

SpringBoot 微服務 2024

— 段維瀚老師 —

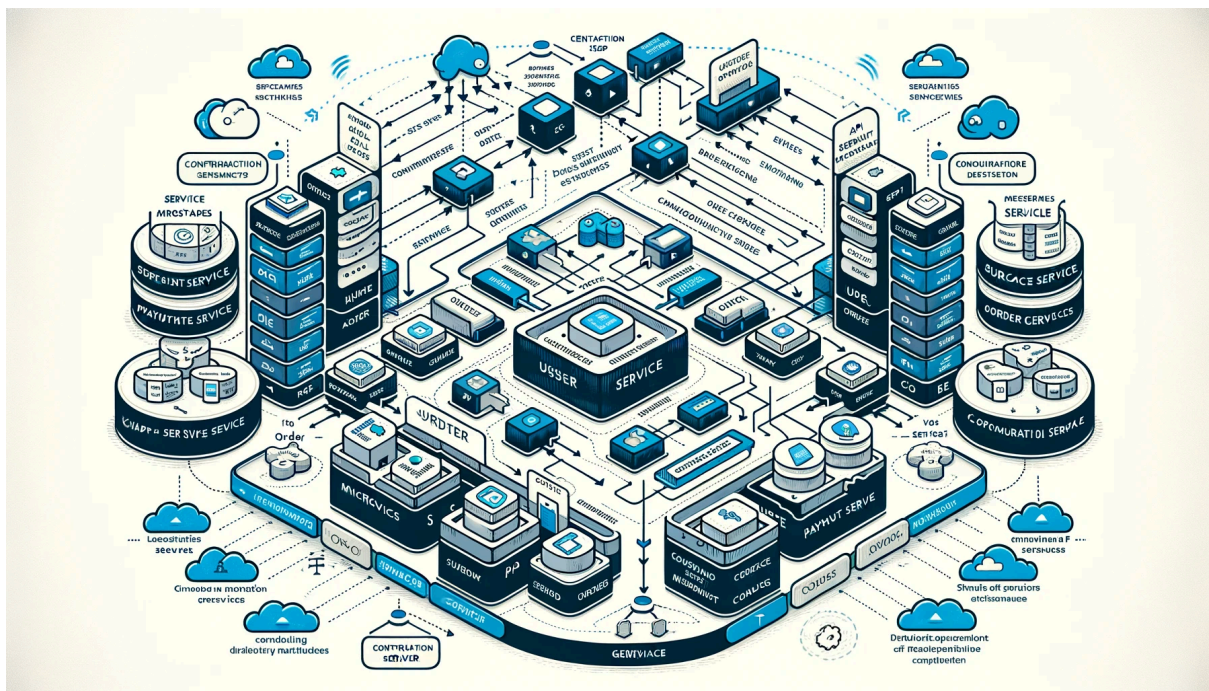
微服務的由來

微服務 (Microservices) 架構風格是一種將應用程式作為一組小的服務來開發的方法，每個服務運行在其獨立的進程中，通常以輕量級機制（通常是 **HTTP** 資源 API）相互通信。這些服務圍繞業務能力構建，可通過完全自動化的部署機制**獨立部署**。這些**小型服務**使得應用程式易於理解、開發和測試，並且可以更快地部署到生產環境中。

微服務的概念並非一夜之間出現的，而是隨著時間演進，從過去的軟體開發實踐中逐漸發展而來。早在2000年代初，企業就開始採用服務導向架構 (SOA) 來拆分大型的、單一架構的應用程式，但SOA通常關注於企業級的服務共享和重用。

微服務作為一個術語和概念，是在2011年至2012年左右開始獲得關注的。2012年，一群軟體工程師在威尼斯舉辦的一次研討會上討論了這種服務導向架構的**細化**形式，隨後這種架構風格被稱為“微服務”。James Lewis和Martin Fowler在2014年撰寫的一篇關鍵文章中對微服務架構進行了闡述，這篇文章幫助將微服務的概念推向了更廣泛的受眾。

微服務架構的採用被許多技術先驅公司所推動，如**Netflix**、Amazon、eBay和Twitter等，它們將系統從單體架構轉變為更加靈活、可擴展的微服務架構，從而提高了開發效率和系統的可靠性。這些成功案例進一步推動了微服務架構的普及和採用。



SpringBoot 微服務 2024

— 段維瀚老師 —

SOA(服務導向架構)與MicroServices(微服務)的異同

比較點	SOA(服務導向架構)	MicroServices(微服務)
架構風格	企業級集成服務	分散式、敏捷和可擴展的獨立服務
服務粒度	粗粒度服務	細粒度服務
通訊協議	SOAP, XML-RPC/JSON-RPC	偏好輕量級協議如HTTP/REST
數據管理	共享數據模型，集中式數據庫	分散式數據管理，每個服務管理自己的數據
部署	單體式，部署頻率較低	支持獨立部署，支持持續部署
技術依賴	服務間經常使用統一技術和標準	各服務可使用不同技術和語言
服務間通訊	重量級，經常依賴同步協議	輕量級，經常使用異步消息
可擴展性	可擴展，但由於集中式治理可能較複雜	因獨立服務和簡化的擴展而高度可擴展
開發方法	集中式，有共享治理和標準	分散式，跨功能團隊負責特定服務

微服務的本質

微服務的本質在於將大型複雜的軟件系統拆分成許多小型、鬆耦合的、可以獨立部署的服務單元。每個服務單元都圍繞著具體的業務功能構建，它們通過定義良好的輕量級通信機制(如 HTTP RESTful API)相互協作，共同構成整個系統的功能。微服務架構的核心原則和特點包括：

1 服務細粒度和獨立部署

每個微服務都是圍繞著特定的業務功能設計的，具有較小的責任範圍，並且可以獨立於其他服務進行部署和升級。

2 去中心化治理

微服務鼓勵使用適合各自服務業務需求的語言和框架，支持技術多樣性，不同的團隊可以根據服務的特定需求選擇最合適的技術棧。

SpringBoot 微服務 2024

— 段維瀚老師 —

3 去中心化數據管理

在微服務架構中，每個服務管理自己的數據庫，避免了不同服務間複雜的數據依賴關係，提高了數據的一致性和完整性。

4 基礎設施自動化

微服務架構傾向於自動化大部分操作任務，包括自動化測試、部署和擴展等，以支持持續集成和持續部署(CI/CD)。

5 容錯性和彈性設計

由於微服務可以獨立部署和擴展，因此它們能夠更好地處理單點故障和性能瓶頸，提高了系統的整體可靠性和可用性。

6 端點獨立運行與負責

微服務鼓勵跨職能團隊負責服務的完整生命週期，從設計、開發、部署到維護，團隊對自己的服務負有完全的責任。

7 輕量級通信

微服務之間通常通過HTTP REST、gRPC等輕量級通信協議進行交互，減少了通信成本，提高了系統的響應速度和靈活性。

微服務的本質在於通過服務的細粒度劃分和獨立部署來實現更快的迭代、更靈活的技術選型、更高的系統可靠性，以及更有效的團隊協作，最終促進快速、持續地交付價值。

SpringBoot 微服務 2024

— 段維瀚老師 —

台灣夜市：微服務架構的生動類比故事

-段維瀚老師分享

在一個充滿活力的台灣夜市裡，攤位林立，人聲鼎沸，各式各樣的小吃、遊戲和商品攤位猶如一個個獨立運作的微服務，共同構成了一個龐大而多元的夜市生態系統。

Eureka/Consul(服務註冊與發現)：

想像有一個中央資訊台，每個攤位在開市時都要到這裡報到，告訴資訊台自己的位置和提供的服務。遊客只需到資訊台一查，就能知道他們想尋找的攤位在哪裡。這就像是Eureka或Consul，幫助微服務互相發現和溝通。

Ribbon(客戶端負載均衡)：

當某個知名小吃攤前排起了長隊，一個機靈的助手會建議等待的顧客去其他類似的攤位，以分散人流。這就類似於Ribbon，它會幫助分散對某個服務的請求，避免過載。

Feign(聲明式HTTP客戶端)：

當你想要從一個遊戲攤位兌換獎品時，攤位老闆透過無線對講機直接向獎品攤位下訂單，這樣你就不需要自己走過去排隊。Feign在微服務架構中，就是這樣一種讓服務間通信更加簡便的工具。

Hystrix/Resilience4j(斷路器)：

如果某個遊戲攤位突然故障，工作人員會立即引導顧客到其他攤位，同時快速處理問題，防止長時間排隊等待造成的不滿。這類似於斷路器模式，當一個服務失效時，能夠快速進入保護狀態，並提供替代方案。

Gateway(API閘道)：

夜市的主入口就像是API Gateway，所有人都要通過這裡進入。它負責引導顧客到他們想去的地方，同時也是保安檢查的地點，保證夜市的秩序與安全。

Spring Cloud Config(配置中心)：

夜市管理辦公室有一個大型告示板，上面記錄了所有攤位的規則和價格信息。攤位如果有變動，只需更新這個告示板，所有攤位就能同步最新的規則。這就像是配置中心，統一管理微服務的配置信息。

SpringBoot 微服務 2024

— 段維瀚老師 —

Spring Cloud Stream(消息驅動):

攤位之間通過使用小紙條傳遞信息來互相合作完成一些大型訂單，比如一場生日派對的食物和裝飾。這就類似於Spring Cloud Stream，它通過輕量級的消息傳遞來連接各個微服務。

Spring Cloud Sleuth(分布式追蹤):

管理員透過一套系統來追蹤每個顧客的行動路徑，以改善夜市的佈局和服務。Spring Cloud Sleuth為微服務架構提供類似的功能，追蹤請求通過各個微服務的路徑。

OAuth(授權認證):

想要進入VIP區的遊客需要出示特定的身份證明才能進入。OAuth在微服務中負責類似的授權認證工作，保證資源的安全訪問。

Docker(容器化):

每個攤位都有自己的小推車或攤車，可以輕鬆地在夜市中移動和設置，就像Docker容器一樣，讓每個微服務都在獨立的環境中運行，易於部署和移動。

RabbitMQ(消息隊列):

當遊戲攤位太忙碌無法即時處理所有請求時，會使用號碼牌系統讓顧客等待叫號。這類似於RabbitMQ的消息隊列，暫存大量請求，按順序逐一處理。

透過這個充滿活力的台灣夜市故事，我們可以看到微服務架構中各種關鍵機制的生動類比，從而更加直觀地理解它們在實際應用中的作用和重要性。

SpringBoot 微服務 2024

— 段維瀚老師 —

Netflix 是什麼？



Netflix 是高畫質串流服務，透過數千種可連結網路的行動裝置，提供各種獲獎肯定的節目、電影、動畫、紀錄片等精彩內容。只要支付一筆經濟實惠的月費，就能隨時隨地，盡情觀賞。

Netflix 為何轉向微服務？

在2008年，Netflix經歷了一場轉折性的災難，當時它主要依賴DVD郵寄租賃服務，並剛開始探索充滿機遇與技術挑戰的線上串流市場。一個看似普通的工作日突然變成了一個關鍵時刻，當Netflix的一個主要數據中心發生嚴重故障，由於當時的架構過於集中，這一事件導致整個系統陷入癱瘓，數據庫服務器損壞，造成了數據不一致和服務中斷，持續了數天。

這次災難暴露了Netflix依賴單一巨型應用架構的脆弱性，這種架構下所有功能緊密耦合，任何小故障都可能引發整體崩潰。因此，Netflix的技術團隊決定尋找一種更可靠、靈活和可擴展的解決方案，他們將目光轉向了微服務架構。

微服務架構的採用就像是將一個大型綜合市場改造成熱鬧多彩的夜市，每個攤位都有自己的特色小吃，獨立運作卻相互配合。Netflix將其龐大的應用拆分成數以百計的小型服務，每個服務如同夜市中的一個攤位，負責一個具體的功能，比如用戶認證、影片推薦或影片播放等。

這種架構轉變帶來了諸多好處。首先，它提高了系統的可靠性，就像夜市中即便一個攤位遇到問題，其他攤位仍可繼續營業一樣，微服務架構下即便某個服務失效，整個系統仍能保持運行。其次，這使得Netflix能夠更快速地開發和部署新功能，每個微服務都可以獨立開發和更新。

SpringBoot 微服務 2024

— 段維瀚老師 —

，加速了創新的步伐。最後，微服務架構還提升了Netflix在全球用戶基礎擴展方面的能力，能夠根據用戶需求彈性調整特定服務。

為了支持這一轉型，Netflix選擇了AWS，因其高可靠性、可擴展性和全球基礎設施。借助AWS，Netflix得以快速應對數據中心故障，並支撐快速增長的在線業務。AWS的多樣化服務和工具也使得Netflix能夠有效管理和擴展其微服務架構，確保了高可用性和性能。

通過轉向微服務架構，Netflix不僅成功克服了2008年的危機，還奠定了成為全球領先在線影視串流服務提供商的基礎。微服務架構的採用，就像夜市中各具特色的攤位相互協作，為Netflix帶來了可靠性、靈活性和創新速度的提升，成為其成功故事中不可或缺的一部分。

在2008年遭遇故障之前，Netflix使用的是傳統的單體架構(Monolithic Architecture)。在這種架構中，Netflix的應用作為一個單一、龐大的系統運行，所有的功能和服務都緊密耦合在一起，共享相同的資源和數據庫。

SpringBoot 微服務 2024

— 段維瀚老師 —

單體架構的優點:

簡單性:

在初期，單體架構因其簡單性而受到青睞，特別是當應用規模較小，功能較少時，這種架構易於開發、測試、部署和管理。

一致性:

單體架構下，所有的服務和功能使用相同的技術棧和平台，這使得開發和管理過程中的一致性得以保持。

開發效率:

初始階段，由於所有功能都集中在一個代碼庫中，開發和維護的效率相對較高。

SpringBoot 微服務 2024

— 段維瀚老師 —

單體架構的缺點:

可擴展性差:

隨著應用規模的增長，單體應用變得越來越龐大和複雜，這使得添加新功能、維護和擴展變得困難。

部署風險:

在單體架構中，即使是很小的更改也需要重新部署整個應用，這增加了部署風險，並可能導致更長的停機時間。

技術債務:

隨著時間的推移，單體應用中的代碼可能變得難以理解和修改，積累了技術債務，從而阻礙了新功能的快速開發和迭代。

可靠性問題:

在單體架構中，一個小小的缺陷就有可能影響整個應用的穩定性和可用性，正如Netflix在2008年所經歷的那樣。

正因為單體架構的這些限制，尤其是在面對快速增長和全球擴展的需求時，Netflix最終決定轉向微服務架構，以提高系統的可靠性、靈活性和可擴展性。微服務架構通過將一個龐大的應用拆分成多個小型、獨立運作的服務來克服單體架構的這些缺點，每個服務負責應用中的一個特定功能，服務之間通過輕量級的通信機制協作，形成一個完整的系統。這種架構轉變使得Netflix能夠快速創新，更好地管理其全球用戶基礎，從而在後來的年份裡成為全球領先的在線影片流服務提供商。

SpringBoot 微服務 2024

— 段維瀚老師 —

微服務(Microservices)

微服務(Microservices)是一種軟體架構風格，它將一個應用程式拆分成一組小型、獨立的服務，每個服務運行在其自己的進程中，並通過輕量級的通訊機制(通常是HTTP RESTful API)相互協作。每個微服務專注於執行一個特定的業務功能，並且擁有自己的數據存儲方式，這樣它們就可以獨立於其他服務地被開發、部署、運營和擴展。

微服務的核心特點包括：

獨立性：每個微服務都是獨立運行的，具有自己的堆棧和依賴項，這使得它們可以獨立於其他服務進行開發和部署。這種獨立性也意味著服務之間的故障隔離，一個服務的故障不會直接影響到其他服務。

專一性：每個微服務都專注於一個特定的業務功能或處理一個特定的業務需求，使得它在設計和實現上都更為簡潔明確。

模塊化：微服務通過將大型應用拆分成小型、獨立的模塊來提高系統的模塊化程度，從而便於理解、開發和維護。

去中心化治理和數據管理：在微服務架構中，每個服務都可以使用最適合其業務需求的語言和數據存儲技術，這增加了技術的多樣性，但同時也提高了靈活性。

彈性和擴展性：微服務架構允許獨立擴展特定功能的服務，無需擴展整個應用，從而提供了更大的擴展性和資源利用效率。

持續交付和部署：微服務支持快速、自動化的部署，這使得持續集成和持續交付(CI/CD)變得更加容易實現，從而加快了新功能的推出速度和市場反應時間。

SpringBoot 微服務 2024

— 段維瀚老師 —

微服務的挑戰

儘管微服務架構帶來了許多好處，但它也帶來了一些挑戰，包括服務間通信的複雜性、數據一致性的維護、分佈式系統的複雜性管理、服務發現和負載均衡的需求，以及對於部署、監控和日誌管理等基礎設施支持的增加需求。

第1堂課：微服務架構基礎與Spring Boot入門

學習活動：

深入討論微服務架構的核心優勢及所面臨的挑戰，引導學員理解微服務的設計哲學與實踐意義。介紹Spring Boot框架的基本原理，包括其自動配置機制、起步依賴的概念以及Actuator的監控功能。

實戰練習：

指導學員動手搭建一個Spring Boot應用，從零開始實現一個提供歡迎信息的RESTful API。引導學員為應用添加健康檢查和基本的度量收集功能，加深對Spring Boot監控能力的理解。

預期成果：

學員將能夠獨立搭建並運行一個基於Spring Boot的微服務應用。

學員將掌握RESTful API的設計原則並能夠實現基本的API接口。

SpringBoot 微服務 2024

－ 段維瀚老師 －

第2堂課：Eureka - 服務註冊與發現

學習活動：

深入探討服務註冊與發現的重要性，並介紹Eureka和Consul這兩種服務註冊中心的基礎知識和配置方法。

透過實例引導學員理解如何在微服務架構中實現服務的自動註冊與發現。

實戰練習：

指導學員搭建一個Eureka Server服務，並將之前創建的「歡迎攤位」應用註冊至服務中心。

實現一個簡單的服務消費者應用，演示如何從Eureka中發現並調用「歡迎攤位」服務。

預期成果：

學員將能夠配置並運行Eureka/Consul服務註冊中心，並理解服務註冊與發現的工作原理。

學員將掌握如何在微服務架構下實現服務間的相互調用。

SpringBoot 微服務 2024

– 段維瀚老師 –

第3堂課: Feign - 聲明式HTTP客戶端

學習活動:

介紹Feign的聲明式HTTP調用方式, 強調其在簡化微服務間通信方面的優勢。

通過實例學習如何使用Feign來簡化服務間的HTTP請求。

OpenFeign 是一個宣告式的 web 服務客戶端, 它使得撰寫 HTTP 客戶端變得更加簡單。使用 OpenFeign, 你可以透過簡單的接口描述, 來定義服務綁定, 而不需要自己構建 HTTP 請求的細節。它是 Spring Cloud Netflix 的一部分, 但也可以獨立使用。

在微服務架構中, 服務之間的通信是一個重要的環節。OpenFeign 提供了一個簡潔的方法來呼叫 RESTful 服務。你只需創建一個接口並使用注解來配置需要呼叫的微服務的細節。

OpenFeign 會自動處理 URL 的組成、請求的發送以及回應的接收與反序列化。

特點:

易於使用: 只需透過接口和注解來定義服務綁定, 無需編寫實際的 HTTP 客戶端代碼。

靈活性: 支持自定義客戶端配置, 如超時設置、錯誤處理等。

整合性: 與 Spring 和 Spring Cloud 環境無縫整合, 支持負載平衡、服務發現等微服務常見需求。

自動錯誤處理: 可以透過自定義錯誤解碼器來處理遠程呼叫中的錯誤。

使用場景:

微服務之間的同步 HTTP 呼叫。

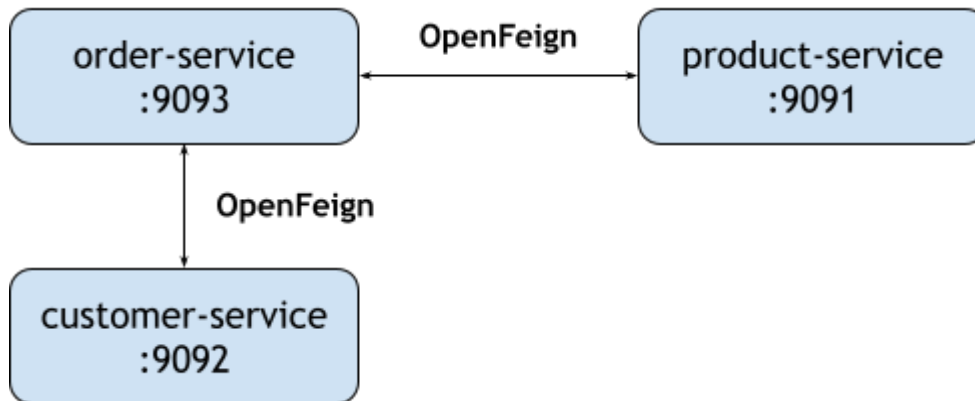
消費 RESTful Web 服務。

在微服務架構中進行服務間的通信。

SpringBoot 微服務 2024

— 段維瀚老師 —

實戰練習：



order-service作為客戶端，使用OpenFeign來調用product-service和customer-service中的API。箭頭表示調用的方向，從order-service指向其他兩個服務。這種架構允許order-service根據業務需求，獲取產品信息和客戶信息。

pom.xml 加入以下 **dependency**

```
<!-- Spring Cloud OpenFeign →
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Application 啟動類上加上 `@EnableFeignClients(basePackages = "your packages")`

例如: `@EnableFeignClients(basePackages = "com.example.demo.client")`

```
@SpringBootApplication
@EnableFeignClients(basePackages = "your packages")
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

SpringBoot 微服務 2024

－ 段維瀚老師 －

FeignClient 配置設定

```
package com.example.demo.client;
import org.springframework.cloud.openfeign.FeignClient;
...

@FeignClient(name = "product-service")
public interface ProductClient {
    // 取得指定商品
    @GetMapping("/products/{id}")
    Product getProductById(@PathVariable("id") Integer id);

    // 增加或減少商品庫存
    ...
}
```

預期成果：

學員將能夠使用Feign進行微服務間的聲明式HTTP調用，並理解其在微服務開發中的優勢。

SpringBoot 微服務 2024

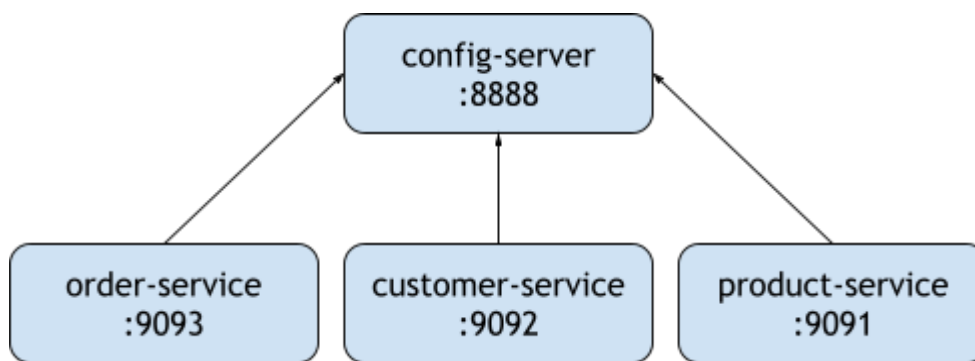
— 段維瀚老師 —

第4堂課: Config Server - 集中配置

學習活動:

微服務架構中的 Config Server 是一種集中式配置管理服務，用於管理微服務應用的所有配置信息。在分布式系統中，由於服務數量眾多，手動管理每個服務的配置既不現實也容易出錯。Config Server 提供了一個中心點來集中存儲和管理所有微服務的配置。

Config Server 通常會配合版本控制系統(如 Git)使用，將配置信息存儲於版本控制庫中，這樣就可以享有版本控制系統的所有好處，比如版本追蹤、變更審查等。當微服務啟動或運行時，它們會向 Config Server 請求所需的配置信息，Config Server 再從配置庫中讀取相應的配置信息並提供給請求的服務。



透過使用 Config Server, 可以達到以下幾個好處:

1. 集中管理配置: 所有的配置信息都集中存儲在一個地方, 易於管理和審計。
2. 動態更新配置: 修改配置信息後, 可以動態刷新服務的配置而無需重啟服務。
3. 環境隔離: 可以為不同的環境(如開發、測試、生產) 提供不同的配置, 並確保環境之間的隔離。

用配置檔名區分:

customer-service-dev.properties ← 開發環境

customer-service-test.properties ← 測試環境

customer-service.properties ← 生產環境

SpringBoot 微服務 2024

— 段維瀚老師 —

用資料夾區分：

/dev/customer-service.properties ← 開發環境

/test/customer-service.properties ← 測試環境

/config/customer-service.properties ← 生產環境

/shared/shared-service.properties ← 共用配置區

4. 安全性：敏感信息（如數據庫密碼）可以加密存儲，確保配置的安全。

在 Spring Cloud 生態系中，Spring Cloud Config 提供了實現 Config Server 的功能。要設置一個 Config Server，你需要在 Spring Boot 應用中引入 spring-cloud-config-server 依賴，並使用 @EnableConfigServer 注解啟用 Config Server 功能。然後，通過配置文件（如 application.properties）指定配置庫的位置（如 Git 庫的 URL），以及其他相關配置。

實戰練習

Config-Server 設定

pom.xml

```
<!-- Config Server -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

config-server 配置(本地)

```
# 配置 Local 文件存儲庫
spring.profiles.active=native
# 配置文件存儲庫地址 src/main/resources/config
spring.cloud.config.server.native.search-locations=classpath:/config
```

配置參考：

SpringBoot 微服務 2024

— 段維瀚老師 —

```
config-server-8888 [boot] [SpringBoot2024-
  src/main/java
    com.example.demo
      ConfigServer8888Application.java
  src/main/resources
    config
      customer-service.properties
      application.properties
```

config-server 配置(遠端)

```
# 配置 Remote 文件存儲庫
# 略過 SSL 驗證
spring.cloud.config.server.git.skip-ssl-validation=true
# 配置文件存儲庫地址
spring.cloud.config.server.git.uri=https://github.com/你的github名稱
/XXX-Spring-Cloud-Config-Server.git
# github 帳號
spring.cloud.config.server.git.username=你的github帳號
# github token
spring.cloud.config.server.git.password=你的github token
# 啟動時 clone 配置, 目的是為了避免啟動時無法獲取配置
spring.cloud.config.server.git.clone-on-start=true
```

配置參考:

Files

main

Go to file

customer-service.properties

Spring-Cloud-Config-Server / customer-service.properties

vincenttuan Update customer-service.properties

Code Blame 20 lines (14 loc) · 583 Bytes

```
1 # 應用端口
2 server.port=9092
3 # 應用名稱
4 spring.application.name=customer-service
5
6 eureka.instance.prefer-ip-address=true
7 #eureka.instance.ip-address=192.168.30.240
8 eureka.instance.ip-address=192.168.112.166
9
10 # Eureka 客戶端配置
11 eureka.client.service-url.defaultZone=http://${eureka.instance.ip-address}:8761/eureka/
12 eureka.instance.instance-id=${spring.application.name}:${eureka.instance.ip-address}:${server.port}
13
14 # base-path: 設置端點的基本路徑
15 management.endpoints.web.base-path=/actuator
16 # 要公開的端點
17 management.endpoints.web.exposure.include=*
```

SpringBoot 微服務 2024

– 段維瀚老師 –

Client 設定

以 customer-server 為例:

```
spring.config.import=optional:configserver:http://localhost:8888
```

```
spring.application.name=customer-service
```

```
# 注意! 在 Config Server 上的 properties 檔名必需要是
```

```
# {spring.application.name}.properties
```

```
# 例如: customer-service.properties
```

SpringBoot 微服務 2024

— 段維瀚老師 —

第5堂課: Gateway - API閘道

學習活動:

Spring Gateway 是一個基於 Spring Framework 的 API Gateway 解決方案, 它提供了路由、過濾和監控等功能, 以便於將請求路由到後端的微服務。透過 Spring Gateway, 可以實現諸如路徑重寫、請求限流、安全性控制等多種 API 網關常見的需求。

當 Spring Gateway 與 Spring Cloud LoadBalancer (lb) 結合使用時, 可以實現對後端微服務的負載均衡。Spring Cloud LoadBalancer 是一個客戶端負載均衡的解決方案, 它可以根據一定的策略(如輪詢、隨機、響應時間等)從一組服務實例中選擇一個實例來處理請求。

在微服務架構中, 通常會有多個實例提供相同的服務, 以實現高可用性和擴展性。當微服務設定 **server.port=0** 時, Spring Boot 會為應用自動配置一個可用的端口, 這樣做的好處是避免了端口衝突的問題, 尤其是在本地開發環境中運行多個服務實例時。

當這些微服務實例註冊到 Eureka 或其他服務註冊中心時, 它們的地址和端口會被記錄下來。Spring Gateway 會從服務註冊中心獲取這些服務實例的資訊, 並通過 LoadBalancer 自動選擇一個實例來轉發請求, 這樣就實現了負載均衡的功能。

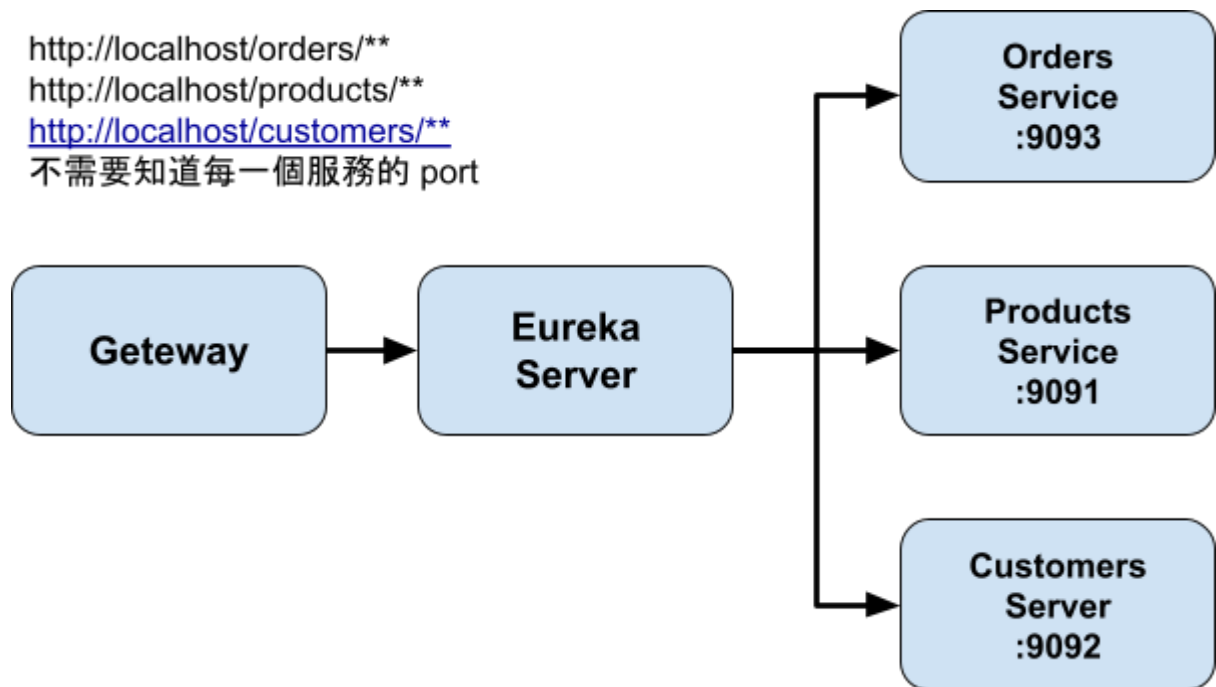
總的來說, Spring Gateway 結合 LoadBalancer 可以有效地管理和轉發對後端微服務的請求, 同時自動配置端口的特性使得運行多個微服務實例變得更加容易, 從而增強了應用的可擴展性和靈活性。

實戰練習:

指導學員搭建一個API Gateway, 作為的統一入口, 並實現對後端微服務的路由和代理。
進階實踐, 為Gateway添加動態路由和權限控制功能, 模擬不同使用者的訪問策略。

SpringBoot 微服務 2024

— 段維瀚老師 —



預期成果：

學員將能夠獨立搭建並配置API Gateway，並深入理解其在微服務架構中的關鍵作用。

學員將掌握API Gateway的進階功能，包括動態路由、過濾器 and 限流等，以滿足更複雜的業務需求。

SpringBoot 微服務 2024

— 段維瀚老師 —

第6堂課：Hystrix/Resilience4j - 斷路器模式

學習活動：

深入解析斷路器模式在微服務架構中的關鍵作用，以及如何使用Hystrix(已進入維護模式)或Resilience4j來實現服務的容錯與回退機制。

引導學員理解斷路器在保障微服務高可用性方面的重要性。

配置於服務層 (Service)：

通常，將斷路器配置在服務層是較為常見和推薦的做法。這是因為服務層通常包含了業務邏輯和對外部系統(如資料庫、其他微服務等)的調用。將斷路器配置在此層可以更有效地管理這些可能產生故障的調用，並提供降級和恢復策略。

這樣做的好處是將業務邏輯和故障保護邏輯(如斷路器)的關注點分離，使得系統的各個部分保持關注單一職責，從而提高了代碼的可維護性和可測試性。

配置於控制器層 (Controller)：

在某些情況下，將斷路器配置在控制器層也是可行的，特別是當你想要針對特定的端點或用戶請求操作提供故障保護時。這可以在請求進入業務邏輯之前提前處理可能的故障，並對用戶提供立即的回饋。

然而，將斷路器配置在控制器層可能會增加控制器的負擔，使其除了處理 HTTP 請求和響應之外，還需要處理斷路器相關的邏輯。

實戰練習：

指導學員為「碳烤攤位」服務添加Hystrix/Resilience4j斷路器，以保護對其他服務的調用不會因單點故障而影響整個系統。

通過模擬部分服務失效的情景，讓學員觀察並理解斷路器在實際應用中的保護效果。

SpringBoot 微服務 2024

— 段維瀚老師 —

預期成果：

學員將掌握斷路器模式的基本原理及其在微服務架構中的應用方法。

學員將能夠在微服務中實現容錯保護和服務降級，提高系統的韌性和可用性。

SpringBoot 微服務 2024

– 段維瀚老師 –

第7堂課: RabbitMQ - 訊息佇列

下載安裝與設定:

<https://www.rabbitmq.com/docs/install-windows>

Direct Downloads

Description	Download	Signature
Installer for Windows systems (from GitHub)	rabbitmq-server-3.13.1.exe.asc	Signature

下載: rabbitmq-server-3.13.1.exe.asc 並安裝

環境變數

在 path 加入 C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.1\sbin

開啟 **RabbitMQ Web Manager**

指令: rabbitmq-plugins enable rabbitmq_management

啟動 **RabbitMQ**

指令: rabbitmq-service start

關閉 **RabbitMQ**

指令: rabbitmq-service stop

查詢 **Java** 應用程式活動

指令: jps

結果:

49264 BootLanguageServerBootApplication

44548 XMLServerLauncher

47604 Jps

8060 SpringToolSuite4

刪除 某 **Java** 應用程式活動

指令: taskkill /PID 活動ID /F

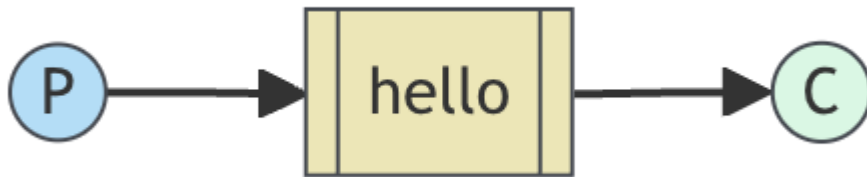
例如: taskkill /PID 49264 /F

SpringBoot 微服務 2024

— 段維瀚老師 —

學習活動：

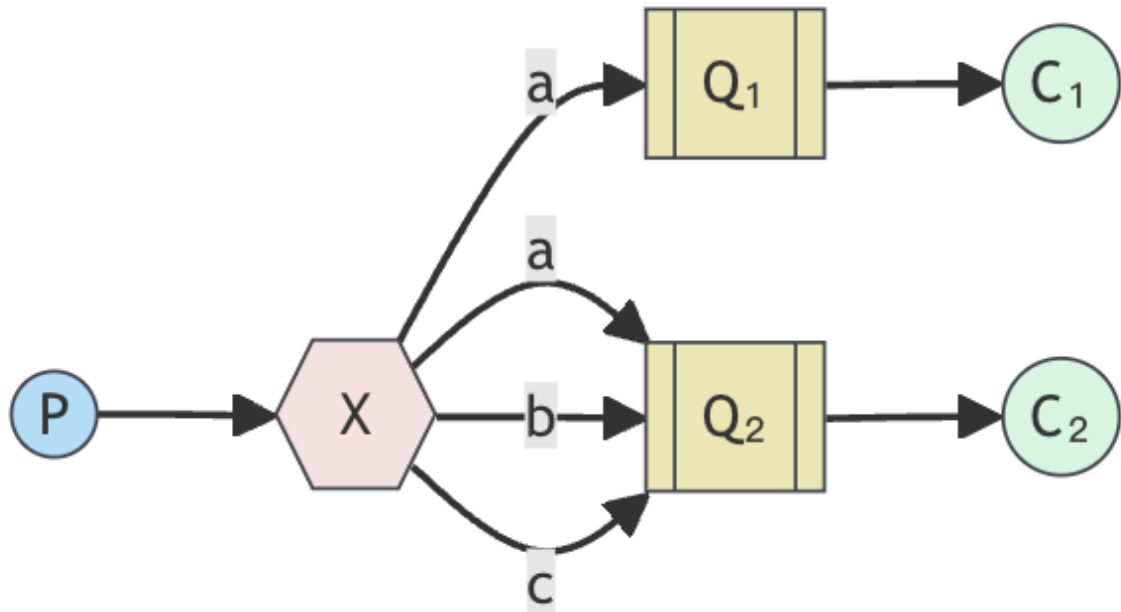
RabbitMQ 是一個基於高級消息隊列協議 (AMQP, Advanced Message Queuing Protocol) 的開源消息代理軟件。它的主要功能是在分佈式系統中介紹消息的生產者和消費者之間的通信，從而實現異步消息傳遞、系統解耦和負載均衡等特性。



SpringBoot 微服務 2024

— 段維瀚老師 —

運作原理：



RabbitMQ 的運作原理主要基於以下幾個核心概念：

1. 生產者 (**Producer**) : 生產者是發送消息的一方。在 RabbitMQ 中，生產者將消息發送到交換機 (Exchange)，而不是直接發送到隊列。
2. 交換機 (**Exchange**) : 交換機是接收生產者發送的消息並根據某種規則將其路由到一個或多個隊列的實體。交換機有幾種類型，包括直接 (direct)、主題 (topic)、扇出 (fanout) 和頭 (headers)，每種類型都有自己的路由策略。
3. 隊列 (**Queue**) : 隊列是消息最終被送達的地方，等待消費者來處理。每條消息都會被投遞到一個或多個隊列中，根據交換機指定的路由規則。
4. 消費者 (**Consumer**) : 消費者從隊列中取出消息並進行處理。消費者可以對一個或多個隊列進行監聽，當隊列中有新的消息時進行處理。
5. 綁定 (**Binding**) : 綁定是一種關聯，它定義了交換機如何根據路由鍵 (Routing Key) 將消息路由到隊列的規則。

SpringBoot 微服務 2024

— 段維瀚老師 —

通信協定：

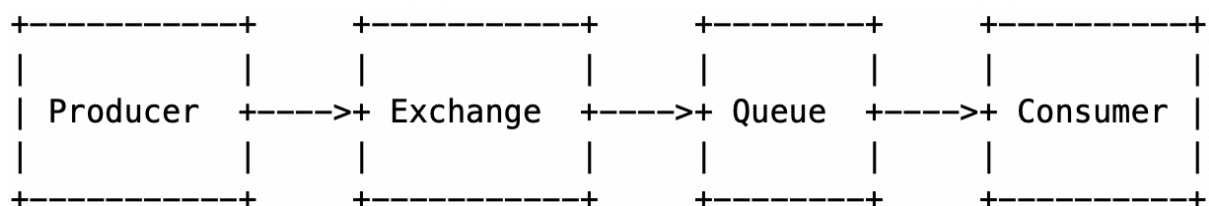
RabbitMQ 主要使用以下兩種協定進行通信：

1. 高級消息隊列協議(**AMQP**)：AMQP 是一個網絡協議，用於在分佈式或異質系統中進行異步消息通信。RabbitMQ 最初被設計為 AMQP 的一個實現，它支持 AMQP 的所有核心特性，包括消息隊列、路由（包括點對點和發布/訂閱）、事務和安全認證。
2. 消息隊列遙測傳輸協議(**MQTT**)和流式文本定向消息協議(**STOMP**)：除了 AMQP 之外，RabbitMQ 還支持 MQTT 和 STOMP 等其他消息協議，使其能夠滿足更多種類的應用需求，特別是在物聯網(IoT)和網頁應用中。

RabbitMQ 的這些特性和協議支持使其成為實現微服務架構中服務間異步通信的理想選擇。

通过使用 RabbitMQ，開發者可以建立一個高效、可擴展且鮮少耦合的系統架構。

這裡有一個簡化的流程圖，說明了 RabbitMQ 中消息如何從生產者經過交換機和隊列最終到達消費者的基本流程：



1. 生產者(**Producer**)：這是消息的發送方。生產者創建一條消息，並將其發送到一個交換機。
2. 交換機(**Exchange**)：交換機接收來自生產者的消息，然後根據特定的路由規則決定消息應該被發送到哪一個或哪些隊列。交換機的類型決定了這些路由規則的行為。
3. 隊列(**Queue**)：隊列是存儲消息的緩衝區，等待消費者來處理這些消息。
4. 消費者(**Consumer**)：消費者連接到隊列，並等待隊列中的消息。當消息到達時，消費者會進行處理。

在這個過程中，消息從生產者到達交換機，交換機根據設定的規則將消息路由到一個或多個隊列，最後由消費者從隊列中取出並處理消息。這整個流程支持消息的異步處理，允許生產者和消費者獨立運作，從而實現系統的解耦和擴展性。

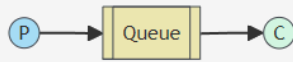
SpringBoot 微服務 2024

— 段維瀚老師 —

RabbitMQ 發送種類

1. "Hello World!"

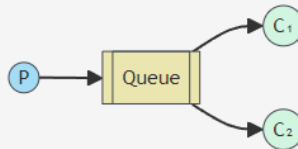
The simplest thing that does *something*



- Python
- Java
- Ruby
- PHP
- C#

2. Work Queues

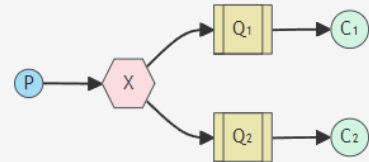
Distributing tasks among workers (the *competing consumers pattern*)



- Python
- Java

3. Publish/Subscribe

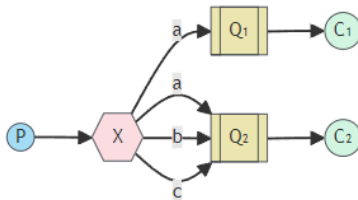
Sending messages to many consumers at once



- Python
- Java

4. Routing

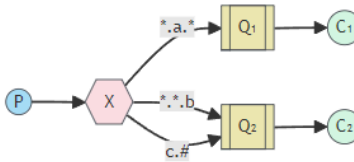
Receiving messages selectively



- Python
- Java

5. Topics

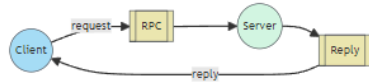
Receiving messages based on a pattern (topics)



- Python
- Java

6. RPC

Request/reply pattern example



- Python
- Java
- Ruby
- PHP
- C#

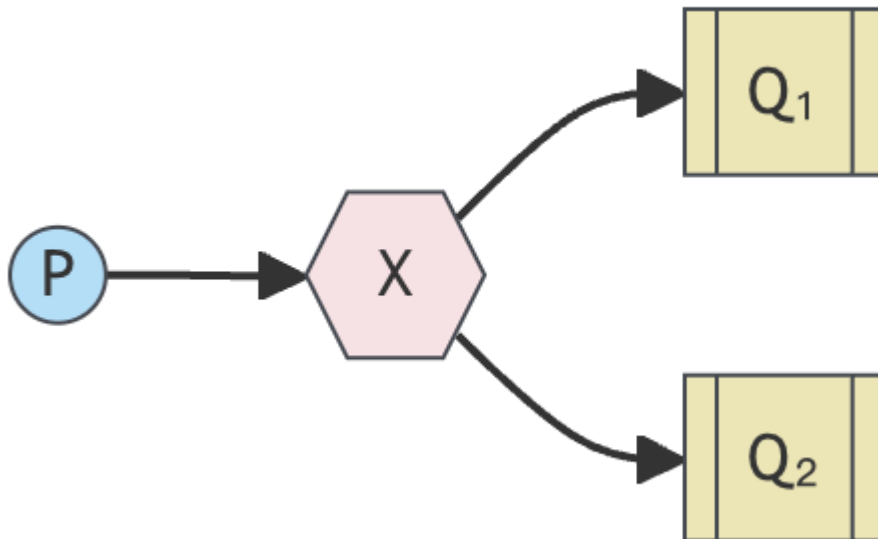
資料來源: <https://www.rabbitmq.com/tutorials>

SpringBoot 微服務 2024

— 段維瀚老師 —

Fanout 扇出交換機：

<https://www.rabbitmq.com/tutorials/tutorial-three-java>



RabbitMQ 的扇出交換機 (Fanout Exchange) 是一種將接收到的消息廣播到它所知道的所有隊列的交換機。特點如下：

- 廣播：消息送達所有綁定到此交換機的隊列，不管路由鍵是什麼。
- 不過濾：扇出交換機忽略綁定時設定的路由鍵，即路由鍵無效。
- 高效：適合實現發布/訂閱模式，如同時通知多個系統或組件更新。

典型應用場景包括：

- 廣播消息，如系統通知。
- 日誌系統，將日誌同時寫入多個儲存系統。
- 股票報價

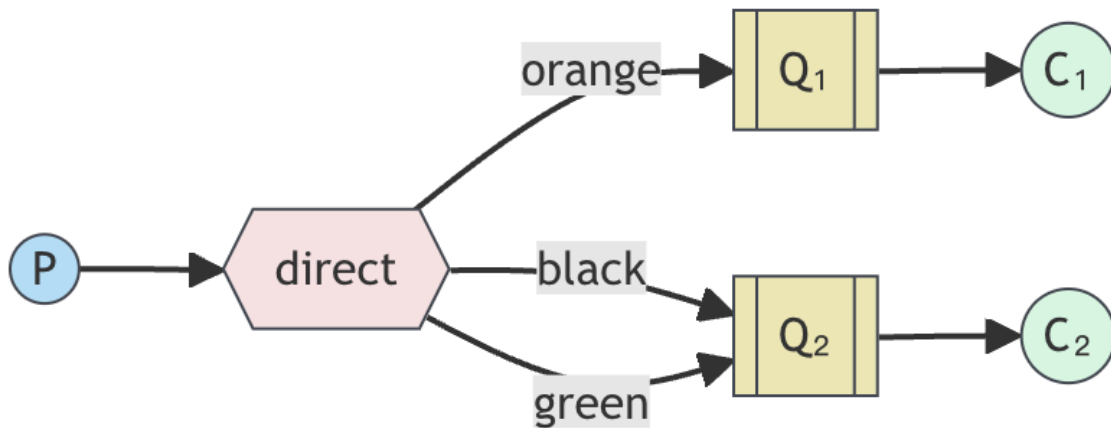
使用扇出交換機時，不需要考慮路由鍵，只需將隊列綁定到交換機即可實現消息的廣播。

SpringBoot 微服務 2024

— 段維瀚老師 —

Direct 直接交換機：

<https://www.rabbitmq.com/tutorials/tutorial-four-java>



RabbitMQ 的直接交換機 (Direct Exchange) 是根據消息中的路由鍵 (Routing Key) 直接將消息送到一個或多個隊列的一種交換機。

特點如下：

- 精確匹配：它會將消息送到路由鍵和隊列綁定鍵 (Binding Key) 完全一致的隊列中。
- 一對一：每條消息只會送到一個匹配的隊列，不會像扇出交換機那樣廣播。
- 靈活：可以通過設置不同的路由鍵和綁定鍵來達到靈活的消息路由方式。

典型應用場景包括：

- 不同系統間的直接通訊，如訂單系統通知庫存系統更新。
- 根據消息類型將其分發到專門的隊列中處理，如將錯誤消息分發到專門處理錯誤的隊列。

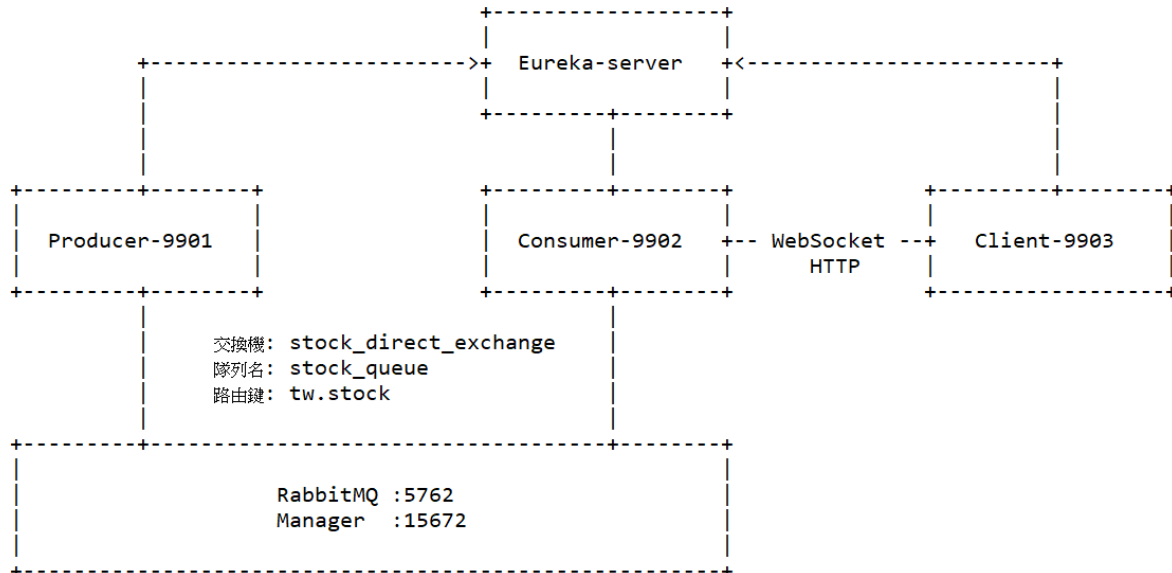
使用直接交換機時，要注意保持生產者指定的路由鍵和隊列的綁定鍵一致，以確保消息能夠被正確分發。

SpringBoot 微服務 2024

— 段維瀚老師 —

第8堂課: RabbitMQ 與微服務應用

Hands on Lab: Spring Cloud Stream RabbitMQ + WebSocket



下載報價資源檔:

<https://drive.google.com/file/d/1EAG5EGnjpKFHnTK0pP4Po5zzDTNqS1B0/view?usp=sharing>

1. 啟動 eureka-server

<http://localhost:8761/>

2. 啟動 rabbitmq-stream-twstock-starter-9901 (Producer)

觀看 console 是否有報價(將 price.json 的資料發射到 RabbitMQ)

`http://localhost:9901/stop` <-- 停止報價

交換機: `stock_direct_exchange` (設定在 `RabbitmqConfig.java` --> `new DirectExchange("stock_direct_exchange")`)

發送時的 routing key: `stock` (設定在 `PriceEmitterService.java` --> `rabbitTemplate.convertAndSend("stock_direct_exchange", "tw.stock", message)`)

3. 啟動 rabbitmq-stream-twstock-consumer-9902 (Consumer)

觀看 console 是否有報價(接收 RabbitMQ 的資料)

隊列名稱: `stockQueue` (設定在 `StockPriceService.java` -->

`@RabbitListener(queues = "stockQueue")`)

收到之後會透過 `WebSocket` 傳送到前端

SpringBoot 微服務 2024

— 段維瀚老師 —

交換機: stock_direct_exchange

隊列: stock_queue

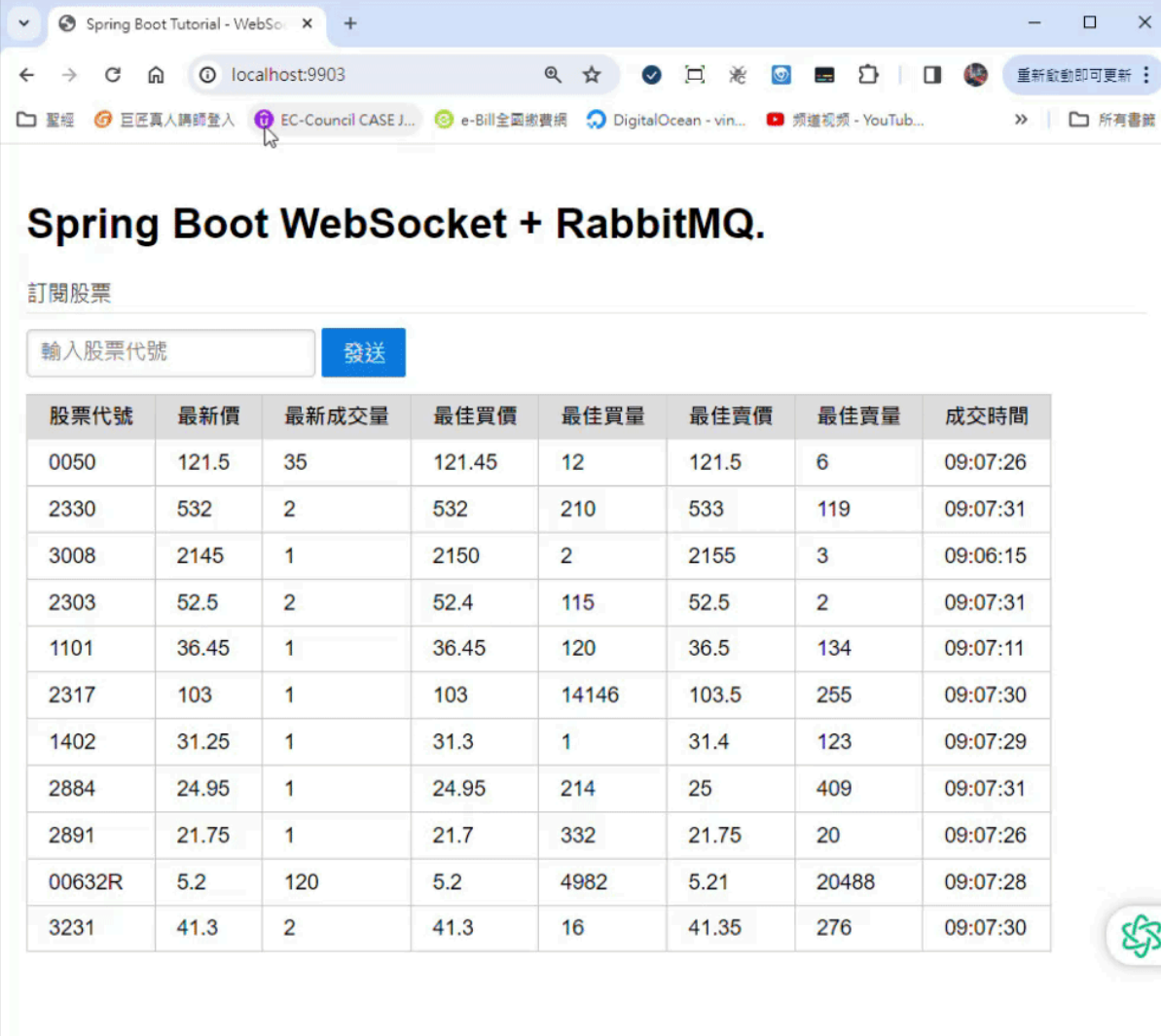
綁定的 routing key: tw.stock

上述三個設定在 RabbitmqConfig.java

4. 啟動 rabbitmq-stream-twstock-client-wibsocket-9903 (Client)

只有 index.html 一個頁面, 會透過 WebSocket 接收報價資料

5. 執行效果



Spring Boot Tutorial - WebSoc x +

localhost:9903

重新啟動即可更新

聖經 巨匠真人講解登入 EC-Council CASE J... e-Bill全國繳費網 DigitalOcean - vin... 頻道视频 - YouTub...

Spring Boot WebSocket + RabbitMQ.

訂閱股票

輸入股票代號 發送

股票代號	最新價	最新成交量	最佳買價	最佳買量	最佳賣價	最佳賣量	成交時間
0050	121.5	35	121.45	12	121.5	6	09:07:26
2330	532	2	532	210	533	119	09:07:31
3008	2145	1	2150	2	2155	3	09:06:15
2303	52.5	2	52.4	115	52.5	2	09:07:31
1101	36.45	1	36.45	120	36.5	134	09:07:11
2317	103	1	103	14146	103.5	255	09:07:30
1402	31.25	1	31.3	1	31.4	123	09:07:29
2884	24.95	1	24.95	214	25	409	09:07:31
2891	21.75	1	21.7	332	21.75	20	09:07:26
00632R	5.2	120	5.2	4982	5.21	20488	09:07:28
3231	41.3	2	41.3	16	41.35	276	09:07:30