關管理中斷可能性的更多方法，請訪問管理中斷影片。

如何標記您的競價請求和實例

新增標籤到Amazon EC2資源可以簡化雲端基礎設施的管理。標籤是一種元數據形式，可用於創建用户友好的名稱，增強可搜索性，並改善多個用户之間的協調。您也可以使用標籤來自動化流程的腳本和部分流程。閱讀有關標記的更多信息Amazon EC2資源，請前往使用標籤中的Amazon EC2Linux 執行個體使用者指南。

標記 請求

要向競價請求添加標籤，您需要標記它們之後他們已經提出要求. 來自的返回值`requestSpotInstances()`為您提供RequestSpotInstancesResult對象，您可以使用它來獲取用於標記的競價請求 ID：

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResult requestResult = ec2.requestSpotInstances(requestRequest);
List<SpotInstanceRequest> requestResponses = requestResult.getSpotInstanceRequests();

// A list of request IDs to tag
ArrayList<String> spotInstanceRequestIds = new ArrayList<String>();

// Add the request ids to the hashset, so we can determine when they hit the
// active state.
for (SpotInstanceRequest requestResponse : requestResponses) {
    System.out.println("Created Spot Request:
 "+requestResponse.getSpotInstanceRequestId());
    spotInstanceRequestIds.add(requestResponse.getSpotInstanceRequestId());
}
```

獲得 ID 後，您可以通過將請求的 ID 添加到CreateTagsRequest並呼叫Amazon EC2客户端的`createTags()`方法：

```
// The list of tags to create
ArrayList<Tag> requestTags = new ArrayList<Tag>();
requestTags.add(new Tag("keyname1","value1"));

// Create the tag request
CreateTagsRequest createTagsRequest_requests = new CreateTagsRequest();
createTagsRequest_requests.setResources(spotInstanceRequestIds);
createTagsRequest_requests.setTags(requestTags);

// Tag the spot request
try {
    ec2.createTags(createTagsRequest_requests);
}
catch (AmazonServiceException e) {
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

標記執行個體

與競價請求本身類似，您只能在創建實例後對其進行標記，這將在滿足競價請求後發生（它不再位於開放狀態）。

您可以透過呼叫Amazon EC2客戶端的`describeSpotInstanceRequests()`方法與DescribeSpotInstanceRequestsRequest物件。返回的DescribeSpotInstanceRequestsResult物件包含SpotInstanceRequest對象，可用於查詢 Spot 請求的狀態，並在其執行個體不再位於開放狀態。

競價請求不再打開後，您可以從`SpotInstanceRequest`對象，通過調用其`getInstanceId()`方法。

```
boolean anyOpen; // tracks whether any requests are still open

// a list of instances to tag.
ArrayList<String> instanceIds = new ArrayList<String>();

do {
    DescribeSpotInstanceRequestsRequest describeRequest =
        new DescribeSpotInstanceRequestsRequest();
    describeRequest.setSpotInstanceRequestIds(spotInstanceRequestIds);
```

```
    anyOpen=false; // assume no requests are still open

    try {
        // Get the requests to monitor
        DescribeSpotInstanceRequestsResult describeResult =
            ec2.describeSpotInstanceRequests(describeRequest);

        List<SpotInstanceRequest> describeResponses =
            describeResult.getSpotInstanceRequests();

        // are any requests open?
        for (SpotInstanceRequest describeResponse : describeResponses) {
                if (describeResponse.getState().equals("open")) {
                    anyOpen = true;
                    break;
                }
                // get the corresponding instance ID of the spot request
                instanceIds.add(describeResponse.getInstanceId());
        }
    }
    catch (AmazonServiceException e) {
        // Don't break the loop due to an exception (it may be a temporary issue)
        anyOpen = true;
    }

    try {
        Thread.sleep(60*1000); // sleep 60s.
    }
    catch (Exception e) {
        // Do nothing if the thread woke up early.
    }
} while (anyOpen);
```

現在，您可以標記返回的實例：

```
// Create a list of tags to create
ArrayList<Tag> instanceTags = new ArrayList<Tag>();
instanceTags.add(new Tag("keyname1","value1"));

// Create the tag request
CreateTagsRequest createTagsRequest_instances = new CreateTagsRequest();
createTagsRequest_instances.setResources(instanceIds);
```

```
createTagsRequest_instances.setTags(instanceTags);

// Tag the instance
try {
    ec2.createTags(createTagsRequest_instances);
}
catch (AmazonServiceException e) {
    // Write out any exceptions that may have occurred.
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

取消競價請求和終止實例

取消 Spot 請求

若要取消 Spot 執行個體請求，請呼叫`cancelSpotInstanceRequests`在Amazon EC2客戶端具有[CancelSpotInstanceRequestsRequest](#)物件。

```
try {
    CancelSpotInstanceRequestsRequest cancelRequest = new
 CancelSpotInstanceRequestsRequest(spotInstanceRequestIds);
    ec2.cancelSpotInstanceRequests(cancelRequest);
} catch (AmazonServiceException e) {
    System.out.println("Error cancelling instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

終止競價型實例

您可以通過將正在運行的競價型實例的 ID 傳遞給Amazon EC2客戶端的`terminateInstances()`方法。

```
try {
    TerminateInstancesRequest terminateRequest = new
 TerminateInstancesRequest(instanceIds);
```

```
    ec2.terminateInstances(terminateRequest);
} catch (AmazonServiceException e) {
    System.out.println("Error terminating instances");
    System.out.println("Caught Exception: " + e.getMessage());
    System.out.println("Reponse Status Code: " + e.getStatusCode());
    System.out.println("Error Code: " + e.getErrorCode());
    System.out.println("Request ID: " + e.getRequestId());
}
```

整合練習

為了將這些結合在一起，我們提供了一個更面向對象的方法，將本教程中展示的步驟結合到一個易於使用的類中。我們實例化一個名為Requests來執行這些動作。我們還創建了GettingStartedApp類，它有一個主方法，我們執行高級函數調用。

您可以在以下網址查看或下載本範例的完整源程式碼：GitHub。

恭喜您！您已完成高級請求功能教程，該教程用於開發競價型實例軟件AWS SDK for Java。

管理 Amazon EC2 執行個體

建立執行個體

建立新Amazon EC2實例通過調用卓越亞馬遜客户端的runInstances方法，為其提供RunInstancesRequest包含Amazon Machine Image (AMI)使用，並使用執行個體類型。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.InstanceType;
import com.amazonaws.services.ec2.model.RunInstancesRequest;
import com.amazonaws.services.ec2.model.RunInstancesResult;
import com.amazonaws.services.ec2.model.Tag;
```

Code

```
RunInstancesRequest run_request = new RunInstancesRequest()
    .withImageId(ami_id)
    .withInstanceType(InstanceType.T1Micro)
    .withMaxCount(1)
```

```
    .withMinCount(1);

RunInstancesResult run_response = ec2.runInstances(run_request);

String reservation_id =
 run_response.getReservation().getInstances().get(0).getInstanceId();
```

請參完整範例。

啟動執行個體

啟動Amazon EC2實例中，請調用卓越亞馬遜客户端的`startInstances`方法，為其提供StartInstancesRequest，其中包含要啟動的執行個體之 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.StartInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StartInstancesRequest request = new StartInstancesRequest()
    .withInstanceIds(instance_id);

ec2.startInstances(request);
```

請參完整範例。

停止執行個體

若要停止Amazon EC2實例中，請調用卓越亞馬遜客户端的`stopInstances`方法，為其提供StopInstancesRequest，其中包含要停止的執行個體之 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
```

```
import com.amazonaws.services.ec2.model.StopInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

StopInstancesRequest request = new StopInstancesRequest()
    .withInstanceIds(instance_id);

ec2.stopInstances(request);
```

請參完整範例。

重新啟動 執行個體

若要重新啟動Amazon EC2實例中，請調用卓越亞馬遜客户端的`rebootInstances`方法，為其提供RebootInstancesRequest，其中包含要重新啟動的實例的 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.RebootInstancesRequest;
import com.amazonaws.services.ec2.model.RebootInstancesResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

RebootInstancesRequest request = new RebootInstancesRequest()
    .withInstanceIds(instance_id);

RebootInstancesResult response = ec2.rebootInstances(request);
```

請參完整範例。

描述 執行個體

若要列出您的執行個體，請建立DescribeInstancesRequest並致電卓越亞馬遜客户端的`describeInstances`方法。它將返回DescribeInstancesResult物件，您可以使用此物件列出Amazon EC2執行個體。

執行個體依照保留分組。每個保留對應到呼叫 `startInstances`，用以啟動執行個體。若要列出您的執行個體，您必須先呼叫DescribeInstancesResult類別getReservations' method, and then call `getInstances在每個返回保留物件。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeInstancesRequest;
import com.amazonaws.services.ec2.model.DescribeInstancesResult;
import com.amazonaws.services.ec2.model.Instance;
import com.amazonaws.services.ec2.model.Reservation;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();
boolean done = false;

DescribeInstancesRequest request = new DescribeInstancesRequest();
while(!done) {
    DescribeInstancesResult response = ec2.describeInstances(request);

    for(Reservation reservation : response.getReservations()) {
        for(Instance instance : reservation.getInstances()) {
            System.out.printf(
                "Found instance with id %s, " +
                "AMI %s, " +
                "type %s, " +
                "state %s " +
                "and monitoring state %s",
                instance.getInstanceId(),
                instance.getImageId(),
                instance.getInstanceType(),
                instance.getState().getName(),
                instance.getMonitoring().getState());
        }
    }

    request.setNextToken(response.getNextToken());

    if(response.getNextToken() == null) {
        done = true;
    }
```

```
}
```

結果會分頁；您可以將結果物件的`getNextToken`方法添加到原始請求對象的`setNextToken`方法，然後在下次調用`describeInstances`。

請參完整範例。

監控執行個體

您可以監控 Amazon EC2 執行個體的各個面向，例如 CPU 和網路使用率、可用記憶體和剩餘磁碟空間。若要進一步了解執行個體監控，請參監控Amazon EC2中的Amazon EC2Linux 執行個體使用者指南。

要開始監控實例，您必須創建MonitorInstancesRequest，並將要監控之執行個體的 ID 建立，然後將其傳遞給卓越亞馬遜客户端的`monitorInstances`方法。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.MonitorInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

MonitorInstancesRequest request = new MonitorInstancesRequest()
        .withInstanceIds(instance_id);

ec2.monitorInstances(request);
```

請參完整範例。

停止執行個體監控

要停止監控實例，請創建UnmonitorInstancesRequest，以停止監控之執行個體之 ID，然後將其傳遞給卓越亞馬遜客户端的`unmonitorInstances`方法。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.UnmonitorInstancesRequest;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

UnmonitorInstancesRequest request = new UnmonitorInstancesRequest()
    .withInstanceIds(instance_id);

ec2.unmonitorInstances(request);
```

請參完整範例。

詳細資訊

- RunInstances中的Amazon EC2API 參考
- DescribeInstances中的Amazon EC2API 參考
- StartInstances中的Amazon EC2API 參考
- StopInstances中的Amazon EC2API 參考
- RebootInstances中的Amazon EC2API 參考
- MonitorInstances中的Amazon EC2API 參考
- UnmonitorInstances中的Amazon EC2API 參考

在 Amazon EC2 中使用彈性 IP 地址

EC2-Classic 正在淘汰

> **Warning**
>
> 我們將於 2022 年 8 月 15 日淘汰 EC2-Classic。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱「」從 EC2-Classic 遷移至 VPC中的Amazon EC2 Linux 執行個體使用者指南或Amazon EC2 Windows 執行個體使用者指南。另請參閱部落格文章EC2-Classic 網路正在淘汰-本文介紹如何準備。

配置彈性 IP 地址

若要使用彈性 IP 地址，您可以先將一個地址配置給帳戶，再將其與您的執行個體或網路介面建立關聯。

若要配彈性 IP 地址，請呼叫 Amazon onec2client`allocateAddress`方法使用AllocateAddressRequest包含網路類型 (傳統 EC2 或 VPC) 的物件。

返回的AllocateAddressResult包含配置 ID，可透過 Associate 將地址關聯至執行個體。AssociateAddressRequest到亞馬遜 2 客戶端`associateAddress`方法。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AllocateAddressRequest;
import com.amazonaws.services.ec2.model.AllocateAddressResult;
import com.amazonaws.services.ec2.model.AssociateAddressRequest;
import com.amazonaws.services.ec2.model.AssociateAddressResult;
import com.amazonaws.services.ec2.model.DomainType;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

AllocateAddressRequest allocate_request = new AllocateAddressRequest()
    .withDomain(DomainType.Vpc);

AllocateAddressResult allocate_response =
    ec2.allocateAddress(allocate_request);

String allocation_id = allocate_response.getAllocationId();

AssociateAddressRequest associate_request =
    new AssociateAddressRequest()
        .withInstanceId(instance_id)
        .withAllocationId(allocation_id);

AssociateAddressResult associate_response =
    ec2.associateAddress(associate_request);
```

請參閱完整的範例。

描述彈性 IP 地址

若要列出指派給您帳戶的彈性 IP 地址，請呼叫 Amazon onec2clientdescribeAddresses方法。其會傳回DescribeAddressesResult您可以使用它來取得清單Address代表您帳戶中彈性 IP 地址的物件。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.Address;
import com.amazonaws.services.ec2.model.DescribeAddressesResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DescribeAddressesResult response = ec2.describeAddresses();

for(Address address : response.getAddresses()) {
    System.out.printf(
            "Found address with public IP %s, " +
            "domain %s, " +
            "allocation id %s " +
            "and NIC id %s",
            address.getPublicIp(),
            address.getDomain(),
            address.getAllocationId(),
            address.getNetworkInterfaceId());
}
```

請參閱完整的範例。

釋放彈性 IP 地址

若要釋彈性 IP 地址，請呼叫亞馬遜 2clientreleaseAddress方法，傳遞一個ReleaseAddressRequest包含您要釋放彈性 IP 地址的配置 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.ReleaseAddressRequest;
```

```
import com.amazonaws.services.ec2.model.ReleaseAddressResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

ReleaseAddressRequest request = new ReleaseAddressRequest()
    .withAllocationId(alloc_id);

ReleaseAddressResult response = ec2.releaseAddress(request);
```

釋出彈性 IP 地址後，系統會將該地址釋放到AWSIP 地址集區，您以後可能會無法使用。請務必更新您的 DNS 記錄以及與該地址通訊的任何伺服器或裝置。如果您嘗試釋出已經發佈的彈性 IP 地址，您將獲得AuthFailure如果該地址已分配給另一個地址，則會出現錯誤AWS 帳戶。

如果您使用 EC2-Classic 或預設 VPC，則釋出彈性 IP 地址會自動將其與任何已關聯的執行個體取消關聯。若要取消關聯彈性 IP 地址但不將其釋出，請使用 Amazon onec2clientdisassociateAddress方法。

如果您是使用非預設 VPC，在嘗試釋出該彈性 IP 地址前，您必須先使用 disassociateAddress 將其取消關聯。否則為Amazon EC2傳回錯誤 (InvalidIPAddress。InUse。

請參閱完整的範例。

詳細資訊

- 彈性 IP 地址中的Amazon EC2Linux 執行個體使用者指南
- AllocateAddress中的Amazon EC2API 參考
- DescribeAddresses中的Amazon EC2API 參考
- ReleaseAddress中的Amazon EC2API 參考

使用區域與可用區域

描述區域

若要列出您賬户可用的區域，請呼叫卓越亞馬遜客户端的describeRegions方法。其會傳回DescribeRegionsResult。呼叫傳回物件的 getRegions 方法以取得代表每個區域的 Region 物件清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeRegionsResult;
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

```
DescribeRegionsResult regions_response = ec2.describeRegions();

for(Region region : regions_response.getRegions()) {
    System.out.printf(
        "Found region %s " +
        "with endpoint %s",
        region.getRegionName(),
        region.getEndpoint());
}
```

參完整範例。

描述可用區域

若要列出您帳戶每個可用的可用區域，請呼叫卓越亞馬遜客戶端的`describeAvailabilityZones`方法。其會傳回DescribeAvailabilityZonesResult。呼叫其 `getAvailabilityZones` 方法以取得代表每個可用區域的 AvailabilityZone 物件清單。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeRegionsResult;
import com.amazonaws.services.ec2.model.Region;
import com.amazonaws.services.ec2.model.AvailabilityZone;
import com.amazonaws.services.ec2.model.DescribeAvailabilityZonesResult;
```

Code

```
DescribeAvailabilityZonesResult zones_response =
```

```
    ec2.describeAvailabilityZones();

for(AvailabilityZone zone : zones_response.getAvailabilityZones()) {
    System.out.printf(
        "Found availability zone %s " +
        "with status %s " +
        "in region %s",
        zone.getZoneName(),
        zone.getState(),
        zone.getRegionName());
}
```

參完整範例。

描述帳戶

若要描述您的帳戶，請呼叫卓越亞馬遜客户端的describeAccountAttributes方法。這個方法會傳回DescribeAccountAttributesResult物件。叫用此物件的 getAccountAttributes 方法來取得 AccountAttribute 物件的清單。您可以逐一查看清單以擷取 AccountAttribute 物件。

您可以透過叫用 AccountAttribute 對象的 getAttributeValues 方法來取得您帳戶的屬性值。此方法會傳回 AccountAttributeValue 物件的清單。您可以逐一查看第二個清單以顯示屬性值 (請參閱下列程式碼範例)。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.AccountAttributeValue;
import com.amazonaws.services.ec2.model.DescribeAccountAttributesResult;
import com.amazonaws.services.ec2.model.AccountAttribute;
import java.util.List;
import java.util.ListIterator;
```

Code

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

try{
    DescribeAccountAttributesResult accountResults = ec2.describeAccountAttributes();
    List<AccountAttribute> accountList = accountResults.getAccountAttributes();
```

```
    for (ListIterator iter = accountList.listIterator(); iter.hasNext(); ) {

        AccountAttribute attribute = (AccountAttribute) iter.next();
        System.out.print("\n The name of the attribute is
 "+attribute.getAttributeName());
        List<AccountAttributeValue> values = attribute.getAttributeValues();

         //iterate through the attribute values
        for (ListIterator iterVals = values.listIterator(); iterVals.hasNext(); ) {
            AccountAttributeValue myValue = (AccountAttributeValue) iterVals.next();
            System.out.print("\n The value of the attribute is
 "+myValue.getAttributeValue());
        }
    }
    System.out.print("Done");
}
catch (Exception e)
{
    e.getStackTrace();
}
```

請參閱 GitHub 上的完整範例。

其他資訊

- 區域與可用區域中的Amazon EC2Linux 執行個體使用者指南
- DescribeRegions中的Amazon EC2API 參考
- DescribeAvailabilityZones中的Amazon EC2API 參考

使用 Amazon EC2 金鑰對

建立金鑰對

若要建立資訊 key pair，請呼叫卓越亞馬遜客户端的`createKeyPair`方法並搭配CreateKeyPairRequest，其中包含密鑰的名稱。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
```

```
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateKeyPairRequest;
import com.amazonaws.services.ec2.model.CreateKeyPairResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

CreateKeyPairRequest request = new CreateKeyPairRequest()
    .withKeyName(key_name);

CreateKeyPairResult response = ec2.createKeyPair(request);
```

參以下資源完整範例。

描述金鑰對

若要列出您的鑰對或取得相關資訊，請呼叫 Amazonec2Client 的`describeKeyPairs`方法。其會傳回DescribeKeyPairsResult，您可通過呼叫它的`getKeyPairs`方法，傳回KeyPairInfo物件。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeKeyPairsResult;
import com.amazonaws.services.ec2.model.KeyPairInfo;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DescribeKeyPairsResult response = ec2.describeKeyPairs();

for(KeyPairInfo key_pair : response.getKeyPairs()) {
    System.out.printf(
        "Found key pair with name %s " +
        "and fingerprint %s",
        key_pair.getKeyName(),
        key_pair.getKeyFingerprint());
}
```

參以下資源完整範例。

刪除金鑰對

要刪除 key pair，請調用卓越亞馬遜客户端的deleteKeyPair方法，將其傳遞一個DeleteKeyPairRequest，其中包含要刪除的 key pair 的名稱。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteKeyPairRequest;
import com.amazonaws.services.ec2.model.DeleteKeyPairResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DeleteKeyPairRequest request = new DeleteKeyPairRequest()
    .withKeyName(key_name);

DeleteKeyPairResult response = ec2.deleteKeyPair(request);
```

參以下資源完整範例。

詳細資訊

- Amazon EC2金鑰對中的Amazon EC2Linux 執行個體使用者指南
- CreateKeyPair中的Amazon EC2API 參考
- DescribeKeyPairs中的Amazon EC2API 參考
- DeleteKeyPair中的Amazon EC2API 參考

在 Amazon EC2 中使用安全群組

建立安全群組

要創建一個安全組，請使用包含密鑰名稱的亞馬遜客戶端CreateSecurityGroupRequest的createSecurityGroup方法調用。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

CreateSecurityGroupRequest create_request = new
    CreateSecurityGroupRequest()
        .withGroupName(group_name)
        .withDescription(group_desc)
        .withVpcId(vpc_id);

CreateSecurityGroupResult create_response =
    ec2.createSecurityGroup(create_request);
```

請參閱完整範例。

設定安全群組

安全群組可以同時控制 Amazon EC2 執行個體上的傳入 (輸入) 和傳出 (輸出) 流量。

若要將輸入規則新增至安全性群組，請使用 AmazonEC2ClientauthorizeSecurityGroupIngress 方法，提供安全性群組的名稱以及您要在AuthorizeSecurityGroupIngressRequest物件中指派給它的存取規則 (IpPermission)。以下範例說明如何將 IP 許可新增至安全群組。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupResult;
```

Code

```
IpRange ip_range = new IpRange()
```

```
    .withCidrIp("0.0.0.0/0");

IpPermission ip_perm = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(80)
    .withFromPort(80)
    .withIpv4Ranges(ip_range);

IpPermission ip_perm2 = new IpPermission()
    .withIpProtocol("tcp")
    .withToPort(22)
    .withFromPort(22)
    .withIpv4Ranges(ip_range);

AuthorizeSecurityGroupIngressRequest auth_request = new
    AuthorizeSecurityGroupIngressRequest()
        .withGroupName(group_name)
        .withIpPermissions(ip_perm, ip_perm2);

AuthorizeSecurityGroupIngressResult auth_response =
    ec2.authorizeSecurityGroupIngress(auth_request);
```

若要將輸出規則新增至安全性群組，AuthorizeSecurityGroupEgressRequest請在 AmazonEC2Client 的`authorizeSecurityGroupEgress`方法中提供類似的資料。

請參閱完整範例。

描述安全群組

要描述您的安全組或獲取有關它們的信息，請調用 AmazonEC2Client 的`describeSecurityGroups`方法。它返回一個 DescribeSecurityGroupsResult，您可以通過調用其`getSecurityGroups`方法來訪問安全組列表的安全組列表，該方法返回SecurityGroup對象的列表。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsRequest;
import com.amazonaws.services.ec2.model.DescribeSecurityGroupsResult;
```

Code

```
final String USAGE =
    "To run this example, supply a group id\n" +
    "Ex: DescribeSecurityGroups <group-id>\n";

if (args.length != 1) {
    System.out.println(USAGE);
    System.exit(1);
}

String group_id = args[0];
```

請參閱完整範例。

刪除安全群組

要刪除安全組，請調用 AmazonEC2Client 的deleteSecurityGroup方法，將其傳遞給包DeleteSecurityGroupRequest含要刪除的安全組的 ID。

匯入

```
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.DeleteSecurityGroupRequest;
import com.amazonaws.services.ec2.model.DeleteSecurityGroupResult;
```

Code

```
final AmazonEC2 ec2 = AmazonEC2ClientBuilder.defaultClient();

DeleteSecurityGroupRequest request = new DeleteSecurityGroupRequest()
    .withGroupId(group_id);

DeleteSecurityGroupResult response = ec2.deleteSecurityGroup(request);
```

請參閱完整範例。

詳細資訊

- Amazon EC2Linux 執行個體Amazon EC2使用者指南中的安全性群組
- 在 Linux 執行個體的Amazon EC2使用者指南中授權 Linux 執行個體的輸入流量

- CreateSecurityGroup在Amazon EC2 API 參考資料中
- DescribeSecurityGroups在Amazon EC2 API 參考資料中
- DeleteSecurityGroup在Amazon EC2 API 參考資料中
- AuthorizeSecurityGroupIngress在Amazon EC2 API 參考資料中

IAM 範例使用AWS SDK for Java

本節提供編寫程式的範例IAM通過使用AWS SDK for Java。

AWS Identity and Access Management(IAM) 可讓您安全地控制AWS服務和資源。您可以透過 IAM 建立及管理AWS使用者和組，並使用各種許可來允許和拒絕其存取AWS的費用。如需 IAM 的完整指南，請參觀IAM使用者指南。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 管理 IAM 存取金鑰
- 管理 IAM 使用者
- 使用 IAM 帳戶別名
- 處理 IAM 政策
- 處理 IAM 伺服器憑證

管理 IAM 存取金鑰

建立存取金鑰

要創建 IAM 訪問密鑰，請調用卓越亞馬遜身份管理客户端`createAccessKey`方法使用CreateAccessKeyRequest物件。

`CreateAccessKeyRequest`有兩個構造函數-一個採用用户名，另一個沒有參數。如果您使用的版本不接受任何參數，則必須使用`withUserNamesetter` 方法，再將它傳遞給`createAccessKey`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccessKeyRequest request = new CreateAccessKeyRequest()
    .withUserName(user);

CreateAccessKeyResult response = iam.createAccessKey(request);
```

請參閱 GitHub 上的完整範例。

列出存取金鑰

若要列出特定使用者的存取金鑰，請建立ListAccessKeysRequest物件，其中包含要列出其金鑰的使用者名稱，然後將該物件傳遞給卓越亞馬遜物件管理客户端的`listAccessKeys`方法。

> **Note**
>
> 如果您沒有將用户名提供給`listAccessKeys`，它將嘗試列出與AWS 帳戶，簽署了請求。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AccessKeyMetadata;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysRequest;
import com.amazonaws.services.identitymanagement.model.ListAccessKeysResult;
```

Code

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

boolean done = false;
ListAccessKeysRequest request = new ListAccessKeysRequest()
        .withUserName(username);

while (!done) {

    ListAccessKeysResult response = iam.listAccessKeys(request);

    for (AccessKeyMetadata metadata :
            response.getAccessKeyMetadata()) {
        System.out.format("Retrieved access key %s",
                metadata.getAccessKeyId());
    }

    request.setMarker(response.getMarker());

    if (!response.getIsTruncated()) {
        done = true;
    }
}
```

listAccessKeys 的結果會分頁 (每個呼叫預設最多 100 個記錄)。您可以致電getIsTruncated在傳回的ListAccessKeysResult物件，查看查詢傳回的結果是否少於可用結果。如果是這樣，那麼調用setMarker在ListAccessKeysRequest並將它傳回下次調用listAccessKeys。

請參閱 GitHub 上的完整範例。

擷取存取金鑰的上次使用時間

若要獲取上次使用存取金鑰的時間，請致電卓越亞馬遜物件管理客户端的getAccessKeyLastUsed方法與訪問密鑰的 ID (可以使用GetAccessKeyLastUsedRequest對象，或者直接到直接獲取訪問密鑰 ID 的重載。

然後，您可以使用返回的GetAccessKeyLastUsedResult物件，以檢索金鑰的上次使用。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedRequest;
```

```
import com.amazonaws.services.identitymanagement.model.GetAccessKeyLastUsedResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest()
    .withAccessKeyId(access_id);

GetAccessKeyLastUsedResult response = iam.getAccessKeyLastUsed(request);

System.out.println("Access key was last used at: " +
        response.getAccessKeyLastUsed().getLastUsedDate());
```

請參閱 GitHub 上的完整範例。

啟用或停用存取金鑰

您可以通過創建UpdateAccessKeyRequest對象，提供訪問密鑰 ID、可選的用户名和所需狀態，然後將請求物件傳遞給卓越亞馬遜物件管理客户端的updateAccessKey方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.UpdateAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateAccessKeyRequest request = new UpdateAccessKeyRequest()
    .withAccessKeyId(access_id)
    .withUserName(username)
    .withStatus(status);

UpdateAccessKeyResult response = iam.updateAccessKey(request);
```

請參閱 GitHub 上的完整範例。

刪除存取金鑰

要永久刪除訪問密鑰，請調用卓越亞馬遜身份管理客户端的deleteKey方法，為其提供DeleteAccessKeyRequest包含訪問密鑰的 ID 和用户名。

> **Note**
>
> 金鑰一旦刪除，就不能再擷取或使用。若要暫時停用金鑰，稍後再行啟動，請改為使用 updateAccessKey 方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccessKeyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccessKeyRequest request = new DeleteAccessKeyRequest()
    .withAccessKeyId(access_key)
    .withUserName(username);

DeleteAccessKeyResult response = iam.deleteAccessKey(request);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- CreateAccessKey在 IAM API 參考中的
- ListAccessKeys在 IAM API 參考中的
- GetAccessKeyLastUsed在 IAM API 參考中的
- UpdateAccessKey在 IAM API 參考中的
- DeleteAccessKey在 IAM API 參考中的

管理 IAM 使用者

建立使用者

直接提供使用者名稱或使用包含使用AmazonIdentityManagementClient者`createUser`名稱的CreateUserRequest物件，以建立新的 IAM 使用者。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateUserRequest;
import com.amazonaws.services.identitymanagement.model.CreateUserResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateUserRequest request = new CreateUserRequest()
    .withUserName(username);

CreateUserResult response = iam.createUser(request);
```

請參閱（詳見）的完整實例GitHub。

列出使用者

若要列出您帳戶的 IAM 使用者，請建立新的使用者ListUsersRequest並將其傳遞至AmazonIdentityManagementClient的`listUsers`方法。您可以透過呼叫`getUsers`傳回的ListUsersResult物件來擷取使用者清單。

`listUsers` 傳回的使用者清單會分頁。您可以呼叫回應物件的 `getIsTruncated` 方法，檢查是否有更多可擷取的結果。如果返回`true`，則調用請求對象的`setMarker()`方法，將響應對象方`getMarker()`法的返回值傳遞給它。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListUsersRequest;
import com.amazonaws.services.identitymanagement.model.ListUsersResult;
```

```
import com.amazonaws.services.identitymanagement.model.User;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

boolean done = false;
ListUsersRequest request = new ListUsersRequest();

while(!done) {
    ListUsersResult response = iam.listUsers(request);

    for(User user : response.getUsers()) {
        System.out.format("Retrieved user %s", user.getUserName());
    }

    request.setMarker(response.getMarker());

    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

請參閱（詳見）的完整實例GitHub。

更新使用者

若要更新使用者，請呼叫AmazonIdentityManagementClient物件的updateUser方法，該方法會取得可用來變更使用者名稱或路徑的UpdateUserRequest物件。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateUserRequest;
import com.amazonaws.services.identitymanagement.model.UpdateUserResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();
```

```
UpdateUserRequest request = new UpdateUserRequest()
    .withUserName(cur_name)
    .withNewUserName(new_name);

UpdateUserResult response = iam.updateUser(request);
```

請參閱（詳見）的完整實例GitHub。

刪除使用者

若要刪除使用者，deleteUser請使用要刪除之使用者名稱AmazonIdentityManagementClient的UpdateUserRequest物件集來呼叫要刪除的要求。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteConflictException;
import com.amazonaws.services.identitymanagement.model.DeleteUserRequest;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteUserRequest request = new DeleteUserRequest()
    .withUserName(username);

try {
    iam.deleteUser(request);
} catch (DeleteConflictException e) {
    System.out.println("Unable to delete user. Verify user is not" +
            " associated with any resources");
    throw e;
}
```

請參閱（詳見）的完整實例GitHub。

詳細資訊

- 使用IAM者指南中的 IAM 使用者

- 在使用IAM者指南中管理 IAM 使用者
- CreateUser在 IAM API 參考資料中
- ListUsers在 IAM API 參考資料中
- UpdateUser在 IAM API 參考資料中
- DeleteUser在 IAM API 參考資料中

使用 IAM 帳戶別名

若希望您的登入頁面的 URL 含有您的公司名稱或其他好記的識別符，而非您的AWS 帳戶 ID，則可建立您的別名AWS 帳戶。

> **Note**
>
> AWS每個帳戶僅支援一個帳戶別名。

建立帳戶別名

若要建立帳戶別名，請使用包含別名名稱的CreateAccountAliasRequest物件呼叫AmazonIdentityManagementClient的`createAccountAlias`方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.CreateAccountAliasResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

CreateAccountAliasRequest request = new CreateAccountAliasRequest()
    .withAccountAlias(alias);

CreateAccountAliasResult response = iam.createAccountAlias(request);
```

請參閱（詳見）的完整實例GitHub。

列出帳戶別名

若要列出帳戶的別名 (如果有)，請呼叫 AmazonIdentityManagementClient 的 `listAccountAliases` 方法。

> **Note**
>
> 傳回的ListAccountAliasesResult支援`getIsTruncated`和`getMarker`方法與其他AWS SDK for Java清單方法相同，但只AWS 帳戶能有一個帳戶別名。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAccountAliasesResult;
```

code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAccountAliasesResult response = iam.listAccountAliases();

for (String alias : response.getAccountAliases()) {
    System.out.printf("Retrieved account alias %s", alias);
}
```

請參閱（詳見）的完整實例GitHub。

刪除帳戶別名

若要刪除帳戶的別名 (如果有)，請呼叫 AmazonIdentityManagementClient 的 `deleteAccountAlias` 方法。刪除帳戶別名時，您必須使用DeleteAccountAliasRequest物件提供其名稱。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
```

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasRequest;
import com.amazonaws.services.identitymanagement.model.DeleteAccountAliasResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteAccountAliasRequest request = new DeleteAccountAliasRequest()
    .withAccountAlias(alias);

DeleteAccountAliasResult response = iam.deleteAccountAlias(request);
```

請參閱（詳見）的完整實例GitHub。

詳細資訊

- 用AWS戶指南中的帳戶 ID 及其別名IAM
- CreateAccountAlias在 IAM API 參考資料中
- ListAccountAliases在 IAM API 參考資料中
- DeleteAccountAlias在 IAM API 參考資料中

處理 IAM 政策

建立政策

若要建立新的政策，請在CreatePolicyRequest添加到亞馬遜身份管理客户的createPolicy方法。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreatePolicyRequest;
import com.amazonaws.services.identitymanagement.model.CreatePolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
```

```
    AmazonIdentityManagementClientBuilder.defaultClient();

CreatePolicyRequest request = new CreatePolicyRequest()
    .withPolicyName(policy_name)
    .withPolicyDocument(POLICY_DOCUMENT);

CreatePolicyResult response = iam.createPolicy(request);
```

IAM 政策文件是 JSON 字串，其中有記錄良好的語法。以下範例提供存取權以便對 DynamoDB 提出特定請求。

```
public static final String POLICY_DOCUMENT =
    "{" +
    "  \"Version\": \"2012-10-17\"," +
    "  \"Statement\": [" +
    "    {" +
    "        \"Effect\": \"Allow\"," +
    "        \"Action\": \"logs:CreateLogGroup\"," +
    "        \"Resource\": \"%s\"" +
    "    }," +
    "    {" +
    "        \"Effect\": \"Allow\"," +
    "        \"Action\": [" +
    "            \"dynamodb:DeleteItem\"," +
    "            \"dynamodb:GetItem\"," +
    "            \"dynamodb:PutItem\"," +
    "            \"dynamodb:Scan\"," +
    "            \"dynamodb:UpdateItem\"" +
    "       ]," +
    "       \"Resource\": \"RESOURCE_ARN\"" +
    "    }" +
    "   ]" +
    "}";
```

請參閱 GitHub 上的完整範例。

取得政策

要檢索現有政策，請致電卓越亞馬遜身份管理客户端的`getPolicy`方法，提供政策的 ARN。GetPolicyRequest物件。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetPolicyRequest;
import com.amazonaws.services.identitymanagement.model.GetPolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetPolicyRequest request = new GetPolicyRequest()
    .withPolicyArn(policy_arn);

GetPolicyResult response = iam.getPolicy(request);
```

請參閱 GitHub 上的完整範例。

附加角色政策

您可以通過調用卓越亞馬遜身份管理客户端的 http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html[role`attachRolePolicy`方法 ARN 在AttachRolePolicyRequest。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.AttachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.AttachedPolicy;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

AttachRolePolicyRequest attach_request =
    new AttachRolePolicyRequest()
        .withRoleName(role_name)
        .withPolicyArn(POLICY_ARN);

iam.attachRolePolicy(attach_request);
```

請參閱 GitHub 上的完整範例。

列出附加的角色政策

透過呼叫卓越亞馬遜身份管理，列出角色上的連接政策。listAttachedRolePolicies方法。它採用 ListAttachedRolePoliciesRequest 物件，其中包含要列出其政策的角色名稱。

CallgetAttachedPolicies在返回的頁 ListAttachedRolePoliciesResult物件，以取得連接政策的列表。結果可能會被截斷；如果ListAttachedRolePoliciesResult物件的getIsTruncated方法返回true，請呼叫ListAttachedRolePoliciesRequest物件的setMarker方法並使用它來調用listAttachedRolePolicies，以取得下一批次的結果。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesRequest;
import com.amazonaws.services.identitymanagement.model.ListAttachedRolePoliciesResult;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

ListAttachedRolePoliciesRequest request =
    new ListAttachedRolePoliciesRequest()
        .withRoleName(role_name);

List<AttachedPolicy> matching_policies = new ArrayList<>();

boolean done = false;

while(!done) {
    ListAttachedRolePoliciesResult response =
        iam.listAttachedRolePolicies(request);

    matching_policies.addAll(
            response.getAttachedPolicies()
                    .stream()
```

```
                    .filter(p -> p.getPolicyName().equals(role_name))
                    .collect(Collectors.toList()));

    if(!response.getIsTruncated()) {
        done = true;
    }
    request.setMarker(response.getMarker());
}
```

請參閱 GitHub 上的完整範例。

分離角色政策

若要將政策與角色分離，請呼叫卓越亞馬遜身份管理客戶端的`detachRolePolicy`方法 ARN 在DetachRolePolicyRequest。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyRequest;
import com.amazonaws.services.identitymanagement.model.DetachRolePolicyResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DetachRolePolicyRequest request = new DetachRolePolicyRequest()
    .withRoleName(role_name)
    .withPolicyArn(policy_arn);

DetachRolePolicyResult response = iam.detachRolePolicy(request);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- IAM 政策概觀中的IAM使用者指南。
- AWSIAM 策略參考中的IAM使用者指南。

- CreatePolicy在 IAM API 參考中
- GetPolicy在 IAM API 參考中
- AttachRolePolicy在 IAM API 參考中
- ListAttachedRolePolicies在 IAM API 參考中
- DetachRolePolicy在 IAM API 參考中

處理 IAM 伺服器憑證

若要啟用 HTTPS 連線，連線到您的網站或應用程式AWS，您需要 SSL/TLS伺服器憑證。您可以使用由AWSCertificate Manager 或您從外部提供程式獲得的憑證管理器。

建議您使用 ACM 來佈建、管理和部署您的伺服器憑證。使用 ACM，您可以請求憑證，並將其部署到您的AWS資源，並讓 ACM 為您處理憑證續約。ACM 提供的憑證是免費的。如需 ACM 的詳細資訊，請參見ACM 使用者指南。

取得伺服器憑證

您可透過呼叫卓越亞馬遜身份管理客戶端的`getServerCertificate`方法，將其傳遞一個GetServerCertificateRequest與證書的名稱。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.GetServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

GetServerCertificateRequest request = new GetServerCertificateRequest()
            .withServerCertificateName(cert_name);

GetServerCertificateResult response = iam.getServerCertificate(request);
```

請參閱 GitHub 上的完整範例。

列出伺服器憑證

若要列出您的伺服器憑證，請呼叫卓越亞馬遜身份管理客户端的listServerCertificates方法，並搭配ListServerCertificatesRequest。其會傳回ListServerCertificatesResult。

呼叫傳回 ListServerCertificateResult 物件的 getServerCertificateMetadataList 方法以取得 ServerCertificateMetadata 物件的清單，您可以用來取得每個憑證的相關資訊。

結果可能會被截斷；如果ListServerCertificateResult物件的getIsTruncated方法返回true中，呼叫ListServerCertificatesRequest物件的setMarker方法並使用它來調用listServerCertificates，以獲得下一批次的結果。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesRequest;
import com.amazonaws.services.identitymanagement.model.ListServerCertificatesResult;
import com.amazonaws.services.identitymanagement.model.ServerCertificateMetadata;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

boolean done = false;
ListServerCertificatesRequest request =
        new ListServerCertificatesRequest();

while(!done) {

    ListServerCertificatesResult response =
        iam.listServerCertificates(request);

    for(ServerCertificateMetadata metadata :
            response.getServerCertificateMetadataList()) {
        System.out.printf("Retrieved server certificate %s",
                metadata.getServerCertificateName());
    }

    request.setMarker(response.getMarker());
```

```
    if(!response.getIsTruncated()) {
        done = true;
    }
}
```

請參閱 GitHub 上的完整範例。

更新伺服器憑證

您可透過呼叫卓越亞馬遜身份管理客戶端的updateServerCertificate方法。它需要設定 UpdateServerCertificateRequest 物件，並搭配伺服器憑證的目前名稱以及要使用的新名稱或新路徑。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.UpdateServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

UpdateServerCertificateRequest request =
    new UpdateServerCertificateRequest()
        .withServerCertificateName(cur_name)
        .withNewServerCertificateName(new_name);

UpdateServerCertificateResult response =
    iam.updateServerCertificate(request);
```

請參閱 GitHub 上的完整範例。

刪除伺服器憑證

若要刪除伺服器憑證，請呼叫卓越亞馬遜身份管理客戶端的deleteServerCertificate方法，並搭配DeleteServerCertificateRequest包含證書名稱。

匯入

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
```

```
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateRequest;
import com.amazonaws.services.identitymanagement.model.DeleteServerCertificateResult;
```

Code

```
final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

DeleteServerCertificateRequest request =
    new DeleteServerCertificateRequest()
        .withServerCertificateName(cert_name);

DeleteServerCertificateResult response =
    iam.deleteServerCertificate(request);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 使用伺服器憑證中的IAM使用者指南
- GetServerCertificate在 IAM API 參考。
- ListServerCertificates在 IAM API 參考。
- UpdateServerCertificate在 IAM API 參考。
- DeleteServerCertificate在 IAM API 參考。
- ACM 使用者指南

使用 AWS SDK for Java 的 Lambda 範例

本節提供使用AWS SDK for Java編寫 Lambda 程式的範例。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 調用、列出和刪除Lambda函數

調用、列出和刪除Lambda函數

本節提供使用Lambda服務客户端使用AWS SDK for Java。若要了解如何建立Lambda函數，請參如何建立AWS Lambda功能。

主題

- 呼叫 函數
- 列出 函數
- 刪除 函數

呼叫 函數

您可以透過Lambda函數通過創建AWSLambda對象並調用其invoke方法。建立 InvokeRequest 物件以指定其他資訊，例如函數名稱和要傳遞至 Lambda 函數的承載。函數名稱顯示為arn: aws: lambda: us-東#-1:55555556330391: HelloFunction。您可以透過查看AWS Management Console。

若要將有效載荷傳遞給函數，請調用InvokeRequest物件的withPayload方法，並指定 JSON 格式的 String，如下列代碼範例所示。

匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.amazonaws.services.lambda.model.ServiceException;

import java.nio.charset.StandardCharsets;
```

Code

以下程式碼範例示範如何呼叫 Lambda 函數。

```
        String functionName = args[0];
```

```
        InvokeRequest invokeRequest = new InvokeRequest()
                .withFunctionName(functionName)
                .withPayload("{\n" +
                        " \"Hello \": \"Paris\",\n" +
                        " \"countryCode\": \"FR\"\n" +
                        "}");
        InvokeResult invokeResult = null;

        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            invokeResult = awsLambda.invoke(invokeRequest);

            String ans = new String(invokeResult.getPayload().array(),
StandardCharsets.UTF_8);

            //write out the return value
            System.out.println(ans);

        } catch (ServiceException e) {
            System.out.println(e);
        }

        System.out.println(invokeResult.getStatusCode());
```

請參閱 GitHub 上的完整範例。

列出 函數

建立AWSLambda對象並調用其listFunctions方法。此方法會傳回ListFunctionsResult物件。您可以叫用此物件的 getFunctions 方法來傳回 FunctionConfiguration 物件的清單。您可以逐一查看清單以擷取函數的相關資訊。例如，下列 Java 程式碼範例示範如何取得每個函數名稱。

匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.FunctionConfiguration;
import com.amazonaws.services.lambda.model.ListFunctionsResult;
```

```
import com.amazonaws.services.lambda.model.ServiceException;
import java.util.Iterator;
import java.util.List;
```

Code

下列程式碼範例示範如何在Lambda函數名稱。

```
        ListFunctionsResult functionResult = null;

        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            functionResult = awsLambda.listFunctions();

            List<FunctionConfiguration> list = functionResult.getFunctions();

            for (Iterator iter = list.iterator(); iter.hasNext(); ) {
                FunctionConfiguration config = (FunctionConfiguration)iter.next();

                System.out.println("The function name is "+config.getFunctionName());
            }

        } catch (ServiceException e) {
            System.out.println(e);
        }
```

請參閱 GitHub 上的完整範例。

刪除 函數

建立AWSLambda對象並調用其deleteFunction方法。建立一個 DeleteFunctionRequest 物件並將其傳遞給 deleteFunction 方法。此物件包含資訊，例如要刪除的函數名稱。函數名稱顯示為arn: aws: lambda: us-東#-1:55555556330391: HelloFunction。您可以透過查看AWS Management Console。

匯入

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
```

```
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.ServiceException;
import com.amazonaws.services.lambda.model.DeleteFunctionRequest;
```

Code

下列程式碼範例示範如何刪除Lambda函數。

```
        String functionName = args[0];
        try {
            AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                    .withCredentials(new ProfileCredentialsProvider())
                    .withRegion(Regions.US_WEST_2).build();

            DeleteFunctionRequest delFunc = new DeleteFunctionRequest();
            delFunc.withFunctionName(functionName);

            //Delete the function
            awsLambda.deleteFunction(delFunc);
            System.out.println("The function is deleted");

        } catch (ServiceException e) {
            System.out.println(e);
        }
```

請參閱 GitHub 上的完整範例。

使用 AWS SDK for Java 的 Amazon Pinpoint 範例

本節提供使用AWS SDK for Java編寫 Amazon Pinpoint 程式的範例。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 在中創建和刪除應用Amazon Pinpoint

- 建立適用於的端點Amazon Pinpoint
- 在中建立客羣Amazon Pinpoint
- 建立行銷活動Amazon Pinpoint
- 更新頻道Amazon Pinpoint

在中創建和刪除應用Amazon Pinpoint

應用程序是Amazon Pinpoint項目，您可以在該項目中為不同應用程序定義受眾，並通過定製的消息吸引這些受眾。此頁面上的示例演示瞭如何創建新應用或刪除現有應用程序。

建立應用程式

創建新應用程序Amazon Pinpoint通過將應用程序名稱提供給CreateAppRequest對象，然後將該對象傳遞給卓越亞馬遜點客户端的`createApp`方法。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateAppRequest;
import com.amazonaws.services.pinpoint.model.CreateAppResult;
import com.amazonaws.services.pinpoint.model.CreateApplicationRequest;
```

Code

```
CreateApplicationRequest appRequest = new CreateApplicationRequest()
  .withName(appName);

CreateAppRequest request = new CreateAppRequest();
request.withCreateApplicationRequest(appRequest);
CreateAppResult result = pinpoint.createApp(request);
```

請參閱 GitHub 上的完整範例。

刪除應用程式

要刪除應用程序，請調用卓越亞馬遜點客户端的`deleteApp`用於DeleteAppRequest對象，該對象設置為要刪除的應用程序名稱。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
```

Code

```
DeleteAppRequest deleteRequest = new DeleteAppRequest()
  .withApplicationId(appID);

pinpoint.deleteApp(deleteRequest);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 應用程式中的Amazon PinpointAPI 參考
- 應用程式中的Amazon PinpointAPI 參考

建立適用於的端點Amazon Pinpoint

端點會唯一識別您可以使用Amazon Pinpoint。如果您的應用程序啟用了Amazon Pinpoint支持，您的應用程序會自動將終端節點註冊到Amazon Pinpoint，當新用户開啟您的應用程式時。以下範例示範如何以編程方式新增新端點。

建立端點

建立新的端點Amazon Pinpoint通過提供端點數據在EndpointRequest物件。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.UpdateEndpointRequest;
import com.amazonaws.services.pinpoint.model.UpdateEndpointResult;
import com.amazonaws.services.pinpoint.model.EndpointDemographic;
import com.amazonaws.services.pinpoint.model.EndpointLocation;
import com.amazonaws.services.pinpoint.model.EndpointRequest;
import com.amazonaws.services.pinpoint.model.EndpointResponse;
import com.amazonaws.services.pinpoint.model.EndpointUser;
import com.amazonaws.services.pinpoint.model.GetEndpointRequest;
import com.amazonaws.services.pinpoint.model.GetEndpointResult;
```

Code

```
HashMap<String, List<String>> customAttributes = new HashMap<>();
List<String> favoriteTeams = new ArrayList<>();
favoriteTeams.add("Lakers");
favoriteTeams.add("Warriors");
customAttributes.put("team", favoriteTeams);


EndpointDemographic demographic = new EndpointDemographic()
        .withAppVersion("1.0")
        .withMake("apple")
        .withModel("iPhone")
        .withModelVersion("7")
        .withPlatform("ios")
        .withPlatformVersion("10.1.1")
        .withTimezone("America/Los_Angeles");

EndpointLocation location = new EndpointLocation()
        .withCity("Los Angeles")
        .withCountry("US")
        .withLatitude(34.0)
        .withLongitude(-118.2)
        .withPostalCode("90068")
        .withRegion("CA");

Map<String,Double> metrics = new HashMap<>();
metrics.put("health", 100.00);
metrics.put("luck", 75.00);

EndpointUser user = new EndpointUser()
        .withUserId(UUID.randomUUID().toString());

DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm'Z'"); // Quoted "Z" to
 indicate UTC, no timezone offset
String nowAsISO = df.format(new Date());

EndpointRequest endpointRequest = new EndpointRequest()
        .withAddress(UUID.randomUUID().toString())
        .withAttributes(customAttributes)
        .withChannelType("APNS")
        .withDemographic(demographic)
        .withEffectiveDate(nowAsISO)
        .withLocation(location)
```

```
        .withMetrics(metrics)
        .withOptOut("NONE")
        .withRequestId(UUID.randomUUID().toString())
        .withUser(user);
```

然後，建立UpdateEndpointRequest物件 EndpointRequest 物件。最後，請傳遞 UpdateEndpointRequest 對象添加到卓越亞馬遜品點客户端的updateEndpoint方法。

Code

```
UpdateEndpointRequest updateEndpointRequest = new UpdateEndpointRequest()
        .withApplicationId(appId)
        .withEndpointId(endpointId)
        .withEndpointRequest(endpointRequest);

UpdateEndpointResult updateEndpointResponse =
 client.updateEndpoint(updateEndpointRequest);
System.out.println("Update Endpoint Response: " +
 updateEndpointResponse.getMessageBody());
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 新增端點中的Amazon Pinpoint開發人員指南
- 端點中的Amazon PinpointAPI 參考

在中建立客羣Amazon Pinpoint

使用者客羣代表基於共用特性的一部分使用者，例如使用者最近打開您的應用程式的情形，或他們使用何種裝置。以下範例示範如何定義客羣。

建立客群

創建一個新的區段Amazon Pinpoint通過定義線段的維度SegmentDimensions物件。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
```

```
import com.amazonaws.services.pinpoint.model.CreateSegmentRequest;
import com.amazonaws.services.pinpoint.model.CreateSegmentResult;
import com.amazonaws.services.pinpoint.model.AttributeDimension;
import com.amazonaws.services.pinpoint.model.AttributeType;
import com.amazonaws.services.pinpoint.model.RecencyDimension;
import com.amazonaws.services.pinpoint.model.SegmentBehaviors;
import com.amazonaws.services.pinpoint.model.SegmentDemographics;
import com.amazonaws.services.pinpoint.model.SegmentDimensions;
import com.amazonaws.services.pinpoint.model.SegmentLocation;
import com.amazonaws.services.pinpoint.model.SegmentResponse;
import com.amazonaws.services.pinpoint.model.WriteSegmentRequest;
```

Code

```
Pinpoint pinpoint =
 AmazonPinpointClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
Map<String, AttributeDimension> segmentAttributes = new HashMap<>();
segmentAttributes.put("Team", new
 AttributeDimension().withAttributeType(AttributeType.INCLUSIVE).withValues("Lakers"));

SegmentBehaviors segmentBehaviors = new SegmentBehaviors();
SegmentDemographics segmentDemographics = new SegmentDemographics();
SegmentLocation segmentLocation = new SegmentLocation();

RecencyDimension recencyDimension = new RecencyDimension();
recencyDimension.withDuration("DAY_30").withRecencyType("ACTIVE");
segmentBehaviors.setRecency(recencyDimension);

SegmentDimensions dimensions = new SegmentDimensions()
        .withAttributes(segmentAttributes)
        .withBehavior(segmentBehaviors)
        .withDemographic(segmentDemographics)
        .withLocation(segmentLocation);
```

接下來設置SegmentDimensions物件的WriteSegmentRequest，這反過來用於創建一個CreateSegmentRequest物件。然後傳遞 CreateSegmentRequest 對象添加到卓越亞馬遜品額客户端的`createSegment`方法。

Code

```
WriteSegmentRequest writeSegmentRequest = new WriteSegmentRequest()
        .withName("MySegment").withDimensions(dimensions);
```

```
CreateSegmentRequest createSegmentRequest = new CreateSegmentRequest()
        .withApplicationId(appId).withWriteSegmentRequest(writeSegmentRequest);

CreateSegmentResult createSegmentResult = client.createSegment(createSegmentRequest);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- Amazon Pinpoint區段中的Amazon Pinpoint使用者指南
- 建立客群中的Amazon Pinpoint開發人員指南
- 區段中的Amazon PinpointAPI 參考
- 區段中的Amazon PinpointAPI 參考

建立行銷活動Amazon Pinpoint

您可以使用廣告活動來協助提高應用程式與使用者之間的互動。您可以創建一個廣告系列，以通過量身定制的消息或特殊促銷活動與用戶的特定細分接觸。此範例示範如何建立新的標準行銷活動，將自訂推播通知傳送至指定區段。

建立行銷活動

在建立新的行銷活動之前，您必須定義「排程」和「訊息」，並在WriteCampaignRequest物件中設定這些值。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.CreateCampaignRequest;
import com.amazonaws.services.pinpoint.model.CreateCampaignResult;
import com.amazonaws.services.pinpoint.model.Action;
import com.amazonaws.services.pinpoint.model.CampaignResponse;
import com.amazonaws.services.pinpoint.model.Message;
import com.amazonaws.services.pinpoint.model.MessageConfiguration;
import com.amazonaws.services.pinpoint.model.Schedule;
import com.amazonaws.services.pinpoint.model.WriteCampaignRequest;
```

Code

```
Schedule schedule = new Schedule()
        .withStartTime("IMMEDIATE");

Message defaultMessage = new Message()
        .withAction(Action.OPEN_APP)
        .withBody("My message body.")
        .withTitle("My message title.");

MessageConfiguration messageConfiguration = new MessageConfiguration()
        .withDefaultMessage(defaultMessage);

WriteCampaignRequest request = new WriteCampaignRequest()
        .withDescription("My description.")
        .withSchedule(schedule)
        .withSegmentId(segmentId)
        .withName("MyCampaign")
        .withMessageConfiguration(messageConfiguration);
```

然後在中建立新的Amazon Pinpoint行銷活動，方WriteCampaignRequest法是為CreateCampaignRequest物件提供促銷活動設定。最後，將對CreateCampaignRequest象傳遞給AmazonPinpointClient的`createCampaign`方法。

Code

```
CreateCampaignRequest createCampaignRequest = new CreateCampaignRequest()
        .withApplicationId(appId).withWriteCampaignRequest(request);

CreateCampaignResult result = client.createCampaign(createCampaignRequest);
```

請參閱（詳見）的完整實例GitHub。

詳細資訊

- Amazon PinpointAmazon Pinpoint使用者指南中的促銷活動
- 在Amazon Pinpoint開發者指南中建立行銷活動
- Amazon PinpointAPI 參考資料中的促銷活動
- Amazon PinpointAPI 參考資料中的促銷活動
- Amazon PinpointAPI 參考資料中的促銷活動
- Amazon PinpointAPI 參考資料中的行銷活動版本

- Amazon PinpointAPI 參考資料中的行銷活動版本

更新頻道Amazon Pinpoint

通道定義您可以向其傳遞消息的平台類型。此範例說明如何使用 APN 頻道傳送訊息。

更新頻道

在中啟用一個通道Amazon Pinpoint通過提供您想要更新的頻道類型的應用 ID 和請求對象。此示例更新 APN 通道，該通道需要APNS 通道請求物件。將這些設置在UpdateApnsChannelRequest並將該對象傳遞給卓越亞馬遜點客户端的updateApnsChannel方法。

匯入

```
import com.amazonaws.services.pinpoint.AmazonPinpoint;
import com.amazonaws.services.pinpoint.AmazonPinpointClientBuilder;
import com.amazonaws.services.pinpoint.model.APNSChannelRequest;
import com.amazonaws.services.pinpoint.model.APNSChannelResponse;
import com.amazonaws.services.pinpoint.model.GetApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.GetApnsChannelResult;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelRequest;
import com.amazonaws.services.pinpoint.model.UpdateApnsChannelResult;
```

Code

```
APNSChannelRequest request = new APNSChannelRequest()
  .withEnabled(enabled);

UpdateApnsChannelRequest updateRequest = new UpdateApnsChannelRequest()
  .withAPNSChannelRequest(request)
  .withApplicationId(appId);
UpdateApnsChannelResult result = client.updateApnsChannel(updateRequest);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- Amazon Pinpoint頻道中的Amazon Pinpoint使用者指南
- ADM 頻道中的Amazon PinpointAPI 參考

- APN 頻道中的Amazon PinpointAPI 參考
- APN 沙箱頻道中的Amazon PinpointAPI 參考
- VoIP 通道中的Amazon PinpointAPI 參考
- APN VoIP 沙箱通道中的Amazon PinpointAPI 參考
- 百度頻道中的Amazon PinpointAPI 參考
- 電子郵件管道中的Amazon PinpointAPI 參考
- GCM 頻道中的Amazon PinpointAPI 參考
- 簡訊管道中的Amazon PinpointAPI 參考

使用 AWS SDK for Java 的 Amazon S3 範例

本節提供使用AWS SDK for Java編寫 Amazon S3 程式的範例。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 創建、列出和刪除Amazon S3儲存貯體
- 對物件執行Amazon S3作業
- 管理Amazon S3存儲桶和對象的訪問權限
- 管理對 的存取Amazon S3使用儲存儲體政策
- 使用 TransferManager 為了Amazon S3操作
- 設定Amazon S3存儲桶即網站
- 使用Amazon S3用戶端加密

創建、列出和刪除Amazon S3儲存貯體

中的每個物件 (檔案)Amazon S3必須位於桶，它代表物件集合（容器）。每個存儲桶都由鍵（名稱），必須是唯一的。如需儲存貯體及其配置的詳細資訊，請參使用Amazon S3儲存貯體中的Amazon Simple Storage Service使用者指南。

> **Note**
>
> 最佳實務
> 我們建議您在 儲存貯體上啟用 AbortIncompleteMultipartUploadAmazon S3 生命週期規則。此規則指示 Amazon S3 中止啟動後未在指定天數內完成的分段上傳。超過設定的時間限制時，Amazon S3 會中止上傳，然後刪除未完成的上傳資料。
> 如需詳細資訊，請參閱「」具版本控制的儲存貯體的生命週期組態中的Amazon S3使用者指南。

> **Note**
>
> 這些代碼示例假定您理解使用AWS SDK for Java並配置了默認AWS憑據使用設定AWS全權證書和區域促進發展。

建立儲存貯體

使用卓越亞馬遜客户端的`createBucket`方法。新儲存貯體傳回。所以此`createBucket`方法將引發異常，如果存儲桶已經存在。

> **Note**
>
> 要在嘗試創建具有相同名稱的存儲桶之前檢查存儲桶是否已存在，請調用`doesBucketExist`方法。它會傳回`true`（如果存儲桶存在），`false`否則為。

匯入

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;
```

Code

```
if (s3.doesBucketExistV2(bucket_name)) {
    System.out.format("Bucket %s already exists.\n", bucket_name);
    b = getBucket(bucket_name);
} else {
    try {
        b = s3.createBucket(bucket_name);
    } catch (AmazonS3Exception e) {
        System.err.println(e.getErrorMessage());
    }
}
return b;
```

請參閱 GitHub 上的完整範例。

列出儲存貯體

使用卓越亞馬遜客户端的`listBucket`方法。如果成功，則會傳回儲存貯體傳回。

匯入

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.Bucket;

import java.util.List;
```

Code

```
List<Bucket> buckets = s3.listBuckets();
System.out.println("Your {S3} buckets are:");
for (Bucket b : buckets) {
    System.out.println("* " + b.getName());
}
```

請參閱 GitHub 上的完整範例。

刪除儲存貯體

刪除Amazon S3儲存貯體之後，您必須先確保儲存貯體是空的，否則會傳回錯誤。如果您有版本化存儲桶，則必須刪除與儲存貯體相關聯的任何版本控制物件。

> **Note**
>
> 所以此完整範例按順序包含這些步驟中的每個步驟，提供了一個完整的解決方案來刪除Amazon S3存儲桶及其內容。

主題

- 從未版本控制的存儲桶中刪除對象，然後再刪除對象
- 刪除版本控制的儲存貯體之前，請先刪除儲存貯體
- 刪除空的儲存貯體

從未版本控制的存儲桶中刪除對象，然後再刪除對象

使用卓越亞馬遜客户端的`listObjects`方法以檢索物件列表和`deleteObject`以刪除每一個。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" - removing objects from bucket");
ObjectListing object_listing = s3.listObjects(bucket_name);
while (true) {
    for (Iterator<?> iterator =
         object_listing.getObjectSummaries().iterator();
         iterator.hasNext(); ) {
        S3ObjectSummary summary = (S3ObjectSummary) iterator.next();
        s3.deleteObject(bucket_name, summary.getKey());
    }

    // more object_listing to retrieve?
    if (object_listing.isTruncated()) {
        object_listing = s3.listNextBatchOfObjects(object_listing);
```

```
    } else {
        break;
    }
}
```

請參閱 GitHub 上的完整範例。

刪除版本控制的儲存貯體之前，請先刪除儲存貯體

如果您使用版本化存儲桶，您還需要刪除存儲桶中存儲的對象的所有版本，然後才能刪除存儲桶。

使用類似於在存儲桶中刪除數據元時使用的模式，使用 AmazonS3 客户端的`listVersions`方法列出任何版本控制對象，然後`deleteVersion`以刪除每一個。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" - removing versions from bucket");
VersionListing version_listing = s3.listVersions(
        new ListVersionsRequest().withBucketName(bucket_name));
while (true) {
    for (Iterator<?> iterator =
         version_listing.getVersionSummaries().iterator();
         iterator.hasNext(); ) {
        S3VersionSummary vs = (S3VersionSummary) iterator.next();
        s3.deleteVersion(
                bucket_name, vs.getKey(), vs.getVersionId());
    }

    if (version_listing.isTruncated()) {
        version_listing = s3.listNextBatchOfVersions(
                version_listing);
    } else {
        break;
```

```
    }
}
```

請參閱 GitHub 上的完整範例。

刪除空的儲存貯體

從存儲桶中刪除數據元後（包括任何版本控制的對象），您可以使用 AmazonS3 客户端的`deleteBucket`方法。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;
```

Code

```
System.out.println(" OK, bucket ready to delete!");
s3.deleteBucket(bucket_name);
```

請參閱 GitHub 上的完整範例。

對物件執行Amazon S3作業

一個Amazon S3對象表示一個文件或數據的集合。每個物件都必須位於值區內。

> **Note**
>
> 這些程式碼範例假設您瞭解使用中的資料，AWS SDK for Java並且已使用設定認AWS證和開發區域中的資訊來設定預設AWS認證。

主題

- 上傳物件
- 列出物件

- 下載物件
- 複製、移動或重新命名物件
- 刪除物件
- 一次刪除多個物件

上傳物件

使用 AmazonS3 客戶端的`putObject`方法，提供存儲桶名稱，密鑰名稱和要上傳的文件。儲存貯體必須存在，否則會有錯誤結果。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
System.out.format("Uploading %s to S3 bucket %s...\n", file_path, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.putObject(bucket_name, key_name, new File(file_path));
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

列出物件

要獲取存儲桶中的對象列表，請使用 AmazonS3 客戶端的`listObjects`方法，提供存儲桶的名稱。

此方`listObjects`法會傳ObjectListing回提供值區中物件相關資訊的物件。若要列出物件名稱 (key)，請使用方`getObjectSummaries`法取得 S3 ObjectSummary 物件的清單，每個物件都代表儲存貯體中的單一物件。然後調用其`getKey`方法來檢索對象的名稱。

匯入

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsV2Result;
import com.amazonaws.services.s3.model.S3ObjectSummary;
```

Code

```
System.out.format("Objects in S3 bucket %s:\n", bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
ListObjectsV2Result result = s3.listObjectsV2(bucket_name);
List<S3ObjectSummary> objects = result.getObjectSummaries();
for (S3ObjectSummary os : objects) {
    System.out.println("* " + os.getKey());
}
```

請參閱（詳見）的完整實例GitHub。

下載物件

使用 AmazonS3 客戶端的getObject方法，將要下載的存儲桶和對象的名稱傳遞給它。如果成功，該方法返回一個 S3 對象。指定的儲存貯體和物件金鑰必須存在，否則會有錯誤結果。

您可以透過呼叫getObjectContent來取得物件的內容S3Object。這將返回一個 S3 ObjectInputStream，表現為一個標準的 Java InputStream 對象。

下列範例會從 S3 下載物件，並將其內容儲存至檔案 (使用與物件金鑰相同的名稱)。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

import java.io.File;
```

Code

```
System.out.format("Downloading %s from S3 bucket %s...\n", key_name, bucket_name);
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    S3Object o = s3.getObject(bucket_name, key_name);
    S3ObjectInputStream s3is = o.getObjectContent();
    FileOutputStream fos = new FileOutputStream(new File(key_name));
    byte[] read_buf = new byte[1024];
    int read_len = 0;
    while ((read_len = s3is.read(read_buf)) > 0) {
        fos.write(read_buf, 0, read_len);
    }
    s3is.close();
    fos.close();
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
} catch (FileNotFoundException e) {
    System.err.println(e.getMessage());
    System.exit(1);
} catch (IOException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

複製、移動或重新命名物件

您可以使用儲存貯體複製物件到另一個儲存貯體到另一個儲存貯體到另一個儲存貯體到另一個 `copyObject` 它會取得要複製來源的值區名稱、要複製的物件，以及目的地值區名稱。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

Code

```
try {
    s3.copyObject(from_bucket, object_key, to_bucket, object_key);
} catch (AmazonServiceException e) {
```

```
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Done!");
```

請參閱（詳見）的完整實例GitHub。

> **Note**
>
> 您可以搭copyObject配 deleteObject 來移動或重新命名物件，方法是先將物件複製到新名稱(您可以使用與來源和目的地相同的值區)，然後從舊位置刪除物件。

刪除物件

使用 AmazonS3 客戶端的deleteObject方法，將要刪除的存儲桶和對象的名稱傳遞給它。指定的儲存貯體和物件金鑰必須存在，否則會有錯誤結果。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteObject(bucket_name, object_key);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

一次刪除多個物件

使用亞馬遜 S3 客戶端的deleteObjects方法，您可以通過將多個對象的名稱傳遞給鏈接來刪除同一存儲桶中的多個對象：sdk-for-java/v1/ 引用/COM /Amazonaws/服務/s3/模型/DeleteObjectsRequest .html 方法。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    DeleteObjectsRequest dor = new DeleteObjectsRequest(bucket_name)
            .withKeys(object_keys);
    s3.deleteObjects(dor);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱（詳見）的完整實例GitHub。

管理Amazon S3存儲桶和對象的訪問權限

您可以將存取控制清單 (ACL) 用於Amazon S3桶和對象，用於精細控制您的Amazon S3的費用。

> **Note**
>
> 這些代碼示例假定您理解使用AWS SDK for Java並已配置默認AWS憑據使用設定AWSD. 資料和區域促進發展。

獲取存儲桶的訪問控制列表

要獲取存儲桶的當前 ACL，請調用卓越亞馬遜的`getBucketAcl`方法，將其傳遞給儲存貯體名稱查詢。這個方法會傳回AccessControlList物件。要獲取列表中的每個訪問授權，請調用其`getGrantsAsList`方法，它將返回一個標準的 Java 列表授予物件。

匯入

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Grant;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

設置存儲桶的訪問控制列表

要為存儲段的 ACL 添加或修改權限，請調用 AmazonS3 的`setBucketAcl`方法。它需要一個AccessControlList包含要設定的被授權者和存取級別的清單之物件。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

```
try {
    // get the current ACL
    AccessControlList acl = s3.getBucketAcl(bucket_name);
    // set access for the grantee
    EmailAddressGrantee grantee = new EmailAddressGrantee(email);
    Permission permission = Permission.valueOf(access);
    acl.grantPermission(grantee, permission);
    s3.setBucketAcl(bucket_name, acl);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

> **Note**
>
> 您可以直接使用承授者類，或者使用EmailAddressGrantee類通過電子郵件設置被授權者，就像我們在這裏所做的那樣。

請參閱 GitHub 上的完整範例。

獲取對象的訪問控制列表

要獲取對象的當前 ACL，請調用卓越亞馬遜的getObjectAcl方法，將其傳遞給儲存貯體名稱和物件名稱查詢。LIGEgetBucketAcl，則此方法返回一個AccessControlList對象，您可以使用它來檢查每個授予。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Grant;
```

Code

```
try {
    AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
```

```
    List<Grant> grants = acl.getGrantsAsList();
    for (Grant grant : grants) {
        System.out.format("  %s: %s\n", grant.getGrantee().getIdentifier(),
                grant.getPermission().toString());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

設置對象的訪問控制列表

要為某個對象的 ACL 添加或修改權限，請調用 AmazonS3 的`setObjectAcl`方法。它需要一個AccessControlList包含要設定的被授權者和存取級別的清單之物件。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
```

Code

```
    try {
        // get the current ACL
        AccessControlList acl = s3.getObjectAcl(bucket_name, object_key);
        // set access for the grantee
        EmailAddressGrantee grantee = new EmailAddressGrantee(email);
        Permission permission = Permission.valueOf(access);
        acl.grantPermission(grantee, permission);
        s3.setObjectAcl(bucket_name, object_key, acl);
    } catch (AmazonServiceException e) {
        System.err.println(e.getErrorMessage());
        System.exit(1);
    }
}
```

> **Note**
>
> 您可以直接使用承授者類，或者使用EmailAddressGrantee類通過電子郵件設置被授權者，就像我們在這裏所做的那樣。

請參閱 GitHub 上的完整範例。

詳細資訊

- GET 儲存貯體 acl中的Amazon S3API 參考
- PUT 儲存貯體 acl中的Amazon S3API 參考
- GET 物件 acl中的Amazon S3API 參考
- PUT 物件 acl中的Amazon S3API 參考

管理對 的存取Amazon S3使用儲存儲體政策

您可以設置、獲取或刪除儲存儲體政策來管理對Amazon S3儲存儲體。

設定儲存儲體政策

您可以通過下列方式為特定的 S3 儲存儲體設定儲存儲體政策：

- 調用卓越亞馬遜客户端的`setBucketPolicy`並向其提供SetBucketPolicyRequest
- 直接通過使用`setBucketPolicy`採用存儲桶名稱和策略文本（JSON 格式）的重載

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Principal;
```

Code

```
    s3.setBucketPolicy(bucket_name, policy_text);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
```

```
    System.exit(1);
}
```

使用策略類生成或驗證策略

將儲存儲體政策提供給setBucketPolicy，您可以執行下列作業：

- 將策略直接指定為 JSON 格式的文本字符串
- 使用政策類別

通過使用Policy類，您也不需要為文本字符串的正確格式化。要獲取 JSON 策略文本，請從Policy類，請使用其toJson方法。

匯入

```
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
        new Statement(Statement.Effect.Allow)
                .withPrincipals(Principal.AllUsers)
                .withActions(S3Actions.GetObject)
                .withResources(new Resource(
                        "{region-arn}s3:::" + bucket_name + "/*")));
return bucket_policy.toJson();
```

所以此Policy類也提供fromJson方法，該方法可以嘗試使用傳入的 JSON 字符串構建策略。該方法對其進行驗證，以確保文本可以轉換為有效的策略結構，並且將失敗並返回IllegalArgumentException（如果策略文本無效）。

```
Policy bucket_policy = null;
try {
    bucket_policy = Policy.fromJson(file_text.toString());
} catch (IllegalArgumentException e) {
    System.out.format("Invalid policy text in file: \"%s\"",
```

```
            policy_file);
    System.out.println(e.getMessage());
}
```

您可以使用此技術對從文件或其他方式讀入的策略進行預驗證。

請參閱 GitHub 上的完整範例。

取得儲存儲體政策

若要檢索Amazon S3存儲桶，請調用卓越亞馬遜客户端的getBucketPolicy方法，將其傳遞要從中獲取策略的存儲桶的名稱。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
  try {
      BucketPolicy bucket_policy = s3.getBucketPolicy(bucket_name);
      policy_text = bucket_policy.getPolicyText();
  } catch (AmazonServiceException e) {
      System.err.println(e.getErrorMessage());
      System.exit(1);
  }
```

如果指定的存儲桶不存在，如果您無權訪問該存儲桶，或者如果它沒有存儲桶策略，則AmazonServiceException被拋回。

請參閱 GitHub 上的完整範例。

刪除儲存貯體政策

要刪除存儲桶策略，請調用 AmazonS3 客户端的deleteBucketPolicy，為其提供儲存儲體名稱。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

Code

```
try {
    s3.deleteBucketPolicy(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

即使存儲桶沒有策略，此方法也會成功。如果您指定的存儲桶名稱不存在，或者如果您沒有訪問該存儲桶的權限，則`AmazonServiceException`被拋回。

請參閱 GitHub 上的完整範例。

詳細資訊

- 存取原則語言概觀中的Amazon Simple Storage Service使用者指南
- 儲存貯體政策範例中的Amazon Simple Storage Service使用者指南

使用 TransferManager 為了Amazon S3操作

您可以使用AWS SDK for Java TransferManager 類可靠地將文件從本地環境傳輸到Amazon S3並將對象從一個 S3 位置複製到另一個 S3 位置。`TransferManager`可以獲取傳輸的進度，暫停或恢復上傳和下載。

> **Note**
>
> 最佳實務
> 我們建議您在 儲存貯體上啟用 AbortIncompleteMultipartUploadAmazon S3 生命週期規則。此規則指示 Amazon S3 中止啟動後未在指定天數內完成的分段上傳。超過設定的時間限制時，Amazon S3 會中止上傳，然後刪除未完成的上傳資料。
> 如需詳細資訊，請參閱「」具版本控制之儲存儲體的生命週期組態中的Amazon S3使用者指南。

> **Note**
>
> 這些代碼示例假定您理解使用AWS SDK for Java並配置了默認AWS憑據使用設定AWS發展資料和區域。

上傳檔案及目錄

TransferManager 可以將文件、文件列表和目錄上傳到任何Amazon S3存儲桶先前創建的。

主題

- 上傳單一檔案
- 上傳文件列表
- 上傳目錄

上傳單一檔案

呼叫轉移管理器upload方法，提供Amazon S3存儲桶名稱、鍵（對象）名稱和標準 JavaFile (檔案)物件代表要上傳的檔案。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload xfer = xfer_mgr.upload(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
```

```
    //  or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

所以此upload方法返回立即，提供Upload物件以用於檢查傳輸狀態或等候它完成。

請參閱等待傳輸完成如需詳細資訊，請參waitForCompletion以成功完成轉接，然後再調用轉移管理器的shutdownNow方法。在等待傳輸完成時，您可以輪詢或偵聽有關其狀態和進度的更新。請參閱獲取傳輸狀態和進度如需詳細資訊。

請參閱 GitHub 上的完整範例。

上傳文件列表

若要在一個操作中上傳多個文件，請調用 TransferManageruploadFileList方法，提供下列各項：

- 同時Amazon S3儲存貯體名稱
- 一個索引鍵字首在創建對象的名稱前面（存儲桶中放置對象的路徑）
- 一個File (檔案)對象，表示要從中創建文件路徑的相對目錄
- 一個清單對象，其中包含一組File (檔案)上傳物件

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
ArrayList<File> files = new ArrayList<File>();
```

```
for (String path : file_paths) {
    files.add(new File(path));
}

TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadFileList(bucket_name,
            key_prefix, new File("."), files);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待傳輸完成如需詳細資訊，請參waitForCompletion以成功完成轉接，然後再調用轉移管理器的shutdownNow方法。在等待傳輸完成時，您可以輪詢或偵聽有關其狀態和進度的更新。請參閱獲取傳輸狀態和進度如需詳細資訊。

所以此MultipleFileUpload對象返回uploadFileList可用於查詢傳輸狀態或進度。請參閱輪詢傳輸的當前進度和使用 ProgressListener 獲取傳輸進度如需詳細資訊。

您也可以使用MultipleFileUpload的getSubTransfers方法來獲取單個Upload對象為每個要傳輸的文件。如需詳細資訊，請參閱「」獲取子傳輸的進度。

請參閱 GitHub 上的完整範例。

上傳目錄

您可以使用轉接管理器的uploadDirectory方法上傳整個文件目錄，並選擇遞歸複製子目錄中的文件。您提供Amazon S3存儲桶名稱、S3 key prefix、File (檔案)對象，表示要複製的本地目錄，boolean值，指示是否要遞歸地複製子目錄（真的或者假的。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;
```

```
import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    MultipleFileUpload xfer = xfer_mgr.uploadDirectory(bucket_name,
            key_prefix, new File(dir_path), recursive);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待傳輸完成如需詳細資訊，請參waitForCompletion以成功完成轉接，然後再調用轉移管理器的shutdownNow方法。在等待傳輸完成時，您可以輪詢或偵聽有關其狀態和進度的更新。請參閱獲取傳輸狀態和進度如需詳細資訊。

所以此MultipleFileUpload對象返回uploadFileList可用於查詢傳輸狀態或進度。請參閱輪詢傳輸的當前進度和使用 ProgressListener 獲取傳輸進度如需詳細資訊。

您也可以使用MultipleFileUpload的getSubTransfers方法來獲取單個Upload對象為每個要傳輸的文件。如需詳細資訊，請參閱「」獲取子傳輸的進度。

請參閱 GitHub 上的完整範例。

下載檔案或目錄

使用 TransferManager 類下載單個文件 (Amazon S3對象）或一個目錄（ Amazon S3存儲桶名稱後跟對象前綴）Amazon S3。

主題

- 下載單一檔案
- 下載目錄

下載單一檔案

使用轉讓管理器的download方法，提供Amazon S3存儲桶名稱，其中包含要下載的對象、密鑰（對象）名稱以及File (檔案)物件，用於代表要在本機系統上創建的檔案。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Download xfer = xfer_mgr.download(bucket_name, key_name, f);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待傳輸完成如需詳細資訊，請參waitForCompletion以成功完成轉接，然後再調用轉移管理器的shutdownNow方法。在等待傳輸完成時，您可以輪詢或偵聽有關其狀態和進度的更新。請參閱獲取傳輸狀態和進度如需詳細資訊。

請參閱 GitHub 上的完整範例。

下載目錄

要下載一組共享通用 key prefix（類似於文件系統上的目錄）的文件，請從Amazon S3，請使用TransferManagerdownloadDirectory方法。方法採用Amazon S3存儲桶名稱，其中包含要下載的

對象、所有對象共享的對象前綴以及File (檔案)對象，該對象表示要在本地系統上將文件下載到的目錄。如果命名目錄尚不存在，則會創建。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Download;
import com.amazonaws.services.s3.transfer.MultipleFileDownload;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;

import java.io.File;
```

Code

```
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();

try {
    MultipleFileDownload xfer = xfer_mgr.downloadDirectory(
            bucket_name, key_prefix, new File(dir_path));
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱等待傳輸完成如需詳細資訊，請參waitForCompletion以成功完成轉接，然後再調用轉移管理器的shutdownNow方法。在等待傳輸完成時，您可以輪詢或偵聽有關其狀態和進度的更新。請參閱獲取傳輸狀態和進度如需詳細資訊。

請參閱 GitHub 上的完整範例。

複製物件

若要將物件從 S3 儲存儲體複製到另一個儲存儲體，請使用 TransferManagercopy方法。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
```

Code

```
System.out.println("Copying s3 object: " + from_key);
System.out.println("      from bucket: " + from_bucket);
System.out.println("     to s3 object: " + to_key);
System.out.println("        in bucket: " + to_bucket);

TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Copy xfer = xfer_mgr.copy(from_bucket, from_key, to_bucket, to_key);
    // loop with Transfer.isDone()
    XferMgrProgress.showTransferProgress(xfer);
    // or block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(xfer);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱 GitHub 上的完整範例。

等待傳輸完成

如果您的應用程序（或線程）可以阻塞直到傳輸完成，則可以使用轉接界面的`waitForCompletion`方法阻止，直到傳輸完成或發生異常。

```
try {
    xfer.waitForCompletion();
} catch (AmazonServiceException e) {
    System.err.println("Amazon service error: " + e.getMessage());
    System.exit(1);
} catch (AmazonClientException e) {
    System.err.println("Amazon client error: " + e.getMessage());
    System.exit(1);
} catch (InterruptedException e) {
    System.err.println("Transfer interrupted: " + e.getMessage());
```

```
    System.exit(1);
}
```

如果你輪詢事件，你會得到傳輸進度之前呼叫`waitForCompletion`，在單獨的線程上實現輪詢機制，或者使用ProgressListener。

請參閱 GitHub 上的完整範例。

獲取傳輸狀態和進度

TransferManager 返回的每個類`upload*`、`download*`，和`copy`方法返回以下類之一的實例，具體取決於它是單文件操作還是多文件操作。

類別	退回方
Copy (複製)	`copy`
下載	`download`
MultipleFileDownload	`downloadDirectory`
上傳	`upload`
MultipleFileUpload	`uploadFileList` ，`uploadDirectory`

所有這些類都實現了轉接界面。`Transfer`提供了有用的方法來獲取傳輸進度、暫停或恢復傳輸以及獲取傳輸的當前或最終狀態。

主題

- 輪詢傳輸的當前進度
- 使用 ProgressListener 獲取傳輸進度
- 獲取子傳輸的進度

輪詢傳輸的當前進度

此循環會打印傳輸的進度，檢查其運行時的當前進度，並在完成後打印其最終狀態。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
// print the transfer's human-readable description
System.out.println(xfer.getDescription());
// print an empty progress bar...
printProgressBar(0.0);
// update the progress bar while the xfer is ongoing.
do {
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        return;
    }
    // Note: so_far and total aren't used, they're just for
    // documentation purposes.
    TransferProgress progress = xfer.getProgress();
    long so_far = progress.getBytesTransferred();
    long total = progress.getTotalBytesToTransfer();
    double pct = progress.getPercentTransferred();
    eraseProgressBar();
    printProgressBar(pct);
} while (xfer.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = xfer.getState();
System.out.println(": " + xfer_state);
```

請參閱 GitHub 上的完整範例。

使用 ProgressListener 獲取傳輸進度

您可以將ProgressListener轉移到任何使用轉接界面的addProgressListener方法。

一個ProgressListener只需要一種方法，progressChanged，它需要一個ProgressEvent物件。您可以使用該對象來獲取操作的總字節，方法是調用其getBytes方法，以及到目前為止通過調用getBytesTransferred。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
File f = new File(file_path);
TransferManager xfer_mgr = TransferManagerBuilder.standard().build();
try {
    Upload u = xfer_mgr.upload(bucket_name, key_name, f);
    // print an empty progress bar...
    printProgressBar(0.0);
    u.addProgressListener(new ProgressListener() {
        public void progressChanged(ProgressEvent e) {
            double pct = e.getBytesTransferred() * 100.0 / e.getBytes();
            eraseProgressBar();
            printProgressBar(pct);
        }
    });
    // block with Transfer.waitForCompletion()
    XferMgrProgress.waitForCompletion(u);
    // print the final state of the transfer.
    TransferState xfer_state = u.getState();
    System.out.println(": " + xfer_state);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
xfer_mgr.shutdownNow();
```

請參閱 GitHub 上的完整範例。

獲取子傳輸的進度

所以此MultipleFileUpload類可以返回有關其子傳輸的信息，方法是調用其getSubTransfers方法。它會傳回一個不可修改的收集的上傳對象，提供每個子傳輸的單個傳輸狀態和進度。

匯入

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.*;
import com.amazonaws.services.s3.transfer.Transfer.TransferState;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
```

Code

```
Collection<? extends Upload> sub_xfers = new ArrayList<Upload>();
sub_xfers = multi_upload.getSubTransfers();

do {
    System.out.println("\nSubtransfer progress:\n");
    for (Upload u : sub_xfers) {
        System.out.println("  " + u.getDescription());
        if (u.isDone()) {
            TransferState xfer_state = u.getState();
            System.out.println("  " + xfer_state);
        } else {
            TransferProgress progress = u.getProgress();
            double pct = progress.getPercentTransferred();
            printProgressBar(pct);
            System.out.println();
        }
    }

    // wait a bit before the next update.
    try {
        Thread.sleep(200);
```

```
    } catch (InterruptedException e) {
        return;
    }
} while (multi_upload.isDone() == false);
// print the final state of the transfer.
TransferState xfer_state = multi_upload.getState();
System.out.println("\nMultipleFileUpload " + xfer_state);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 物件金鑰中的Amazon Simple Storage Service使用者指南

設定Amazon S3存儲桶即網站

您可以設定Amazon S3儲存儲體作為網站。若要這樣做，您需要設定其網站配置。

> **Note**
>
> 這些代碼示例假定您理解使用AWS SDK for Java並配置了默認AWS憑據使用設定AWS全權證書和區域促進發展。

設定儲存儲體的網站組態

若要設置Amazon S3存儲桶的網站配置，請調用卓越亞馬遜的`setWebsiteConfiguration`方法與要設置配置的存儲桶名稱，以及BucketWebsiteConfiguration對象，其中包含存儲桶的網站配置。

設定索引文件是必需的；所有其他參數都是選用參數。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
```

Code

```
    String bucket_name, String index_doc, String error_doc) {
BucketWebsiteConfiguration website_config = null;

if (index_doc == null) {
    website_config = new BucketWebsiteConfiguration();
} else if (error_doc == null) {
    website_config = new BucketWebsiteConfiguration(index_doc);
} else {
    website_config = new BucketWebsiteConfiguration(index_doc, error_doc);
}

final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.setBucketWebsiteConfiguration(bucket_name, website_config);
} catch (AmazonServiceException e) {
    System.out.format(
            "Failed to set website configuration for bucket '%s'!\n",
            bucket_name);
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
```

> **Note**
>
> 設置網站配置不會修改存儲桶的訪問權限。要使您的文件在 Web 上可見，您還需要設置儲存儲體政策，它允許公用讀取儲存儲體中的文件。如需詳細資訊，請參閱「」管理對 的存取Amazon S3儲存儲體使用儲存儲體政策。

請參閱 GitHub 上的完整範例。

獲取存儲桶的網站配置

若要獲取Amazon S3存儲桶的網站配置，請調用卓越亞馬遜的`getWebsiteConfiguration`方法，其中包含要檢索其配置的儲存儲體的名稱。

配置將作為BucketWebsiteConfiguration物件。如果存儲桶沒有網站配置，則`null`將被返回。

匯入

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    BucketWebsiteConfiguration config =
            s3.getBucketWebsiteConfiguration(bucket_name);
    if (config == null) {
        System.out.println("No website configuration found!");
    } else {
        System.out.format("Index document: %s\n",
                config.getIndexDocumentSuffix());
        System.out.format("Error document: %s\n",
                config.getErrorDocument());
    }
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to get website configuration!");
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

刪除儲存儲體的網站組態

刪除Amazon S3存儲桶的網站配置，請調用卓越亞馬遜的`deleteWebsiteConfiguration`方法，其中包含要從中刪除配置的儲存儲體名稱。

匯入

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
```

Code

```
final AmazonS3 s3 =
 AmazonS3ClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
try {
    s3.deleteBucketWebsiteConfiguration(bucket_name);
} catch (AmazonServiceException e) {
    System.err.println(e.getErrorMessage());
    System.out.println("Failed to delete website configuration!");
    System.exit(1);
}
```

請參閱 GitHub 上的完整範例。

詳細資訊

- PUT 儲存貯體網站中的Amazon S3API 參考
- GET 儲存貯體網站中的Amazon S3API 參考
- DELETE 儲存貯體網站中的Amazon S3API 參考

使用Amazon S3用戶端加密

使用Amazon S3加密客户端是一種方法，您可以為存儲在Amazon S3。本節中的範例演示了如何創建和配置Amazon S3加密客户端。

如果您是密碼學的新使用者，請參密碼編譯基礎中的AWSKMS 開發人員指南，瞭解加密術語和算法的基本概述。有關所有加密技術支持的信息AWS開發套件，請參AWS適用於 的 開發套件支援Amazon S3用戶端加密中的Amazon Web Services一般參考。

> **Note**
>
> 這些代碼示例假定您理解使用AWS SDK for Java並配置了默認AWS憑據使用設定AWS全權證書和區域促進發展。

如果您使用的是 1.11.836 或更早版本的AWS SDK for Java，請參Amazon S3加密用户端遷移，瞭解有關將應用程序遷移到更高版本的信息。如果無法遷移，請參閱這個完整的示例(在 GitHub 上)。

否則，如果您使用的是版本是 1.11.837 或更新版AWS SDK for Java，請瀏覽下面列出的示例主題以使用Amazon S3用户端加密。

主題

- Amazon S3使用用於用户端密鑰的加密
- Amazon S3使用AWSKMS 託管密鑰

Amazon S3使用用於用户端密鑰的加密

下列範例使用亞馬遜 3 加密客户端 2 構建器類別創建Amazon S3啟用了用户端加密功能。啟用後，您將上傳到Amazon S3使用此客户端將被加密。您從中獲取的任何對象Amazon S3將自動解密。

> **Note**
>
> 下列範例示範如何使用Amazon S3使用由用户管理的用户端密鑰進行加密。若要了解如何搭配使用加密功能AWSKMS 託管密鑰，請參閱Amazon S3使用AWSKMS 託管密鑰。

啟用客户端時，您可以從兩種加密模式中進行選擇Amazon S3加密：嚴格的身份驗證或身份驗證。下列章節介紹如何啟用每種類型。要瞭解每種模式使用哪些算法，請參閱CryptoMode定義。

所需進口

為這些示例導入以下類。

匯入

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.services.s3.model.CryptoConfigurationV2;
import com.amazonaws.services.s3.model.CryptoMode;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.StaticEncryptionMaterialsProvider;
```

嚴格驗證加密

如果沒有，則嚴格驗證加密為默認模式`CryptoMode`指定。

要顯式啟用此模式，請指定`StrictAuthenticatedEncryption`中的值`withCryptoConfiguration`方法。

> **Note**
>
> 若要使用用户端驗證加密，必須包含最新的充氣城堡罐文件在應用程序的類路徑中。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
         .build();

s3Encryption.putObject(bucket_name, ENCRYPTED_KEY2, "This is the 2nd content to
 encrypt");
```

驗證加密模式

當您使用AuthenticatedEncryption模式下，在加密過程中應用改進的密鑰包裝算法。在此模式下解密時，算法可以驗證解密對象的完整性，並在檢查失敗時拋出異常。若要了解驗證加密工作方式的詳細資訊，請參Amazon S3驗證加密博客文章。

> **Note**
>
> 若要使用用户端驗證加密，必須包含最新的充氣城堡罐文件在應用程序的類路徑中。

要啟用此模式，請指定AuthenticatedEncryption中的值withCryptoConfiguration方法。

Code

```
AmazonS3EncryptionV2 s3EncryptionClientV2 =
 AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.DEFAULT_REGION)
         .withClientConfiguration(new ClientConfiguration())
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode(CryptoMode.AuthenticatedEncryption))
         .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
 EncryptionMaterials(secretKey)))
```

```
        .build();

s3EncryptionClientV2.putObject(bucket_name, ENCRYPTED_KEY1, "This is the 1st content to
 encrypt");
```

Amazon S3使用AWSKMS 託管密鑰

下列範例使用亞馬遜 3 加密客户端 2 構建器類別建立Amazon S3啟用了用户端加密的用户端加密功能。配置後，您將上傳到Amazon S3將被加密。您從中獲取的任何對象Amazon S3將自動解密。

> **Note**
>
> 下列範例會示範如何使用Amazon S3使用AWSKMS 託管密鑰。若要了解如何使用自己的密鑰使用加密功能，請參Amazon S3使用用户端密鑰。

啟用客户端時，您可以從兩種加密模式中進行選擇Amazon S3加密：嚴格的身份驗證或驗證。下列各節會示範如何啟用每種類型。要瞭解每種模式使用哪些算法，請參閱CryptoMode 碼定義。

所需進口

為這些示例導入以下類。

匯入

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.GenerateDataKeyRequest;
import com.amazonaws.services.kms.model.GenerateDataKeyResult;
import com.amazonaws.services.s3.AmazonS3EncryptionClientV2Builder;
import com.amazonaws.services.s3.AmazonS3EncryptionV2;
import com.amazonaws.services.s3.model.CryptoConfigurationV2;
import com.amazonaws.services.s3.model.CryptoMode;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.KMSEncryptionMaterialsProvider;
```

嚴格驗證加密

如果沒有，則嚴格驗證加密為默認模式`CryptoMode`指定為。

要顯式啟用此模式，請指定`StrictAuthenticatedEncryption`中的值`withCryptoConfiguration`方法。

> **Note**
>
> 若要使用經驗證的加密功能，您必須包含最新的Bouncy Castle文件在應用程序的類路徑中。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.StrictAuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();

s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

呼叫`putObject`方法Amazon S3加密客户端上傳對象。

Code

```
s3Encryption.putObject(bucket_name, ENCRYPTED_KEY3, "This is the 3rd content to encrypt
 with a key created in the {console}");
```

您可以使用相同的用户端檢索對象。此示例調用`getObjectAsString`方法來檢索存儲的字符串。

Code

```
System.out.println(s3Encryption.getObjectAsString(bucket_name, ENCRYPTED_KEY3));
```

驗證加密模式

當您使用`AuthenticatedEncryption`模式下，在加密過程中應用改進的密鑰包裝算法。在此模式下解密時，算法可以驗證解密對象的完整性，並在檢查失敗時拋出異常。如需有關驗證加密工作方式的詳細資訊，請參閱Amazon S3驗證用户端加密博客文章。

> **Note**
>
> 若要使用經驗證的加密功能，您必須包含最新的Bouncy Castle文件在應用程序的類路徑中。

要啟用此模式，請指定AuthenticatedEncryption中的值withCryptoConfiguration方法。

Code

```
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
         .withRegion(Regions.US_WEST_2)
         .withCryptoConfiguration(new
 CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
         .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
         .build();
```

設定AWS KMS客戶

所以此Amazon S3加密用户端創建AWS KMS客户端，除非明確指定了一個。

設置此自動創建的區域AWS KMS客户端，將awsKmsRegion。

Code

```
Region kmsRegion = Region.getRegion(Regions.AP_NORTHEAST_1);

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
        .withRegion(Regions.US_WEST_2)
        .withCryptoConfiguration(new
 CryptoConfigurationV2().withAwsKmsRegion(kmsRegion))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

或者，您可以使用自己的AWS KMS客户端初始化加密客户端。

Code

```
AWSKMS kmsClient = AWSKMSClientBuilder.standard()
        .withRegion(Regions.US_WEST_2);
        .build();

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
```

```
        .withRegion(Regions.US_WEST_2)
        .withKmsClient(kmsClient)
        .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode((CryptoMode.AuthenticatedEncryption)))
        .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))
        .build();
```

使用 AWS SDK for Java 的 Amazon SQS 範例

本節提供使用AWS SDK for Java編寫 Amazon SQS 程式的範例。

> **Note**
>
> 範例僅包含示範每個技術所需的程式碼。GitHub 上提供完整程式碼範例。您可以從那裡下載單一原始檔案或將儲存庫複製到本機，以取得建置和執行的所有範例。

主題

- 使用Amazon SQS訊息佇列
- 傳送、接收和刪除Amazon SQS訊息
- 為啟用長輪詢Amazon SQS訊息 QueueRequest
- 設定可見性逾時Amazon SQS
- 在 Amazon SQS 中使用無效字母佇列

使用Amazon SQS訊息佇列

訊息佇列是邏輯容器，用於在 Amazon SQS 中可靠地傳送訊息。有兩種佇列類型：標準和先進先出 (FIFO)。若要進一步了解佇列和這些類型之間的差異，請參Amazon SQS開發人員指南。

本主題說明如何使用AWS SDK for Java建立、列出、刪除和取得 Amazon SQS 佇列的 URL。

建立佇列

使用卓越亞馬遜 SQS 客戶端的`createQueue`方法，提供CreateQueueRequest物件，描述佇列參數。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
```

```
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
CreateQueueRequest create_request = new CreateQueueRequest(QUEUE_NAME)
        .addAttributesEntry("DelaySeconds", "60")
        .addAttributesEntry("MessageRetentionPeriod", "86400");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

您可以使用簡化形式`createQueue`（只需要隊列名稱）來創建標準隊列。

```
sqs.createQueue("MyQueue" + new Date().getTime());
```

請參閱 GitHub 上的完整範例。

列出佇列

若要列出Amazon SQS佇列，請呼叫 AmazonSQS 客户端的`listQueues`方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesResult;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
ListQueuesResult lq_result = sqs.listQueues();
System.out.println("Your SQS Queue URLs:");
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
```

```
}
```

使用listQueues重載沒有任何參數返回所有佇列。您可以依以下項目篩選返回的結果：ListQueuesRequest物件。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ListQueuesRequest;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String name_prefix = "Queue";
lq_result = sqs.listQueues(new ListQueuesRequest(name_prefix));
System.out.println("Queue URLs with prefix: " + name_prefix);
for (String url : lq_result.getQueueUrls()) {
    System.out.println(url);
}
```

請參閱 GitHub 上的完整範例。

取得佇列 URL

致電亞馬遜 SQS 客户的getQueueUrl方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
String queue_url = sqs.getQueueUrl(QUEUE_NAME).getQueueUrl();
```

請參閱 GitHub 上的完整範例。

刪除佇列

提供佇列的URL到卓越亞馬遜 SQS 客户的deleteQueue方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
sqs.deleteQueue(queue_url);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 操作說明Amazon SQS佇列的運作中的Amazon SQS開發人員指南
- CreateQueue中的Amazon SQSAPI 參考
- GetQueueUrl中的Amazon SQSAPI 參考
- ListQueues中的Amazon SQSAPI 參考
- DeleteQueues中的Amazon SQSAPI 參考

傳送、接收和刪除Amazon SQS訊息

此主題說明如何傳送、接收和刪除Amazon SQS訊息。訊息一律使用 SQS 佇列來傳送。

傳送訊息

將單個訊息添加到Amazon SQS隊列，方法是調用卓越亞馬遜客户端的sendMessage方法。提供 SendMessageRequest 物件，其中包含佇列的 URL、訊息本文，以及選用的延遲值 (以秒為單位)。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
```

Code

```
SendMessageRequest send_msg_request = new SendMessageRequest()
```

```
        .withQueueUrl(queueUrl)
        .withMessageBody("hello world")
        .withDelaySeconds(5);
sqs.sendMessage(send_msg_request);
```

請參閱 GitHub 上的完整範例。

一次傳送多個訊息

您可在單一請求中傳送多個訊息。要發送多條消息，請使用 AmazonSQS 客户端的`sendMessageBatch`方法，它採用SendMessageBatchRequest，其中包含佇列 URL 和訊息列表（每個都是SendMessageBatchRequestEntry）傳送。您還可以為每條消息設置一個可選的延遲值。

匯入

```
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
import com.amazonaws.services.sqs.model.SendMessageBatchRequestEntry;
```

Code

```
SendMessageBatchRequest send_batch_request = new SendMessageBatchRequest()
        .withQueueUrl(queueUrl)
        .withEntries(
                new SendMessageBatchRequestEntry(
                        "msg_1", "Hello from message 1"),
                new SendMessageBatchRequestEntry(
                        "msg_2", "Hello from message 2")
                        .withDelaySeconds(10));
sqs.sendMessageBatch(send_batch_request);
```

請參閱 GitHub 上的完整範例。

接收訊息

呼叫 AmazonSQS 客户端的`receiveMessage`方法，可傳遞佇列的 URL。訊息會以 Message 物件清單的形式傳回。

匯入

```
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
```

```
import com.amazonaws.services.sqs.model.SendMessageBatchRequest;
```

Code

```
List<Message> messages = sqs.receiveMessage(queueUrl).getMessages();
```

收到後刪除留言

收到訊息並處理它的內容後，傳送訊息的接收控點和佇列 URL 給 AmazonSQS 客戶端的`deleteMessage`方法。

Code

```
for (Message m : messages) {
    sqs.deleteMessage(queueUrl, m.getReceiptHandle());
}
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 操作說明Amazon SQS佇列的運作方式中的Amazon SQS開發人員指南
- SendMessage中的Amazon SQSAPI 參考
- SendMessageBatch中的Amazon SQSAPI 參考
- ReceiveMessage中的Amazon SQSAPI 參考
- DeleteMessage中的Amazon SQSAPI 參考

為啟用長輪詢Amazon SQS訊息 QueueRequest

Amazon SQS使用短輪詢預設使用僅會查詢服務器的子集 (按照加權隨機分佈) 來判定是否有訊息可加入回應之中。

長輪詢有助於降低使用Amazon SQS通過減少空響應的數量，當沒有可用於返回的消息以回覆 ReceiveMessage 請求發送至Amazon SQS隊列並消除錯誤的空響應。

> **Note**
>
> 您可以將長輪詢頻率從1-20 秒。

在建立排列時啟用長輪詢

若要在創建Amazon SQS隊列中，設置`ReceiveMessageWaitTimeSeconds`上的屬性`CreateQueueRequest`對象之前調用卓越亞馬遜 SQS 類`createQueue`方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
```

Code

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Enable long polling when creating a queue
CreateQueueRequest create_request = new CreateQueueRequest()
        .withQueueName(queue_name)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");

try {
    sqs.createQueue(create_request);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

請參閱 GitHub 上的完整範例。

在現有佇列上啟用長輪詢

除了在創建隊列時啟用長輪詢外，還可以通過設置`ReceiveMessageWaitTimeSeconds`在`SetQueueAttributesRequest`之前調用卓越亞馬遜 SQS 類`setQueueAttributes`方法。

匯入

```
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
SetQueueAttributesRequest set_attrs_request = new SetQueueAttributesRequest()
        .withQueueUrl(queue_url)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");
sqs.setQueueAttributes(set_attrs_request);
```

請參閱 GitHub 上的完整範例。

在收到訊息時啟用長輪詢

您可以在接收消息時啟用長輪詢，方法是在ReceiveMessageRequest您提供給卓越亞馬遜 SQS 類的`receiveMessage`方法。

> **Note**
>
> 您應該確保AWS客户端的請求超時大於最長輪詢時間 (20s)，因此您的`receiveMessage`請求在等待下一個輪詢事件時不會超時！

匯入

```
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
```

Code

```
ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queue_url)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- Amazon SQS長輪詢中的Amazon SQS開發人員指南
- CreateQueue中的Amazon SQSAPI 參考
- ReceiveMessage中的Amazon SQSAPI 參考
- SetQueueAttributes中的Amazon SQSAPI 參考

設定可見性逾時Amazon SQS

當收到消息時Amazon SQS，它一直保留在隊列中，直到它被刪除，以確保收到。已收到但未刪除的消息將在給定可見性逾時，以幫助防止在處理和刪除郵件之前多次接收消息。

> **Note**
>
> 當您使用標準隊列，可見性逾時並不保證不會收到兩次訊息。如果您使用的是標準隊列，請確保您的代碼可以處理同一消息多次傳遞的情況。

設定單個訊息的訊息可見性逾時

收到消息後，您可以修改其可見性超時，方法是將其收據句柄傳遞到ChangeMessageVisibilityRequest您傳遞給卓越亞馬遜 SQS 類changeMessageVisibility方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Get the receipt handle for the first message in the queue.
String receipt = sqs.receiveMessage(queue_url)
                    .getMessages()
                    .get(0)
                    .getReceiptHandle();

sqs.changeMessageVisibility(queue_url, receipt, timeout);
```

請參閱 GitHub 上的完整範例。

一次設置多條消息的消息可見性超時

若要同時為多條消息設置消息可見性超時，請創建ChangeMessageVisibilityBatchRequestEntry對象，每個對象都包含一個唯一的 ID 字符串和一個收據句柄。然後，將列表傳遞給Amazon SQS客戶端類changeMessageVisibilityBatch方法。

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.ChangeMessageVisibilityBatchRequestEntry;
import java.util.ArrayList;
import java.util.List;
```

Code

```
AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

List<ChangeMessageVisibilityBatchRequestEntry> entries =
    new ArrayList<ChangeMessageVisibilityBatchRequestEntry>();

entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg1",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout));

entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "unique_id_msg2",
            sqs.receiveMessage(queue_url)
               .getMessages()
               .get(0)
               .getReceiptHandle())
        .withVisibilityTimeout(timeout + 200));

sqs.changeMessageVisibilityBatch(queue_url, entries);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 可見性逾時中的Amazon SQS開發人員指南
- SetQueueAttributes中的Amazon SQSAPI 參考
- GetQueueAttributes中的Amazon SQSAPI 參考
- ReceiveMessage中的Amazon SQSAPI 參考

- ChangeMessageVisibility中的Amazon SQSAPI 參考
- ChangeMessageVisibilityBatch中的Amazon SQSAPI 參考

在 Amazon SQS 中使用無效字母佇列

Amazon SQS支援無效字母排列。無效字母佇列即為其他 (來源) 佇列，可以針對無法成功處理的訊息以進行作業。您可以在無效字母佇列中擱置並隔離這類訊息，以確定無法成功處理訊息的原因。

創建無效字母佇列

死信隊列的創建方式與常規隊列相同，但具有以下限制：

- 死信隊列必須與源隊列與隊列類型相同（FIFO 或標準）。
- 死信隊列必須使用相同的AWS 帳戶和區域作為來源佇列。

在這裏，我們創建兩個相同Amazon SQS隊列，其中一個將作為死信隊列：

匯入

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.AmazonSQSException;
```

Code

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

// Create source queue
try {
    sqs.createQueue(src_queue_name);
} catch (AmazonSQSException e) {
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}

// Create dead-letter queue
try {
    sqs.createQueue(dl_queue_name);
} catch (AmazonSQSException e) {
```

```
    if (!e.getErrorCode().equals("QueueAlreadyExists")) {
        throw e;
    }
}
```

請參閱 GitHub 上的完整範例。

為來源佇列指定無效字母佇列

要指定死信隊列，您必須首先創建重新驅動策略，然後在隊列的屬性中設置策略。重新驅動器策略在 JSON 中指定，它指定死信隊列的 ARN 以及在將消息發送到死信隊列之前可以接收和不處理的最大次數。

要為源隊列設置重新驅動策略，請調用 AmazonSQS 類的`setQueueAttributes`方法與SetQueueAttributesRequest對象，您已為其設置`RedrivePolicy`屬性與 JSON 重新驅動策略相結合。

匯入

```
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesResult;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

Code

```
String dl_queue_url = sqs.getQueueUrl(dl_queue_name)
                         .getQueueUrl();

GetQueueAttributesResult queue_attrs = sqs.getQueueAttributes(
        new GetQueueAttributesRequest(dl_queue_url)
            .withAttributeNames("QueueArn"));

String dl_queue_arn = queue_attrs.getAttributes().get("QueueArn");

// Set dead letter queue with redrive policy on source queue.
String src_queue_url = sqs.getQueueUrl(src_queue_name)
                          .getQueueUrl();

SetQueueAttributesRequest request = new SetQueueAttributesRequest()
        .withQueueUrl(src_queue_url)
        .addAttributesEntry("RedrivePolicy",
                "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\""
                + dl_queue_arn + "\"}");
```

```
sqs.setQueueAttributes(request);
```

請參閱 GitHub 上的完整範例。

詳細資訊

- 使用Amazon SQS無效信件佇列中的Amazon SQS開發人員指南
- SetQueueAttributes中的Amazon SQSAPI 參考

使用 AWS SDK for Java 的 Amazon SWF 範例

Amazon SWF是一項工作流管理服務，可幫助開發人員構建和擴展分佈式工作流，這些工作流可以具有並行或順序步驟，包括活動、子工作流或甚至Lambda任務。

有兩種方式可以使用Amazon SWF使用AWS SDK for Java，通過使用 SWF客戶對象，或通過使用 AWS Flow Framework。所以此AWS Flow Framework，因為它大量使用了註釋，並依賴於其他庫，如 AspectJ 和 Spring 框架。但是，對於大型或複雜項目，您可以通過使用AWS Flow Framework。如需詳細資訊，請參閱 。AWS Flow Framework的 Java 開發人員指南。

本節將提供編寫程式的範例Amazon SWF通過使用AWS SDK for Java客户端。

主題

- SWF 基本概況
- 建立簡單的Amazon SWF應用程式
- Lambda 任務
- 正常關閉活動和工作流工作人員
- 註冊網域
- 列出網域

SWF 基本概況

這些是一般的模式，用於使用Amazon SWF使用AWS SDK for Java。它主要用於參考。如需完整的入門教程，請參構建簡單Amazon SWF應用。

相依性

基本Amazon SWF應用程序將需要以下依賴項，這些依賴項包含在AWS SDK for Java：

- aws-java-sdk-1.12.*.jar
- 公共記錄 -1.2.*.jar
- http客户端 -4.3.*.jar
- httpco-4.3.*.jar
- 傑克遜註釋 -2.12.*.jar
- 傑克遜核心 -2.12.*.jar
- 傑克遜數據庫 -2.12.*.jar
- 約達時間 2.8.*.jar

> **Note**
>
> 這些軟件包的版本號因您所擁有的 SDK 版本而有所不同，但是軟件開發工具包提供的版本已經過了兼容性測試，並且是您應該使用的版本。

AWS Flow Framework對於 Java 應用程序需要額外的設置，和附加依賴關係。請參。AWS Flow Framework適用於 Java 開發人員指南，瞭解如何使用框架的詳細資訊。

匯入

通常，您可以使用以下導入進行代碼開發：

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

但是，只導入您需要的類是一個很好的做法。您可能最終會在`com.amazonaws.services.simpleworkflow.model`工作空間：

```
import com.amazonaws.services.simpleworkflow.model.PollForActivityTaskRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskCompletedRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskFailedRequest;
import com.amazonaws.services.simpleworkflow.model.TaskList;
```

如果您是使用AWS Flow Framework，則您將從`com.amazonaws.services.simpleworkflow.flow`工作空間。例如：

```
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
```

```
import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;
```

> **Note**
>
> 所以此AWS Flow Framework對於 Java 來說，除了基礎要求之外的其他要求AWS SDK for Java。如需詳細資訊，請參閱 。AWS Flow Framework適用於 Java 開發人員指南。

使用 SWF 客户端類

您的基本界面Amazon SWF是通過AmazonSimpleWorkflowClient或者AmazonSimpleWorkflowAsyncClient類別。這兩者之間的主要區別在於*AsyncClient類別返回未來對象進行併發（異步）編程。

```
AmazonSimpleWorkflowClient swf = AmazonSimpleWorkflowClientBuilder.defaultClient();
```

建立簡單的Amazon SWF應用程式

本主題將向您介紹使Amazon SWF用程式設計應用程式AWS SDK for Java，同時介紹一些重要概念。

關於此範例

範例專案會建立含有單一活動的工作流程，該活動會接受透過AWS雲端傳遞的工作流程資料 (在的傳統中HelloWorld，這將是要問候的人的名字)，然後列印回應的問候語。

雖然這在表面上看起來非常簡單，但Amazon SWF應用程序由許多一起工作的部分組成：

- 網域，用作工作流程執行資料的邏輯容器。
- 一或多個工作流程，代表定義工作流程活動和子工作流程執行的邏輯順序的程式碼元件。
- 工作流程背景工作者 (也稱為決策程式)，可輪詢決策任務並排程活動或子工作流程以回應。
- 一或多個活動，每個動作都代表工作流程中的一個工作單位。
- 一種活動工作者，輪詢活動任務並運行活動方法以響應。
- 一或多個任務清單，這些任務清單是由Amazon SWF用來向工作流程和活動 Worker 發出請求所維護的佇列。適用於工作流程工作者的任務清單上的任務稱為決策任務。那些意味著活動工作者稱為活動任務。
- 開始工作流程執行的工作流程啟動器。

在幕後，您可以Amazon SWF協調這些元件的作業、協調其來自AWS雲端的流程、在它們之間傳遞資料、處理逾時和活動訊號通知，以及記錄工作流程執行歷程記錄。

先決條件

開發環境

本教程中使用的開發環境包括：

- AWS SDK for Java。
- 阿帕奇 Maven（3.3.1）。
- JDK 1.7 或更新版本。本教程是使用 JDK 1.8.0 開發和測試的。
- 一個好的 Java 文本編輯器（您的選擇）。

> **Note**
>
> 如果您使用與 Maven 不同的構建系統，您仍然可以使用適合您環境的適當步驟創建項目，並使用此處提供的概念進行操作。AWS SDK for Java有關配置和使用各種建置系統的詳細資訊，請參閱入門。
> 同樣，但是通過更多的努力，此處顯示的步驟可以使用支持的任何 AWS SDK 來實現。Amazon SWF

所有必要的外部依賴項都包含在中AWS SDK for Java，因此沒有其他需要下載的內容。

AWS访问

若要順利完成本教學課程，您必須能夠存取AWS存取入口網站，如本指南的基本設定一節所述。

這些指示說明如何存取您複製並貼上至本機共用`credentials`檔案的臨時認證。您貼上的臨時登入資料必須與中具有存取 Amazon SWF 權限AWS IAM Identity Center的 IAM 角色相關聯。貼上臨時憑據之後，您的`credentials`檔案將看起來如下：

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token=IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZ2luX2IQoJb3JpZVERYLONGSTRI
```

這些臨時身份證明與default設定檔相關聯。

建立 SWF 專案

1. 使用 Maven 啟動一個新的專案：

```
mvn archetype:generate -DartifactId=helloswf \
-DgroupId=aws.example.helloswf -DinteractiveMode=false
```

   這將創建一個具有標準 maven 項目結構的新項目：

```
helloswf
### pom.xml
### src
    ### main
    #   ### java
    #       ### aws
    #           ### example
    #               ### helloswf
    #                   ### App.java
    ### test
        ### ...
```

   您可以忽略或刪除目test錄及其中包含的所有目錄，我們將不會在本教程中使用它。您也可以刪除App.java，因為我們將用新的類替換它。

2. 編輯項目的pom.xml文件，並通過在塊中添加依賴項來添加aws-java-sdk-simpleworkflow模<dependencies>塊。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-simpleworkflow</artifactId>
    <version>1.11.1000</version>
  </dependency>
</dependencies>
```

3. 確保 Maven 使用 JDK 1.7 + 支持構建您的項目。將以下內容添加到您的項目中（在<dependencies>塊之前或之後）pom.xml：

```
<build>
  <plugins>
```

```
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.6.1</version>
      <configuration>
          <source>1.8</source>
          <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

編碼項目

該示例項目將由四個單獨的應用程序組成，我們將逐個訪問它們：

- HelloTypes.java-包含項目的域，活動和工作流類型數據，與其他組件共享。它也會處理使用 SWF 註冊這些類型。
- ActivityWorker.java-包含活動工作者，該活動工作者輪詢活動任務並運行響應活動。
- WorkflowWorker.java-包含工作流程工作者（決策程序），它輪詢決策任務並安排新活動。
- WorkflowStarter.java-包含工作流啟動器，它會啟動新的工作流程執行，這將導致 SWF 開始產生決策和工作流程任務供您的工作者使用。

所有來源檔案的一般步驟

您創建用於容納 Java 類的所有文件都將具有一些共同點。為了時間的利益，每次將新文件添加到項目中時，都會隱含以下步驟：

1. 在專案`src/main/java/aws/example/helloswf/`目錄中建立檔案。
2. 在每個文件的開頭添加一`package`個聲明以聲明其命名空間。範例專案使用：

```
package aws.example.helloswf;
```

3. 為AmazonSimpleWorkflowClient類別和`com.amazonaws.services.simpleworkflow.model`命名空間中的多個類別新增`import`宣告。為了簡化事情，我們將使用：

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
```

```
import com.amazonaws.services.simpleworkflow.model.*;
```

註冊網域、工作流程和活動類型

我們將從創建一個新的可執行文件類開始HelloTypes.java。此檔案將包含工作流程中不同部分需要瞭解的共用資料，例如活動的名稱和版本和工作流程類型、網域名稱和工作清單名稱。

1. 打開文本編輯器並創建文件HelloTypes.java，添加包聲明並根據常見步驟導入。
2. 宣告類HelloTypes別，並為其提供用於已註冊活動和工作流程類型的值：

```
    public static final String DOMAIN = "HelloDomain";
    public static final String TASKLIST = "HelloTasklist";
    public static final String WORKFLOW = "HelloWorkflow";
    public static final String WORKFLOW_VERSION = "1.0";
    public static final String ACTIVITY = "HelloActivity";
    public static final String ACTIVITY_VERSION = "1.0";
```

   這些值將在整個代碼中使用。

3. 在 String 宣告之後，建立AmazonSimpleWorkflowClient類別的執行個體。這是由提供的Amazon SWF方法的基本介面AWS SDK for Java。

```
private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

   上一個程式碼片段假設暫時認證與設default定檔相關聯。如果您使用不同的設定檔，請依照下列方式修改上述程式碼，並將 *profile_name* 取代為實際設定檔名稱的名稱。

```
private static final AmazonSimpleWorkflow swf =
        AmazonSimpleWorkflowClientBuilder
                .standard()
                .withCredentials(new ProfileCredentialsProvider("profile_name"))
                .withRegion(Regions.DEFAULT_REGION)
                .build();
```

4. 新增註冊 SWF 網域的新函數。網域是許多相關 SWF 活動和工作流程類型的邏輯容器。只有當 SWF 組件存在於相同網域中時，才能彼此通訊。

```
    try {
```

```
        System.out.println("** Registering the domain '" + DOMAIN + "'.");
        swf.registerDomain(new RegisterDomainRequest()
            .withName(DOMAIN)
            .withWorkflowExecutionRetentionPeriodInDays("1"));
    } catch (DomainAlreadyExistsException e) {
        System.out.println("** Domain already exists!");
    }
```

註冊網域時，請提供名稱 (不包括、、:、/控制字元或文字字串 `arn' 的任何 1 至 256 個字元組) 以及保留期間，即在工作流程執行完成後保Amazon SWF留工作流程的執行歷程記錄資料的天數。|工作流程執行保留期上限為 90 天。RegisterDomainRequest如需詳細資訊，請參閱。

如果具有該名稱的域已經存在，則會引發DomainAlreadyExistsException一個。因為我們不擔心是否已經建立網域，所以我們可以忽略例外狀況。

> **Note**
>
> 此程式碼示範使用方法時的常見模式，AWS SDK for Java方法的資料是由`simpleworkflow.model`命名空間中的類別提供，您可以使用可鏈接`0with*`的方法來實例化和填入該類別。

5. 新增函數以註冊新活動類型。活動表示工作流程中的工作單位。

```
    try {
        System.out.println("** Registering the activity type '" + ACTIVITY +
            "-" + ACTIVITY_VERSION + "'.");
        swf.registerActivityType(new RegisterActivityTypeRequest()
            .withDomain(DOMAIN)
            .withName(ACTIVITY)
            .withVersion(ACTIVITY_VERSION)
            .withDefaultTaskList(new TaskList().withName(TASKLIST))
            .withDefaultTaskScheduleToStartTimeout("30")
            .withDefaultTaskStartToCloseTimeout("600")
            .withDefaultTaskScheduleToCloseTimeout("630")
            .withDefaultTaskHeartbeatTimeout("10"));
    } catch (TypeAlreadyExistsException e) {
        System.out.println("** Activity type already exists!");
    }
```

活動類型由名稱和版本識別，用於唯一識別其註冊網域中任何其他活動的活動。活動也包含數個選用參數，例如用來從 SWF 接收工作和資料的預設工作清單，以及許多不同的逾時，您可以使用這些

逾時來設定活動執行的不同部分所需的時間長度。RegisterActivityTypeRequest如需詳細資訊，請參閱。

> **Note**
>
> 所有逾時值均以秒為單位指定。如需Amazon SWF逾時如何影響工作流程執行的完整說明，請參閱逾時類型。

如果您嘗試註冊的活動類型已經存在，TypeAlreadyExistsException則會引發。新增函數以註冊新的工作流程類型。工作流程 (也稱為決策程式) 代表工作流程執行的邏輯。

+

```
    try {
        System.out.println("** Registering the workflow type '" + WORKFLOW +
            "-" + WORKFLOW_VERSION + "'.");
        swf.registerWorkflowType(new RegisterWorkflowTypeRequest()
            .withDomain(DOMAIN)
            .withName(WORKFLOW)
            .withVersion(WORKFLOW_VERSION)
            .withDefaultChildPolicy(ChildPolicy.TERMINATE)
            .withDefaultTaskList(new TaskList().withName(TASKLIST))
            .withDefaultTaskStartToCloseTimeout("30"));
    } catch (TypeAlreadyExistsException e) {
        System.out.println("** Workflow type already exists!");
    }
```

+

與活動類型類似，工作流程類型由名稱和版本識別，並且還具有可配置的逾時。RegisterWorkflowTypeRequest如需詳細資訊，請參閱。

+

如果您嘗試註冊的工作流程類型已經存在，TypeAlreadyExistsException則會引發。最後，提供一個`main`方法，讓類別可執行，該方法會依次註冊網域、活動類型和工作流程類型：

+

```
    registerDomain();
```

```
    registerWorkflowType();
    registerActivityType();
```

您現在可以建置並執行應用程式以執行註冊指令碼，或繼續編寫活動和工作流程 Worker 的程式碼。註冊網域、工作流程和活動之後，您就不需要再次執行此功能 — 這些類型會持續存在，直到您自行取代它們為止。

實施活動工作者

活動是工作流程中的基本工作單位。工作流程提供邏輯、排程要執行的活動 (或其他要採取的動作) 以回應決策任務。典型的工作流程通常由許多活動組成，這些活動可以同步執行、非同步或兩者的組合。

活動 Worker 是針對回應工作流程決策所產生之活動任務輪詢Amazon SWF的程式碼位元。當它收到活動任務時，它會執行對應的活動，並將成功/失敗回應傳回工作流程。

我們將實現一個簡單的活動工作者，驅動單個活動。

1. 打開文本編輯器並創建文件`ActivityWorker.java`，添加包聲明並根據常見步驟導入。

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
```

2. 將`ActivityWorker`類別新增至檔案，並為其指定資料成員，以存放 SWF 用戶端，我們將用來與之互動Amazon SWF：

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

3. 添加我們將用作活動的方法：

```
private static String sayHello(String input) throws Throwable {
    return "Hello, " + input + "!";
}
```

該活動只需要一個字符串，將其組合成一個問候語並返回結果。雖然這項活動幾乎沒有可能引發例外狀況，但最好設計出錯時可能會引發錯誤的活動。

4. 添加一個`main`方法，我們將用作活動任務輪詢方法。我們將通過添加一些代碼來輪詢活動任務的任務列表來啟動它：

```
        System.out.println("Polling for an activity task from the tasklist '"
                + HelloTypes.TASKLIST + "' in the domain '" +
                HelloTypes.DOMAIN + "'.");

        ActivityTask task = swf.pollForActivityTask(
            new PollForActivityTaskRequest()
                .withDomain(HelloTypes.DOMAIN)
                .withTaskList(
                    new TaskList().withName(HelloTypes.TASKLIST)));

        String task_token = task.getTaskToken();
```

活動會呼叫 SWF 用戶端Amazon SWF的pollForActivityTask方法，並指定要在傳入中使用的網域和工作清單來接收工作。PollForActivityTaskRequest

一旦收到任務，我們通過調用任務的getTaskToken方法檢索它的唯一標識符。

5. 接下來，編寫一些代碼來處理進來的任務。在輪詢任務的代碼之後，將以下內容添加到您的main方法中，並檢索其任務令牌。

```
    if (task_token != null) {
        String result = null;
        Throwable error = null;

        try {
            System.out.println("Executing the activity task with input '" +
                    task.getInput() + "'.");
            result = sayHello(task.getInput());
        } catch (Throwable th) {
            error = th;
        }

        if (error == null) {
            System.out.println("The activity task succeeded with result '"
                    + result + "'.");
            swf.respondActivityTaskCompleted(
                new RespondActivityTaskCompletedRequest()
                    .withTaskToken(task_token)
                    .withResult(result));
        } else {
            System.out.println("The activity task failed with the error '"
                    + error.getClass().getSimpleName() + "'.");
```

```
            swf.respondActivityTaskFailed(
                new RespondActivityTaskFailedRequest()
                    .withTaskToken(task_token)
                    .withReason(error.getClass().getSimpleName())
                    .withDetails(error.getMessage()));
        }
    }
```

如果任務令牌不是null，那麼我們就可以開始運行 activity 方法（ sayHello ），為它提供與任務一起發送的輸入數據。

如果工作成功 (未產生錯誤)，則 Worker 會以包含工作 Token 和活動結果資料的RespondActivityTaskCompletedRequest物件呼叫 SWF 用戶端的respondActivityTaskCompleted方法來回應 SWF。

另一方面，如果任務失敗，那麼我們通過調用帶有RespondActivityTaskFailedRequest對象的respondActivityTaskFailed方法來響應，將任務令牌和有關錯誤的信息傳遞給它。

> **Note**
>
> 如果死亡，此活動將不會正常關閉。雖然它超出了本教學課程的範圍，但是在隨附的主題「正常關閉活動」與「工作流程工作者」中提供了此活動 Worker 的替代實作。

實作工作流程工作者

您的工作流程邏輯位於稱為工作流程 Worker 的一段程式碼中。工作流程 Worker 會輪詢由網域中傳送的決策任務，以及Amazon SWF在預設工作清單上，輪詢工作流程類型已註冊的決策任務。

當工作流\ 'b5\ '7b Worker 收到任務時，它會做出某種決定 (通常是否排程新活動) 並採取適當的動作 (例如排程活動)。

1. 打開文本編輯器並創建文件WorkflowWorker.java，添加包聲明並根據常見步驟導入。
2. 添加一些其他導入到文件中：

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;
import java.util.ArrayList;
```

```
import java.util.List;
import java.util.UUID;
```

3. 宣告WorkflowWorker類別，並建立用來存取 SWF 方法之AmazonSimpleWorkflowClient類別的實體。

```
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
```

4. 添加方main法。此方法會持續迴圈，並使用 SWF 用戶端的pollForDecisionTask方法輪詢決策工作。提PollForDecisionTaskRequest供詳細資訊。

```
    PollForDecisionTaskRequest task_request =
        new PollForDecisionTaskRequest()
            .withDomain(HelloTypes.DOMAIN)
            .withTaskList(new TaskList().withName(HelloTypes.TASKLIST));

    while (true) {
        System.out.println(
                "Polling for a decision task from the tasklist '" +
                HelloTypes.TASKLIST + "' in the domain '" +
                HelloTypes.DOMAIN + "'.");

        DecisionTask task = swf.pollForDecisionTask(task_request);

        String taskToken = task.getTaskToken();
        if (taskToken != null) {
            try {
                executeDecisionTask(taskToken, task.getEvents());
            } catch (Throwable th) {
                th.printStackTrace();
            }
        }
    }
```

一旦接收到任務，我們調用它的getTaskToken方法，該方法返回一個字符串，可用於識別任務。如果返回的令牌不是null，那麼我們在executeDecisionTask方法中進一步處理它，將任務令牌和隨任務一起發送的HistoryEvent對象列表傳遞給它。

5. 添加方executeDecisionTask法，取任務令牌（aString）和HistoryEvent列表。

```
    List<Decision> decisions = new ArrayList<Decision>();
    String workflow_input = null;
    int scheduled_activities = 0;
    int open_activities = 0;
    boolean activity_completed = false;
    String result = null;
```

我們還設置了一些數據成員來跟踪事情，例如：

- 用來報告處理任務結果的決策物件清單。
- 保存 "WorkflowExecutionStarted" 事件所提供之工作流程輸入的字串
- 已排程和開啟 (執行中) 活動的計數，以避免在已排程或目前正在執行時排程相同的活動。
- 一個布爾值，表示活動已完成。
- 一個字符串來保存活動結果，用於返回它作為我們的工作流結果。

6. 接下來，根據getEventType方法executeDecisionTask報告的事HistoryEvent件類型，新增一些程式碼來處理隨工作一起傳送的物件。

```
System.out.println("Executing the decision task for the history events: [");
for (HistoryEvent event : events) {
    System.out.println("  " + event);
    switch(event.getEventType()) {
        case "WorkflowExecutionStarted":
            workflow_input =
                event.getWorkflowExecutionStartedEventAttributes()
                     .getInput();
            break;
        case "ActivityTaskScheduled":
            scheduled_activities++;
            break;
        case "ScheduleActivityTaskFailed":
            scheduled_activities--;
            break;
        case "ActivityTaskStarted":
            scheduled_activities--;
            open_activities++;
            break;
        case "ActivityTaskCompleted":
            open_activities--;
            activity_completed = true;
            result = event.getActivityTaskCompletedEventAttributes()
```

```
                            .getResult();
            break;
        case "ActivityTaskFailed":
            open_activities--;
            break;
        case "ActivityTaskTimedOut":
            open_activities--;
            break;
    }
}
System.out.println("]");
```

為了我們的工作流程，我們最感興趣的是：

- "WorkflowExecutionStarted" 事件，表示工作流程執行已開始 (通常意味著您應該執行工作流程中的第一個活動)，並提供提供給工作流程的初始輸入。在這種情況下，它是我們問候語的名稱部分，因此它保存在字符串中以便在調度活動運行時使用。
- 「ActivityTaskCompleted" 事件，在排定的活動完成後傳送。事件資料也包括已完成活動的傳回值。由於我們只有一個活動，我們將使用該值作為整個工作流程的結果。

如果您的工作流程需要其他事件類型，則可以使用它們。如需每個事件類型的相關資訊，請參閱類HistoryEvent別說明。

+ 注意：switch語句中的字符串是在 Java 7 中引入的。如果您使用的是早期版本的 Java，則可以使用該EventType類將String返回的返回值轉換為枚舉值，然後在必String要時返回：history_event.getType()

```
EventType et = EventType.fromValue(event.getEventType());
```

1. 在switch陳述式之後，新增更多程式碼，以根據收到的工作做出適當的決定做出回應。

```
if (activity_completed) {
    decisions.add(
        new Decision()
            .withDecisionType(DecisionType.CompleteWorkflowExecution)
            .withCompleteWorkflowExecutionDecisionAttributes(
                new CompleteWorkflowExecutionDecisionAttributes()
                    .withResult(result)));
} else {
    if (open_activities == 0 && scheduled_activities == 0) {
```

```
        ScheduleActivityTaskDecisionAttributes attrs =
            new ScheduleActivityTaskDecisionAttributes()
                .withActivityType(new ActivityType()
                    .withName(HelloTypes.ACTIVITY)
                    .withVersion(HelloTypes.ACTIVITY_VERSION))
                .withActivityId(UUID.randomUUID().toString())
                .withInput(workflow_input);

        decisions.add(
                new Decision()
                    .withDecisionType(DecisionType.ScheduleActivityTask)
                    .withScheduleActivityTaskDecisionAttributes(attrs));
    } else {
        // an instance of HelloActivity is already scheduled or running. Do nothing,
 another
        // task will be scheduled once the activity completes, fails or times out
    }
}

System.out.println("Exiting the decision task with the decisions " + decisions);
```

- 如果尚未排程活動，我們會以決定做出回應，該ScheduleActivityTask決定會在ScheduleActivityTaskDecisionAttributes結構中提供下一個Amazon SWF應排程之活動的相關資訊，也包括Amazon SWF應傳送至活動的任何資料。
- 如果活動已完成，則我們會考慮整個工作流程已完成，並以CompletedWorkflowExecution決定做出回應，填入CompleteWorkflowExecutionDecisionAttributes結構以提供有關已完成工作流程的詳細資訊。在此情況下，我們將返回活動的結果。

在任何一種情況下，決策資訊都會新增至在方法頂端宣告的Decision清單中。

2. 透過傳回處理任務時收集的Decision物件清單來完成決策任務。在我們一直在編寫的executeDecisionTask方法的末尾添加以下代碼：

```
swf.respondDecisionTaskCompleted(
    new RespondDecisionTaskCompletedRequest()
        .withTaskToken(taskToken)
        .withDecisions(decisions));
```

SWF 用戶端的respondDecisionTaskCompleted方法會使用可識別工作的工作權杖，以及Decision物件清單。

實作工作流程啟動器

最後，我們將撰寫一些程式碼來啟動工作流程執行。

1. 打開文本編輯器並創建文件WorkflowStarter.java，添加包聲明並根據常見步驟導入。
2. 添加WorkflowStarter類：

```
package aws.example.helloswf;


import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.*;

public class WorkflowStarter {
    private static final AmazonSimpleWorkflow swf =

 AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
    public static final String WORKFLOW_EXECUTION = "HelloWorldWorkflowExecution";

    public static void main(String[] args) {
        String workflow_input = "{SWF}";
        if (args.length > 0) {
            workflow_input = args[0];
        }

        System.out.println("Starting the workflow execution '" + WORKFLOW_EXECUTION +
                "' with input '" + workflow_input + "'.");

        WorkflowType wf_type = new WorkflowType()
            .withName(HelloTypes.WORKFLOW)
            .withVersion(HelloTypes.WORKFLOW_VERSION);

        Run run = swf.startWorkflowExecution(new StartWorkflowExecutionRequest()
            .withDomain(HelloTypes.DOMAIN)
            .withWorkflowType(wf_type)
            .withWorkflowId(WORKFLOW_EXECUTION)
            .withInput(workflow_input)
            .withExecutionStartToCloseTimeout("90"));

        System.out.println("Workflow execution started with the run id '" +
                run.getRunId() + "'.");
```

```
    }
}
```

此WorkflowStarter類別由單一方法組成main，該方法需要在命令列上傳遞的選用引數作為工作流程的輸入資料。

SWF 用戶端方法會接受StartWorkflowExecutionRequest物件做為輸入。startWorkflowExecution在這裡，除了指定要執行的網域和工作流程類型之外，我們還提供：

- 人類可讀的工作流程執行名稱
- 工作流程輸入數據（在我們的示例中的命令行中提供）
- 逾時值，表示整個工作流程執行所需的時間長度 (以秒為單位)。

startWorkflowExecution傳回的 Run 物件會提供執行 ID，此值可用來識別工作流程執行記錄中Amazon SWF此特定工作流程執行的工作流程執行。

+ 注意：運行 ID 由生成Amazon SWF，並且與您在開始工作流執行時傳入的工作流執行名稱不同。

建置範例

要使用 Maven 構建示例項目，請轉到helloswf目錄並鍵入：

```
mvn package
```

結果helloswf-1.0.jar將在target目錄中生成。

執行範例

此範例包含四個獨立的可執行檔類別，這些類別彼此獨立執行。

> **Note**
>
> 如果您使用的是 Linux、macOS 或 Unix 系統，則可以在單一終端機視窗中一個接一個地執行所有系統。如果您正在執行 Windows，您應該開啟兩個額外的命令列執行個體，並瀏覽至每個執行個體中的helloswf目錄。

設定 Java 類別路徑

儘管 Maven 已經為您處理了依賴關係，但要運行該示例，您需要提供 AWS SDK 庫及其對 Java 類路徑的依賴關係。您可以將CLASSPATH環境變數設定為 AWS SDK 程式庫的位置，以及 SDK 中的third-party/lib目錄，其中包含必要的相依性：

```
export CLASSPATH='target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/
lib/*'
java example.swf.hello.HelloTypes
```

或者在運行每個應用程序時使用 **java** 命令的-cp選項設置類路徑。

```
java -cp target/helloswf-1.0.jar:/path/to/sdk/lib/*:/path/to/sdk/third-party/lib/* \
  example.swf.hello.HelloTypes
```

您使用的風格取決於您。如果您在構建代碼時沒有任何問題，請嘗試運行示例並獲得一系列「NoClassDefFound」錯誤，很可能是因為類路徑設置不正確。

註冊網域、工作流程和活動類型

在執行 Worker 和工作流程啟動器之前，您需要註冊網域以及工作流程和活動類型。執行此操作的程式碼已在註冊網域工作流程和活動類型中實作。

構建後，如果您已設置 CLASSPATH，則可以通過執行以下命令來運行註冊代碼：

```
    echo 'Supply the name of one of the example classes as an argument.'
```

啟動活動和工作流程工作者

現在類型已註冊，您可以啟動活動和工作流程 Worker。它們將繼續運行並輪詢任務，直到它們被殺死為止，因此您應該在單獨的終端窗口中運行它們，或者，如果您在 Linux，macOS 或 Unix 上運行，則可以使用&操作符使每個任務在運行時產生單獨的進程。

```
    echo 'If there are arguments to the class, put them in quotes after the class
 name.'
    exit 1
```

如果您在不同的視窗中執行這些命令，請省略每一行的 final 運&算子。

開始工作流程執行

現在您的活動和工作流程 Worker 正在輪詢，您可以開始工作流程執行。此程序將執行，直到工作流程傳回已完成狀態為止。你應該在一個新的終端窗口中運行它（除非你使用運&算符將你的工作者作為新的生成進程運行）。

```
fi
```

> **Note**
>
> 如果您想要提供自己的輸入資料，這些資料會先傳遞至工作流程，然後再傳遞至活動，請將其新增至命令列。例如：
>
> ```
> echo "## Running $className..."
> ```

一旦您開始執行工作流程，您應該開始看到 Worker 和工作流程執行本身所交付的輸出。當工作流程最終完成時，其輸出將列印到螢幕上。

此範例的完整來源

您可以在aws-java-developer-guide儲存庫中的 Github 上瀏覽此範例的完整原始碼。

如需詳細資訊

- 如果工作流程輪詢仍在進行中時關閉，此處顯示的 Worker 可能會導致工作遺失。若要瞭解如何正常關閉 Worker，請參閱正常關閉活動和工作流程工作程式。
- 若要進一步了解Amazon SWF，請造訪Amazon SWF首頁或檢視Amazon SWF開發人員指南。
- 您可以使用 AWS Flow Framework for Java，使用註解以優雅的 Java 樣式撰寫更複雜的工作流程。AWS Flow Framework如需詳細資訊，請參閱 Java 開發人員指南。

Lambda 任務

作為一種替代辦法或與之結合,Amazon SWF活動，您可以使用Lambda函數來表示工作流中的工作單位，並將其安排與活動類似。

本主題重點介紹如何實現Amazon SWF Lambda任務AWS SDK for Java。如需有關 的詳細資訊Lambda任務的詳細信息，請參閱AWS Lambda工作中的Amazon SWF開發人員指南。

設置跨服務 IAM 角色以運行您的 Lambda 函數

之前Amazon SWF可以運行Lambda函數時，您需要設定 IAM 角色，以便Amazon SWF運行權限Lambda函式。如需如何執行此操作的完整信息，請參AWS Lambda工作。

當您註冊工作流程時，您需要此 IAM 角色的 Amazon Resource Name (ARN)Lambda任務。

建立 Lambda 函數

你可以寫Lambda函數在許多不同的語言中，包括 Java。有關如何創作、部署和使用Lambda函數的詳細資訊，請參AWS Lambda開發人員指南。

> **Note**
>
> 您使用什麼語言來寫入Lambda函數，它可以通過任何 Amazon SWF工作流程，而不管工作流程程式碼寫入的語言如何。Amazon SWF處理運行函數並將數據傳入函數的詳細信息。

這裏有一個簡單的Lambda函數，該函數可用於代替構建簡單Amazon SWF應用。

- 這個版本是用 JavaScript 編寫的，可以直接使用AWS Management Console：

```
exports.handler = function(event, context) {
    context.succeed("Hello, " + event.who + "!");
};
```

- 這是用 Java 編寫的相同函數，您也可以在 Lambda 上部署和運行它：

```
package example.swf.hellolambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.util.json.JSONException;
import com.amazonaws.util.json.JSONObject;

public class SwfHelloLambdaFunction implements RequestHandler<Object, Object> {
    @Override
    public Object handleRequest(Object input, Context context) {
        String who = "{SWF}";
        if (input != null) {
            JSONObject jso = null;
```

```
            try {
                jso = new JSONObject(input.toString());
                who = jso.getString("who");
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return ("Hello, " + who + "!");
    }
}
```

> **Note**
>
> 若要進一步了解將 Java 函式部署到 Lambda，請參建立部署套件 (Java)中的AWS Lambda開發人員指南。您還需要查看標題為創作的程式設計模型LambdaJava 中的函式。

Lambda函數採用事件或者輸入對象作為第一個參數，上下文對象作為第二個對象，它提供有關運行Lambda函數。這個特定的函數期望輸入在 JSON 中，並且who字段設置為用於創建問候語的名稱。

註冊工作流以便與 Lambda 一起使用

對於工作流來調度Lambda函數，則必須提供 IAM 角色名稱，以提供Amazon SWF具有許可能夠叫用Lambda函數。您可以在工作流程註冊過程中使用`withDefaultLambdaRole`或者`setDefaultLambdaRole`的方法RegisterWorkflowTypeRequest。

```
System.out.println("** Registering the workflow type '" + WORKFLOW + "-" +
 WORKFLOW_VERSION
        + "'.");
try {
    swf.registerWorkflowType(new RegisterWorkflowTypeRequest()
        .withDomain(DOMAIN)
        .withName(WORKFLOW)
        .withDefaultLambdaRole(lambda_role_arn)
        .withVersion(WORKFLOW_VERSION)
        .withDefaultChildPolicy(ChildPolicy.TERMINATE)
        .withDefaultTaskList(new TaskList().withName(TASKLIST))
        .withDefaultTaskStartToCloseTimeout("30"));
}
catch (TypeAlreadyExistsException e) {
```

排程Lambda任務

排程Lambda任務類似於調度活動。請您提供決策使用「計劃函數」DecisionType並使用ScheduleLambdaFunctionDecisionAttributes。

```
running_functions == 0 && scheduled_functions == 0) {
AWSLambda lam = AWSLambdaClientBuilder.defaultClient();
GetFunctionConfigurationResult function_config =
    lam.getFunctionConfiguration(
            new GetFunctionConfigurationRequest()
                .withFunctionName("HelloFunction"));
String function_arn = function_config.getFunctionArn();

ScheduleLambdaFunctionDecisionAttributes attrs =
    new ScheduleLambdaFunctionDecisionAttributes()
        .withId("HelloFunction (Lambda task example)")
        .withName(function_arn)
        .withInput(workflow_input);

decisions.add(
```

在 中ScheduleLambdaFuntionDecisionAttributes，則必須提供名稱的 ARN，這是Lambda函數，以及id，名稱為Amazon SWF將用於識別Lambda函數在歷史日誌中。

您也可以提供可選輸入(針對)Lambda函數並將其開始關閉超時值，這是Lambda函數在生成LambdaFunctionTimedOut事件。

> **Note**
>
> 此代碼使用瓦斯蘭布達克裏斯來檢索Lambda函數，給定函數名稱。您可以使用此技術來避免對完整的 ARN 進行硬編碼（其中包括AWS 帳戶ID）。

在決策者中處理 Lambda 函數事件

Lambda任務將生成許多事件，您可以在工作流工作人員中輪詢決策任務時對這些事件採取操作，這些事件與Lambda任務，使用事件類型值，例如LambdaFunctionScheduled、LambdaFunctionStarted，以及LambdaFunctionCompleted。如果Lambda函數失敗，或者運行時間比其設置的超時值更長，您將收到LambdaFunctionFailed或者LambdaFunctionTimedOut事件類型。

```
boolean function_completed = false;
String result = null;

System.out.println("Executing the decision task for the history events: [");
for (HistoryEvent event : events) {
    System.out.println("  " + event);
    EventType event_type = EventType.fromValue(event.getEventType());
    switch(event_type) {
    case WorkflowExecutionStarted:
        workflow_input =
            event.getWorkflowExecutionStartedEventAttributes()
                 .getInput();
        break;
    case LambdaFunctionScheduled:
        scheduled_functions++;
        break;
    case ScheduleLambdaFunctionFailed:
        scheduled_functions--;
        break;
    case LambdaFunctionStarted:
        scheduled_functions--;
        running_functions++;
        break;
    case LambdaFunctionCompleted:
        running_functions--;
        function_completed = true;
        result = event.getLambdaFunctionCompletedEventAttributes()
                      .getResult();
        break;
    case LambdaFunctionFailed:
        running_functions--;
        break;
    case LambdaFunctionTimedOut:
        running_functions--;
        break;
```

接收來自Lambda功能

當您收到LambdaFunctionCompleted`EventType, you can retrieve your 0 function’s return value by first calling `getLambdaFunctionCompletedEventAttributes在HistoryEvent 佈來獲

取LambdaFunctionCompletedEventAttributes對象，然後調用其getResult方法來檢索Lambda函數：

```
 LambdaFunctionCompleted:
running_functions--;
```

此示例的完整源

您可以瀏覽完整的源代碼:github: `<awsdocs/aws-java-developer-guide/tree/master/doc_source/snippets/helloswf_lambda/>在 Github 上查看這個例子aws-java 開發人員指南儲存庫。

正常關閉活動和工作流工作人員

所以此構建簡便Amazon SWF應用主題提供了由註冊申請、活動和工作流程工作人員以及工作流啟動器組成的簡單工作流應用程序的完整實現。

工作程序類被設計為連續運行，輪詢Amazon SWF以便開展活動或返回決定. 一旦提出輪詢請求，Amazon SWF記錄輪詢器並嘗試為其分配任務。

如果工作流工作人員在長時間輪詢期間終止，Amazon SWF可能仍然嘗試向已終止的工作人員發送任務，從而導致任務丟失（直到任務超時）。

處理這種情況的一種方法是在工作程序終止之前等待所有長輪詢請求返回。

在本主題中，我們將從helloswf，使用 Java 的關閉掛鉤嘗試優雅關閉活動工作人員。

以下是完整的代碼：

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflow;
import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClientBuilder;
import com.amazonaws.services.simpleworkflow.model.ActivityTask;
import com.amazonaws.services.simpleworkflow.model.PollForActivityTaskRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskCompletedRequest;
import com.amazonaws.services.simpleworkflow.model.RespondActivityTaskFailedRequest;
import com.amazonaws.services.simpleworkflow.model.TaskList;

public class ActivityWorkerWithGracefulShutdown {

    private static final AmazonSimpleWorkflow swf =
```

```
AmazonSimpleWorkflowClientBuilder.standard().withRegion(Regions.DEFAULT_REGION).build();
   private static final CountDownLatch waitForTermination = new CountDownLatch(1);
   private static volatile boolean terminate = false;

   private static String executeActivityTask(String input) throws Throwable {
       return "Hello, " + input + "!";
   }

   public static void main(String[] args) {
       Runtime.getRuntime().addShutdownHook(new Thread() {
           @Override
           public void run() {
               try {
                   terminate = true;
                   System.out.println("Waiting for the current poll request" +
                           " to return before shutting down.");
                   waitForTermination.await(60, TimeUnit.SECONDS);
               }
               catch (InterruptedException e) {
                   // ignore
               }
           }
       });
       try {
           pollAndExecute();
       }
       finally {
           waitForTermination.countDown();
       }
   }

   public static void pollAndExecute() {
       while (!terminate) {
           System.out.println("Polling for an activity task from the tasklist '"
                   + HelloTypes.TASKLIST + "' in the domain '" +
                   HelloTypes.DOMAIN + "'.");

           ActivityTask task = swf.pollForActivityTask(new
PollForActivityTaskRequest()
               .withDomain(HelloTypes.DOMAIN)
               .withTaskList(new TaskList().withName(HelloTypes.TASKLIST)));

           String taskToken = task.getTaskToken();
```

```
            if (taskToken != null) {
                String result = null;
                Throwable error = null;

                try {
                    System.out.println("Executing the activity task with input '"
                            + task.getInput() + "'.");
                    result = executeActivityTask(task.getInput());
                }
                catch (Throwable th) {
                    error = th;
                }

                if (error == null) {
                    System.out.println("The activity task succeeded with result '"
                            + result + "'.");
                    swf.respondActivityTaskCompleted(
                        new RespondActivityTaskCompletedRequest()
                            .withTaskToken(taskToken)
                            .withResult(result));
                }
                else {
                    System.out.println("The activity task failed with the error '"
                            + error.getClass().getSimpleName() + "'.");
                    swf.respondActivityTaskFailed(
                        new RespondActivityTaskFailedRequest()
                            .withTaskToken(taskToken)
                            .withReason(error.getClass().getSimpleName())
                            .withDetails(error.getMessage()));
                }
            }
        }
    }
}
```

在此版本中，main函數已被移動到自己的方法中，pollAndExecute。

所以此main函數現在使用CountDownLatch搭配關機掛接導致線程在請求終止後等待長達 60 秒，然後再讓線程關閉。

註冊網域

中的每個工作流和活動Amazon SWF需要域以運行。

1. 建立新RegisterDomainRequest對象，至少為其提供域名和工作流程執行保留期（這兩個參數都是必需的）。
2. 呼叫卓越亞馬遜簡單工作流程客户端註冊域方法與RegisterDomainRequest物件。
3. 抓住DomainAlreadyExistsException如果您請求的域已經存在（在這種情況下，通常不需要任何操作）。

下列程式碼示範此操作：

```
public void register_swf_domain(AmazonSimpleWorkflowClient swf, String name)
{
    RegisterDomainRequest request = new RegisterDomainRequest().withName(name);
    request.setWorkflowExecutionRetentionPeriodInDays("10");
    try
    {
        swf.registerDomain(request);
    }
    catch (DomainAlreadyExistsException e)
    {
        System.out.println("Domain already exists!");
    }
}
```

列出網域

您可以列出Amazon SWF與您的帳號關聯的網域和AWS區域按註冊類型。

1. 建立ListDomainsRequest對象，並指定您感興趣的域的註冊狀態-這是必需的。
2. Call卓越亞馬遜簡單工作流程客户端。列表域使用ListDomainRequest物件。結果在DomainInfos物件。
3. CallgetDomainInfos來取得DomainInfo物件。
4. CallgetName在每個DomainInfo對象來獲取它的名稱。

下列程式碼示範此操作：

```
public void list_swf_domains(AmazonSimpleWorkflowClient swf)
{
    ListDomainsRequest request = new ListDomainsRequest();
    request.setRegistrationStatus("REGISTERED");
    DomainInfos domains = swf.listDomains(request);
    System.out.println("Current Domains:");
    for (DomainInfo di : domains.getDomainInfos())
    {
        System.out.println(" * " + di.getName());
    }
}
```

SDK 附帶的代碼示例

所以此AWS SDK for Java附帶許多 SDK 功能的程式碼範例。您可以研究或修改這些內容來實現自己的AWS解決方案AWS SDK for Java。

如何獲取範例

所以此AWS SDK for Java代碼示例在樣本目錄。如果您使用設定AWS SDK for Java，則系統上已經有了樣本。

您也可以在AWS SDK for JavaGitHub 存儲庫中的src/Lext目錄。

使用命令行構建和運行示例

樣本包括螞蟻構建腳本，以便您可以從命令行輕鬆構建和運行它們。每個示例還包含 HTML 格式的 README 文件，其中包含特定於每個示例的信息。

> **Note**
>
> 如果您正在瀏覽 GitHub 上的示例代碼，請單擊Raw按鈕在查看示例的 README.html 文件時顯示。在原始模式下，HTML 將按照預期在瀏覽器中呈現。

先決條件

在運行任何AWS SDK for Java樣本，您需要設置AWS環境中的登入資料或使用AWS CLI，如設定AWS全權證書和區域促進發展。儘可能使用預設登入資料提供者鏈結。因此，通過這種方式設置您的憑據，您可以避免插入AWS源代碼目錄中的文件中的證書（可能會無意中簽入並公開共享）。

執行範例

1. 更改到包含示例代碼的目錄。例如，如果您位於AWSSDK 下載並希望運行AwsConsoleApp範例中，您可能輸入：

```
cd samples/AwsConsoleApp
```

2. 使用 Ant 建立並執行範例。默認構建目標執行兩個操作，因此您只需輸入：

```
ant
```

示例將信息打印到標準輸出，例如：

```
===========================================

Welcome to the {AWS} Java SDK!

===========================================
You have access to 4 Availability Zones.

You have 0 {EC2} instance(s) running.

You have 13 Amazon SimpleDB domain(s) containing a total of 62 items.

You have 23 {S3} bucket(s), containing 44 objects with a total size of 154767691 bytes.
```

使用 Eclipse IDE 生成和運行示例

如果您使用AWS Toolkit for Eclipse，您也可以在 Eclipse 中根據AWS SDK for Java或將開發工具包添加到現有 Java 項目中。

先決條件

安裝AWS Toolkit for Eclipse，我們建議使用您的安全證書配置工具包。您可以隨時選擇偏好設定來自視窗菜單，然後選擇 AWS工具箱部分。

執行範例

1. 開啟 Eclipse。

2. 建立新AWSJava 專案。在 Eclipse 中，File (檔案)菜單中，選擇新的，然後按一下專案。所以此新增專案精靈即會打開。
3. 展開 AWS 類別，然後選擇 AWSJava 專案。
4. 選擇 Next (下一步)。即會顯示專案設定頁面。
5. 在專案名稱方塊。所以此AWS SDK for Java示例組顯示 SDK 中可用的示例，如前所述。
6. 通過選中每個複選框來選擇要包含在項目中的樣本。
7. 輸入您的AWS登入資料。如果您已經配置了AWS Toolkit for Eclipse，則會自動填寫。
8. 選擇 Finish (完成)。該項目即被創建並添加到專案瀏覽器。
9. 選擇範例`.java`檔案。例如，對於Amazon S3樣本，選擇`S3Sample.java`。
10. 選擇執行來自執行菜單中選單。
11. 在專案瀏覽器，指向構建路徑，然後選擇添加資料庫。
12. 選擇 AWSJava 開發套件，選擇下一頁，然後按照屏幕上的其餘說明進行操作。

安全性 AWS SDK for Java

雲端安全是 Amazon Web Services (AWS) 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。安全是 AWS 與您之間共同承擔的責任。共同責任模型 將此描述為雲端本身的安全和雲端內部的安全。

雲的安全性 — AWS 負責保護運行 AWS 雲中提供的所有服務的基礎設施，並為您提供可以安全使用的服務。我們的安全責任是我們的首要任務 AWS，並且我們的安全性有效性是由第三方審計師定期測試和驗證，作為AWS 合規計劃的一部分。

雲端安全性 — 您的責任取決於您使用的 AWS 服務，以及其他因素，包括資料的敏感性、組織的需求，以及適用的法律和法規。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

主題

- AWS SDK for Java 1.x 中的資料保護
- AWS SDK for Java 對 TLS 的支援
- 身分和存取權管理
- 本 AWS 產品或服務的合規驗證
- 本 AWS 產品或服務的彈性
- 本 AWS 產品或服務的基礎架構安全性
- Amazon S3 加密用戶端移轉

AWS SDK for Java 1.x 中的資料保護

共同責任模式適用於本 AWS 產品或服務中的資料保護。如此模型所述， AWS 負責保護執行所有 AWS 雲端的全域基礎結構。您必須負責維護在此基礎設施上託管之內容的控制權。此內容包括您所使用 AWS 服務的安全組態和管理任務。如需有關資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需歐洲資料保護的相關資訊，請參閱 AWS 安全部落格上的AWS 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您使用 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 認證並設定個別使用者帳戶。如此一來，每個使用者都只會獲得授予完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源進行通訊。
- 使用設定 API 和使用者活動記錄 AWS CloudTrail。
- 使用 AWS 加密解決方案，以 AWS 及服務中的所有預設安全性控制。
- 使用 Amazon Macie 等進階受管安全服務，有助於探索和保護儲存在其中 Amazon S3的個人資料。
- 如果您在透過命令列介面或 API 存取時需要經 AWS 過 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需 FIPS 和 FIPS 端點的相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶帳戶號碼等敏感的識別資訊，放在自由格式的欄位中，例如名稱欄位。這包括當您使用主控台、API 或 AWS SDK 使用本 AWS 產品、 AWS 服務或其他服務時。 AWS CLI 您在此 AWS 產品或服務或其他服務中輸入的任何資料都可能會被拾取以包含在診斷記錄中。當您提供外部伺服器的 URL 時，請勿在驗證您對該伺服器請求的 URL 中包含登入資料資訊。

AWS SDK for Java 對 TLS 的支援

下列資訊僅適用於 Java SSL 實作 (中的預設 SSL 實作 AWS SDK for Java)。如果您使用不同的 SSL 實作，請參閱特定的 SSL 實作，以了解如何強制執行 TLS 版本。

如何檢查 TLS 版本

請參閱 Java 虛擬機器 (JVM) 提供者的說明文件，以判斷您的平台支援哪些 TLS 版本。對於某些 JVM，以下代碼將打印支持哪些 SSL 版本。

```
System.out.println(Arrays.toString(SSLContext.getDefault().getSupportedSSLParameters().getProto
```

要查看運作中的 SSL 交握和使用的 TLS 版本，您可以使用系統屬性 javax.net.debug。

```
java app.jar -Djavax.net.debug=ssl
```

> **Note**
>
> TLS 1.3 與適用於 Java 版本 1.9.5 至 1.10.31 的開發套件不相容。如需詳細資訊，請參閱下列部落格文章。

https://aws.amazon.com/blogs/developer/tls-1-3 -incompatibility-with-aws-sdk-一九五for-java-versions至一三一

強制執行最低 TLS 版本

SDK 始終偏好平台和服務支援的最新 TLS 版本。如果您希望強制執行特定的最低 TLS 版本，請參閱您的 JVM 文件。對於以 OpenJDK 為基礎的 JVM，您可以使用系統屬性。`jdk.tls.client.protocols`

```
java app.jar -Djdk.tls.client.protocols=PROTOCOLS
```

有關協議的支持值，請參閱 JVM 的文檔。

身分和存取權管理

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員控制哪些人可以通過身份驗證 (登入) 和授權 (具有權限) 來使用 AWS 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

主題

- 物件
- 使用身分驗證
- 使用政策管理存取權
- 如何 AWS 服務 使用 IAM
- 疑難排解 AWS 身分和存取

物件

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，具體取決於您在進行的工作 AWS。

服務使用者 — 如果您 AWS 服務 用於執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 AWS 功能來完成工作時，您可能需要其他權限。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取中的功能 AWS，請參閱疑難排解 AWS 身分和存取或 AWS 服務 您正在使用的使用指南。

服務管理員 — 如果您負責公司的 AWS 資源，您可能擁有完整的存取權 AWS。決定您的服務使用者應該存取哪些 AWS 功能和資源是您的工作。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何搭配使用 IAM AWS，請參閱 AWS 服務 您正在使用的使用者指南。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS存取權的詳細資訊。若要檢視可在 IAM 中使用的 AWS 基於身分識別的政策範例，請參閱 AWS 服務 您正在使用的使用者指南。

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。當您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。若要進一步了解，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 AWS中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

聯合身分

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時登入資料進行存取 AWS 服務 。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identi ty Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。 AWS Directory Service同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連線並同步到自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 和應用程式中使用。如需 IAM Identity Center 的相關資訊，請參閱《AWS IAM Identity Center 使用者指南》中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時性憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。若要進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 若要向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分供應商建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。

- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶的資源。角色是授予跨帳户存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取角色和資源型政策間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源類型政策的差異。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色：服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務。
  - 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色來管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內存放存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時性憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

若要了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的相關資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以連接到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。如需瞭解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些委託人 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 和 Amazon VPC 是支援 ACL 的服務範例。 AWS WAF若要進一步了解 ACL，請參閱《Amazon Simple Storage Service 開發人員指南》中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限：許可界限是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限的限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可邊界的相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可邊界。
- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需 Organizations 和 SCP 的相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策：工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合身分使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱《IAM 使用者指南》中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

如何 AWS 服務 使用 IAM

若要深入瞭解如何使 AWS 服務 用大多數 IAM 功能，請參閱 IAM 使用者指南中的與 IAM 搭配使用的 AWS 服務。

要了解如何將特定的 IAM AWS 服務 與 IAM 搭配使用，請參閱相關服務用戶指南的安全部分。

疑難排解 AWS 身分和存取

使用下列資訊可協助您診斷和修正使用和 IAM 時可能會遇到的 AWS 常見問題。

主題

- 我沒有執行操作的授權 AWS
- 我沒有授權執行 iam：PassRole

- 我想允許我以外的人訪 AWS 帳戶 問我的 AWS 資源

我沒有執行操作的授權 AWS

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 awes:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 awes:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我沒有授權執行 iam：PassRole

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS。

有些 AWS 服務 允許您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我想允許我以外的人訪 AWS 帳戶 問我的 AWS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您資源的許可。

如需進一步了解，請參閱以下內容：

- 若要瞭解是否 AWS 支援這些功能，請參閱如何 AWS 服務 使用 IAM。
- 若要了解如何提供您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中的另一個您擁有 AWS 帳戶 的 IAM 使用者提供存取權限。
- 若要了解如何將資源存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南中的提供第三方 AWS 帳戶 擁有的存取權。
- 若要了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策的差異。

本 AWS 產品或服務的合規驗證

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上架構 HIPAA 安全性與合規性 — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 用程式。

> **Note**
>
> 並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 中，保 AWS 服務 護指引並對應至安全控制的最佳實務。

- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub— 這 AWS 服務 提供了內部安全狀態的全面視圖 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

本 AWS 產品或服務的彈性

AWS 全球基礎架構是圍繞 AWS 區域 和可用區域建立的。

AWS 區域 提供多個實體分離和隔離的可用區域，這些區域透過低延遲、高輸送量和高度備援的網路連線。

透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需區域和可用區域的相關 AWS 資訊，請參閱AWS 全域基礎結構。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

本 AWS 產品或服務的基礎架構安全性

此 AWS 產品或服務使用受管理的服務，因此受到 AWS 全球網路安全性的保護。有關 AWS 安全服務以及如何 AWS 保護基礎結構的詳細資訊，請參閱AWS 雲端安全 若要使用基礎架構安全性的最佳做法來設計您的 AWS 環境，請參閱安全性支柱架構良 AWS 好的架構中的基礎結構保護。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取本「 AWS 產品」或「服務」。用戶端必須支援下列項目：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。

- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

本 AWS 產品或服務透過其支援的特定 Amazon 網路服務 (AWS) 服務，遵循共同的責任模式。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性說明文件頁面和符合性計劃 AWS 遵循工作範圍的 AWS 服務。

Amazon S3 加密用戶端移轉

本主題說明如何將應用程式從 () 加密用戶端的版本 1 Amazon Simple Storage Service (V1 Amazon S3) 移轉至第 2 版 (V2)，並確保整個移轉程序的應用程式可用性。

必要條件

Amazon S3 用戶端加密需要下列項目：

- Java 8 或更新版本已安裝在您的應用程式環境中。該 AWS SDK for Java 工作與甲骨文 Java SE 開發工具包和開放 Java 開發工具包（OpenJDK）Amazon Corretto，如紅帽 OpenJDK 和 JDK 的發行版。AdoptOpen
- 充氣城堡加密包。您可以將彈性城堡 .jar 文件放在應用程序環境的類路徑上，或者將對 Artifacldbcprov-ext-jdk15on（使用的 groupId Id）的依賴項添加到 Maven 文件中`org.bouncycastle`。`pom.xml`

移轉概觀

此遷移分兩個階段進行：

1. 更新現有用戶端以讀取新格式。更新您的應用程式以使用 1.11.837 或更新版本的應用程式， AWS SDK for Java 然後重新部署應用程式。這可讓應用程式中的用戶 Amazon S3 端加密服務用戶端解密 V2 服務用戶端所建立的物件。如果您的應用程式使用多個 AWS SDK，您必須個別更新每個 SDK。
2. 將加密和解密用戶端移轉至 V2。一旦所有 V1 加密用戶端都可以讀取 V2 加密格式，請更新應用程式程式碼中的用戶 Amazon S3 端加密和解密用戶端，以使用對等的 V2。

更新現有用戶端以讀取新格式

V2 加密用戶端使用舊版 AWS SDK for Java 不支援的加密演算法。

移轉的第一個步驟是將 V1 加密用戶端更新為使用 1.11.837 或更新版本的. AWS SDK for Java我們建議您更新至最新的發行版本，您可以在 Java API 參考版本 1.x 中找到。) 為此，請更新項目配置中的依賴項。更新專案組態後，請重建專案並重新部署。

完成這些步驟後，應用程式的 V1 加密用戶端將能夠讀取 V2 加密用戶端所寫入的物件。

更新項目配置中的依賴項

修改您的專案組態檔案 (例如 pom.xml 或建置 .gradle)，以使用. AWS SDK for Java然後，重建您的項目並重新部署它。

在部署新的應用程式程式碼之前完成此步驟，有助於確保整個叢集在移轉過程中保持一致的加密和解密作業。

使用 Maven 的示例

來自 pom.xml 文件的片段：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.837</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

示例使用搖籃

來自構建的 .gradle 文件的片段：

```
dependencies {
  implementation platform('com.amazonaws:aws-java-sdk-bom:1.11.837')
  implementation 'com.amazonaws:aws-java-sdk-s3'
}
```

將加密和解密用戶端移轉至 V2

一旦您的專案已更新為最新的 SDK 版本，您可以修改應用程式程式碼以使用 V2 用戶端。若要這麼做，請先更新您的程式碼以使用新的服務用戶端產生器。然後使用已重新命名的建置器上的方法提供加密材料，並視需要進一步設定您的服務用戶端。

這些程式碼片段示範如何搭配使用用戶端加密 AWS SDK for Java，並提供 V1 和 V2 加密用戶端之間的比較。

V1

```
// minimal configuration in V1; default CryptoMode.EncryptionOnly.
EncryptionMaterialsProvider encryptionMaterialsProvider = ...
AmazonS3Encryption encryptionClient = AmazonS3EncryptionClient.encryptionBuilder()
             .withEncryptionMaterials(encryptionMaterialsProvider)
             .build();
```

V2

```
// minimal configuration in V2; default CryptoMode.StrictAuthenticatedEncryption.
EncryptionMaterialsProvider encryptionMaterialsProvider = ...
AmazonS3EncryptionV2 encryptionClient = AmazonS3EncryptionClientV2.encryptionBuilder()
             .withEncryptionMaterialsProvider(encryptionMaterialsProvider)
             .withCryptoConfiguration(new CryptoConfigurationV2()
                           // The following setting allows the client to read V1
 encrypted objects
                           .withCryptoMode(CryptoMode.AuthenticatedEncryption)
             )
             .build();
```

上面的例子設置`cryptoMode`為`AuthenticatedEncryption`。這是一項設定，可讓 V2 加密用戶端讀取 V1 加密用戶端所寫入的物件。如果您的用戶端不需要讀取 V1 用戶端所寫入物件的功能，建議您`StrictAuthenticatedEncryption`改用預設設定。

構建 V2 加密客戶端

V2 加密客戶端可以通過調用亞馬遜 S3 EncryptionClient V2 加密生成器（ ）來構建。

您可以使用 V2 加密用戶端取代所有現有的 V1 加密用戶端。V2 加密用戶端將永遠能夠讀取 V1 加密用戶端所寫入的任何物件，只要您允許 V2 加密用戶端設定為使用 `AuthenticatedEncryption`cryptoMode。

建立新的 V2 加密用戶端與建立 V1 加密用戶端的方式非常相似。不過，有幾個差異：

- 您將使用對`CryptoConfigurationV2`象來配置客戶端而不是`CryptoConfiguration`對象。此為必要參數。
- V2 加密用戶端的預`cryptoMode`設設定為`StrictAuthenticatedEncryption`。對於 V1 加密客戶端，它是`EncryptionOnly`。
- 加密用戶端產生器上的方法 withEncryptionMaterials() 已重新命名為 withEncryptionMaterialsProvider ()。這僅僅是更準確地反映引數類型的外觀變更。設定服務用戶端時，必須使用新方法。

> **Note**
>
> 使用 AES-GCM 進行解密時，請在開始使用解密的資料之前將整個物件讀到最後。這是為了驗證該對象自加密以來沒有被修改。

使用加密材料提供者

您可以繼續使用與 V1 加密用戶端一起使用的相同加密材料提供者和加密材料物件。這些類別負責提供加密用戶端用來保護資料的金鑰。它們可以與 V2 和 V1 加密用戶端互換使用。

設定 V2 加密用戶端

V2 加密用戶端是使用`CryptoConfigurationV2`物件設定的。這個對象可以通過調用其默認構造函數，然後根據需要從默認值修改其屬性來構造。

的預設值`CryptoConfigurationV2`為：

- `cryptoMode = CryptoMode.StrictAuthenticatedEncryption`
- `storageMode = CryptoStorageMode.ObjectMetadata`
- `secureRandom=` 的執行個體 `SecureRandom`
- `rangeGetMode = CryptoRangeGetMode.DISABLED`
- `unsafeUndecryptableObjectPassthrough = false`

請注意 EncryptionOnly，V2 加密用戶端不支援`cryptoMode`。V2 加密用戶端將永遠使用經過驗證的加密來加密內容，並使用 V2 `KeyWrap` 物件保護內容加密金鑰 (CEK)。

下列範例示範如何在 V1 中指定加密組態，以及如何具現化 CryptoConfigurationV2 物件以傳遞給 V 2 加密用戶端產生器。

V1

```
CryptoConfiguration cryptoConfiguration = new CryptoConfiguration()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

V2

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.StrictAuthenticatedEncryption);
```

其他範例

下列範例示範如何解決從 V1 到 V2 的移轉相關的特定使用案例。

設定服務用戶端以讀取 V1 加密用戶端建立的物件

若要讀取先前使用 V1 加密用戶端寫入的物件，請將設定`cryptoMode`為`AuthenticatedEncryption`。下列程式碼片段示範如何使用此設定建構組態物件。

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
        .withCryptoMode(CryptoMode.AuthenticatedEncryption);
```

設定服務用戶端以取得物件的位元組範圍

若要能夠從加密的 S3 物件存取特定位元組範圍，請`get`啟用新的組態設定`rangeGetMode`。依預設，V2 加密用戶端上會停用此設定。請注意，即使啟用此範圍，範圍也`get`只適用於已使用用戶端`cryptoMode`設定支援的演算法加密的物件。如需詳細資訊，請參閱 AWS SDK for Java API 參考CryptoRangeGetMode中的。

如果您計劃使用 V2 Amazon S3 TransferManager 加密用戶端執行加密 Amazon S3 物件的分段下載，則必須先啟用 V2 加密用戶端上的`rangeGetMode`設定。

下列程式碼片段示範如何設定 V2 用戶端以執行範圍`get`。

```
// Allows range gets using AES/CTR, for V2 encrypted objects only
```

```
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withRangeGetMode(CryptoRangeGetMode.ALL);

// Allows range gets using AES/CTR and AES/CBC, for V1 and V2 objects
CryptoConfigurationV2 cryptoConfiguration = new CryptoConfigurationV2()
       .withCryptoMode(CryptoMode.AuthenticatedEncryption)
       .withRangeGetMode(CryptoRangeGetMode.ALL);
```

的開啟 PGP 金鑰 AWS SDK for Java

所有可公開使用的 Maven 構件AWS SDK for Java都會使用 OpenPGP 標準來簽署。下一節提供您驗證成品簽章所需的公開金鑰。

目前的金鑰

下表顯示目前發行版本的開發套件 (適用於 Java 1X) 和開發套件 (適用於 Java 2.x) 的 OpenPGP 金鑰資訊。

金鑰 ID	AC107B386692 D 添加
Type	RSA
大小	4096/4096
已建立	2016-06-30
到期	2024-10-08
使用者 ID	AWS開發套件與工具 < aws-dr-tools @amazon .com
鑰匙, 指紋	二月 9 209F 二樓三方 46 6484 1E55 交流 7B38 6692 日

要將 SDK for Java 的以下 OpenPGP 公鑰複製到剪貼板，請選擇右上角的「複製」圖標。

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

xsFNBFd1gAUBEACqbmmFbxdJgz1lD7wrlskQA1LLuSAC4p8ny9u/D2zLR8Ynk3Yz
mzJuQ+Kfjne2t+xTDex6MPJlMYpOviSWsX2psgvdmeyUpW9ap0lrThNYkc+W5fRc
buFehfbi9LSATZGJi8RG0sCCr5FsYVz0gEk85M2+PeM24cXhQIOZtQUjswX/pdk/
KduGtZASqNAYLKROmRODzUuaokLPo24pfm9bnr1RnRtwt5ktPAA5bM9ZZaGKriej
kT2lPffbBjp8F5AZvmGLtNm2Cmg4FKBvI04SQjy2jjrQ3wBzi5Lc9HTxDuHK/rtV
u6PewUe2WPlnxlXenhMZU1UK4YoSB9E9StQ2VxQiySLHSdxR7Ma4WgYdVLn9bOie
nj3QxLuQ1ZUKF79ES6JaM4tOz1gGcQeU1+UklgjFLuKwmzWRdEIFfxMyvH6qgKnd
U+DioH5mcUwhwffAAsuIJyAdMIEUYh7IfzJJXQf+fF+XfOCl6byOJFWrIGQkAzMu
```

```
CEvaCfwtHC2Lpzo33/WRFeMAuzzd0QJ4uz4xFFvaSOSZHMLHWI9YV/+Pea3X99Ms
0Nlek/LolAJh67MynHeVBOHKrq+fluorWepQivctzN6Y1NOkx5naTPGGaKWK7G2q
TbcY5SMnkIWfLFSougj0Fvmjczq8iZRwYxWA+i+LQvsR9WEXEiQffIWRoQARAQAB
zsFNBFd1gAUBEAC8zNArPWb3dPMThL2xAY+fS60vXdB1SkOtYJpDWpFgvo0d+VQ+
hV6XulGAHAS6xG1WHysPT9KejIRSgLG+e9CaM5yhsxNa1WFGUM4Q9ESo3t+a75Go
7xHIxgFjC046/O6Vh3g9N/PREeuG8zkZ3H2v5fmD+ejyPgk4W9sFL00zjRiZD0FK
VYR/j9uenEC/2NBcLuFy3q6cDfmCoDEOO62kXMnaGz3knzEK/X1SkcjsxRDq7zaQ
lQ1Kou+3dICwy4x5SJQ8jl+eeeEvF2C2/dXmDohb57tqUwioohMUQkmCtvZgEHjy
pUwgp0MTo25gWxkvJlSJKUOb6b1786WNySIzF2gxqlkkEmBl4RAssQkeXjrSmGws
MDyHNqyJeYFusl8sPaSpo+V2n0z+2B070Uq+wmf1S5A5FpegH0PZzzoNZo8I6Qxa
Zje9YSZUijGmZIdEBleRVt3Svhi8MYlnasd4bW2RK1sr7plkBf8QRe6biiQRF3KD
OSn5CbmXpAcHJ1ZHzRRdkXZDNQC6vCJxsy13O0TrhJtAV1Yq347uyUbVi291ISVg
roUVtprsmHoEk5GoOTHbg9SCSt+xi/FiJQC+ubWmIGXoFKMR3UmhDnnzobKcbnbs
/Hd981FdVghYYvq//gTAkJk0WxfGqO30wtXRndPOA0T+qhP3TE+LtGRJ+wARAQAB
wsFlBBgBCgAPBQJXdYAFAhsMBQkHhh+AAAoJEKwQezhmktrdTyEP/0HOVWHwQsaW
jMrGj0OOMFzxGUo8SBmYYTBs29VM8wBGDsPkYCjeZzU16i9iqDpDqxpyqmTigcjH
V8CDx/6xsMBLG2yKaKZ4m3+YnOQf/sQkyCvqiyMF9mS7pDYWy+mPhPuw8TDIfiqg
VhzjSpIMFWPqxVjn6KKbPN/QASr3Pf0cuP6qpHG+NAM6Q5dYkCebyvwzLmg1sVni
l6iSyJd1jBj3D34XrgWS9buyxBB2CjIM76WxfNViJ9zAaPI78X9v6PpDGn0kg6oL
zrusrvBjoZknKQm0SZ+41fx6xvrTPs8uPEzevzJBlkke6kw9+KagY8mrVX1ZenRg
+sY/4vxJreYWQeql67ggx+wFjKDcfhZA7m7OLHODysrGVCLcmuinUBaNlHmLDcGY
XZ+kMCoXfObpuCVByQmNJgEb47EIFlx/+TEeNHKMO+22xL1atFzXfkEVZck+NghL
ZyFDhS3g1bma7puU7r752uiJjA6Iv8+kHDXi+/V7GNpuiEFUYh69QQ2//CS5H51o
sC/Bkb9evSn/Lp8dMubtWAaXDGJMgw9vqZ55N02NK0fvF/IKHnGkvH28rv0OPCvO
WTA/MClv28y0PrSvcvMXnduLtkBEX7TISMPW+n+OTa63/z4YFfEZ7sFLrEm3Q3vJ
MN3mE5i3cw+JGXPSu0nTtgqk/oZv//SS
=Z9u3
-----END PGP PUBLIC KEY BLOCK-----
```

文件歷史記錄

本主題說明《AWS SDK for Java開發人員指南》歷程中的重要變更。

本文件建置於：2023 年 12 月 6 日

2024年1月12日

新增宣告終止 AWS SDK for Java v1.x 支援的橫幅。

2023 年 12 月 6 日

- 提供目前的 OpenPGP 金鑰。

2023 年 3 月 14 日

- 更新了指南以符合 IAM 最佳實務。如需更多詳細資訊，請參閱 IAM 中的安全最佳實務。

2022 年 7 月 28 日

- 新增 EC2-經典版將於 2022 年 8 月 15 日退休的警示。

二零一八年三月 22 日

- 已移除DynamoDB範例中管理 Tomcat 工作階段，因為不再支援該工具。

二 ○ 一七年十一月二

- 新增加加密用戶端的Amazon S3密碼編譯範例，包括新主題：使用用戶Amazon S3端加密和AWSKMS 受管金鑰的用戶Amazon S3端加密，以及使用用戶Amazon S3端主金鑰進行用戶端加密。

二零一七年四月十四

- 對 AWS SDK for Java「Amazon S3範例使用」區段進行了一些更新，包括新主題：管理值區和物件的Amazon S3存取權限以及將Amazon S3值區設定為網站。

二零一七年四月四日

- 新主題「啟用指標」AWS SDK for Java 說明如何產生的應用程式和 SDK 效能指標AWS SDK for Java。

二零一七年四月三日

- 在「 CloudWatch 範例使用AWS SDK for Java」區段中新增CloudWatch 範例：取得量度來源CloudWatch、發佈自訂指標資料、使用 CloudWatch 警示 CloudWatch、在中使用警示動作以及將事件傳送至 CloudWatch

2017 年 3 月 27 日

- 新增更多Amazon EC2範例至使用以下AWS SDK for Java部分的Amazon EC2範例：管理Amazon EC2執行個體、在中使用彈性 IP 位址Amazon EC2、使用區域和可用區域、使用Amazon EC2金鑰配對以及使用中的安全群組Amazon EC2。

2017 年 3 月 21 日

- 在 IAM 範例中新增一組新的 IAM 範例使用以下AWS SDK for Java部分：管理 IAM 存取金鑰、管理 IAM 使用者、使用 IAM 帳戶別名、使用 IAM 政策以及使用 IAM 伺服器憑證

二 ○ 一七年三月 13 日

- 將三個新主題新增至Amazon SQS區段：啟用Amazon SQS訊息佇列的長輪詢、在中設定可見性逾時以及在中Amazon SQS使用無效信件佇列Amazon SQS。

2017 年一月二十六日

- 已新增Amazon S3主題「用 TransferManager 於Amazon S3作業」，以及「使用」一AWS SDK for Java節中的AWS SDK for Java主題的新AWS開發最佳作法。

二零一七年一月六日

- 新增主Amazon S3題：使用儲存Amazon S3貯體原則管理值區的存取權，以及兩個新Amazon SQS主題：使用Amazon SQS訊息佇列和傳送接收和刪除Amazon SQS郵件。

二零一六年十二月六

- 已新增以下項目的範例主題DynamoDB：在中使用表格DynamoDB和使用中的項目DynamoDB。

2016 年 9 月 26 日

- [進階] 區段中的主題已移至 [使用]AWS SDK for Java，因為它們確實是使用 SDK 的核心。

2016 年 8 月 25 日

- 新主題 「建立服務用戶端」已新增至使用AWS SDK for Java，其中示範如何使用用戶端建置工具來簡化用AWS 服務戶端的建立作業。

  [程AWS SDK for Java式碼範例] 區段已更新為 S3 的新範例，這些範例由包含完整範例程式碼GitHub的儲存庫提供支援。

二 ○ 一六年五月二日

- 新主題「非同步程式設計」已新增至 「使用」一AWS SDK for Java節，說明如何使用傳回`Future`物件或採用`AsyncHandler`.

2016 年 4 月 26 日

- SSL 憑證需求主題已移除，因為它不再相關。對 SHA-1 簽署憑證的 Support 已於 2015 年棄用，且已移除安裝測試指令碼的網站。

2016 年 3 月 14 日

- 在Amazon SWF本節中新增了一個新主題：Lambda Tasks，其中說明如何實作將Lambda函數作為任務呼叫的Amazon SWF工作流程，作為使用傳統Amazon SWF活動的替代方案。

二零一六年三月四日

- 「使用此AWS SDK for Java區段的Amazon SWF範例」已更新為新內容：
  - Amazon SWF基本概念-提供有關如何在專案中包含 SWF 的基本資訊。
  - 構建一個簡單的Amazon SWF應用程序-一個新的教程，為 Java 開發人員提供 step-by-step 指導新的Amazon SWF.
  - 正常關閉活動和工作流程工作程式-說明如何使用 Java 的並行類別優雅地關閉 Amazon SWF Worker 類別。

2016 年 2 月 23 日

- AWS SDK for Java開發人員指南的來源已移至aws-java-developer-guide。

十二月二十八日

- 針對 DNS 名稱查閱設定 JVM TTL 已從「進階」移至 「使用」AWS SDK for Java，且已重新撰寫以便清楚。

  將 SDK 與 Apache Maven 搭配使用已更新，其中包含有關如何在專案中包含 SDK 的材料清單 (BOM) 的資訊。

二零一五年八月四日

- 「SSL 憑證需求」是「開始使用」一節中的新主題，說明 AWS「移至 SHA256 簽署的 SSL 憑證」以進行 SSL 連線，以及如何修正早期 1.6 和之前的 Java 環境以使用這些憑證，這些憑證在 2015 年 9 月 30 日之後才能AWS存取。

  > **Note**
  >
  > Java 1.7+ 已經能夠使用 SHA256 簽名的證書。

2014 年 5 月 14 日

- 介紹和入門材料已經過大量修訂，以支援新的指南結構，現在包含有關如何設定AWS認證和開發區域的指引。

  程式碼範例的討論已移至「其他文件與資源」區段中的本身主題中。

  有關如何查看 SDK 修訂歷史記錄的信息已轉移到簡介中。

2014 年 5 月 9 日

- AWS SDK for Java文件的整體結構已簡化，而且「入門」和「其他文件與資源」主題也已更新。

  已新增新主題：

  - 使用AWS證明資料-討論您可以指定證明資料的各種方式以搭配使用AWS SDK for Java。
  - 使用 IAM 角色授與AWS資源的存取權 Amazon EC2-提供有關如何為 EC2 執行個體上執行的應用程式安全地指定登入資料的資訊。

二○一三年九月九日

- 本主題「文件記錄」會追蹤AWS SDK for Java開發人員指南的變更。其為版本備註歷史記錄的相關文件。