项目 1　Spring Cloud 微服务部署概述

Spring Cloud 是分布式微服务架构的一站式解决方案。我们使用 Spring Boot 能够开发单体的业务模块(即单体服务),而利用 Spring Cloud 所提供的组件,则能够高效地将多个单体的业务模块整合成为一个微服务体系。本项目将介绍 Spring Cloud 微服务架构解决方案、Spring Cloud 微服务部署方式,以及 Ubuntu 操作系统下开发环境的搭建和使用,为后续项目的学习做好准备。

任务 1.1　微服务架构和 Spring Cloud

微服务架构是一种系统架构的设计风格。与传统的单体架构不同,微服务架构提倡将一个单一的应用程序拆分成多个小型服务,这些小型服务都在各自独立的进程中运行,服务之间使用轻量级通信机制进行通信。通常情况下,这些小型服务仅针对某一个业务模块来构建。

1.1.1　单体架构和微服务架构

单体架构和微服务架构是软件开发领域两个主要的系统架构风格。前者是业界经典的软件架构类型,大部分早期软件项目采用的都是单体架构,即将应用程序中的所有业务模块都放在一个项目工程中,经过编译和打包后,再部署在一台服务器上运行,如图 1-1 所示。

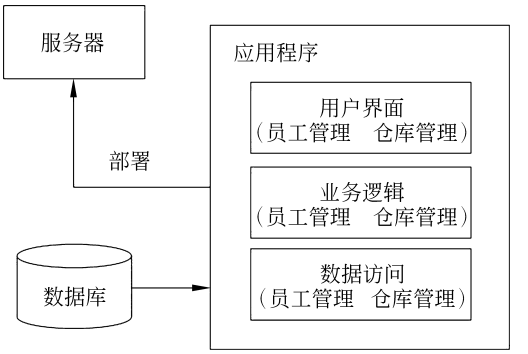

图 1-1　单体架构示意图

在项目的初期,单体架构在开发速度或是维护成本上都具有明显优势,随着业务复杂度不断提高,单体架构显现出诸多问题,主要体现在以下几个方面。

(1) 处理能力有限。随着应用规模的增长,一台服务器很难应对高并发请求和大规模数

据处理的需要。

(2) 耦合度高。所有业务集中于一个工程中,更新或扩展其中的一项业务都有可能对其他业务造成一定的影响。

(3) 部署风险大。整体性部署使得每次发布都可能涉及多个模块的修改和部署,增加了发布的风险和复杂性。

(4) 维护成本高。随着业务功能增加,代码复杂度会不断上升,造成维护成本不断加大,由此可能需要投入更多人力和时间来维护代码和修复问题。

由于单体架构存在这些弊端,许多公司和组织将一些大型项目逐步从单体架构转为新兴的微服务架构。如图 1-2 所示,微服务架构是将应用程序中原有的每个业务模块都独立为一个小型服务,每个服务拥有自己的数据库,并且可以独立部署到一个或多个服务器上,从而保证了服务之间的独立性和数据一致性。

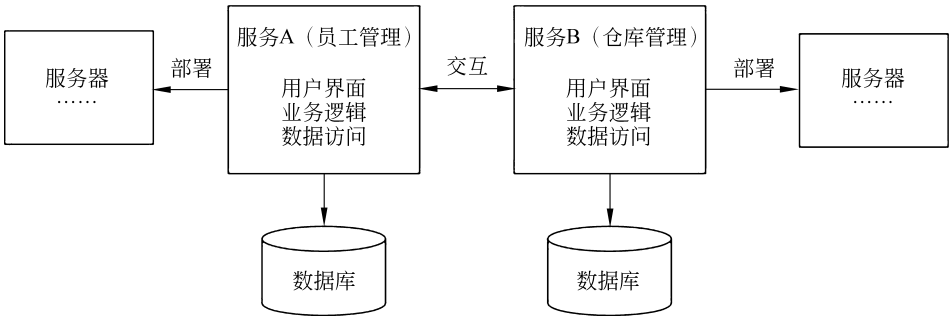

图 1-2 微服务架构示意图

从图 1-2 可以归纳出微服务架构的特点。

(1) 单一职责。每个服务都专注于解决特定的业务问题。

(2) 自治性。每个服务都是独立的,具有自己的数据存储、技术栈和开发团队,可以独立部署和交付。

(3) 轻量级通信。使用轻量级的通信协议(如 Restful API 或消息队列)来实现解耦,提高灵活性。

(4) 可扩展性。可对每个服务分别进行扩展,以应对访问量或业务需求增长的需要。

1.1.2 Spring Cloud 微服务架构

Spring Cloud 是目前国内使用最广泛的微服务架构解决方案。Spring Cloud 中集成了各种微服务功能组件,并基于 Spring Boot 实现了这些组件的自动装配,从而提供了良好的开箱即用体验。Spring Cloud 的常用组件具体如下。

(1) 服务注册发现: Eureka* 、Nacos、Consul。

(2) 统一配置管理: Config* 、Nacos。

(3) 统一网关路由: Gateway、Zuul* 。

(4) 服务远程调用: Feign* 、OpenFeign、Dubbo。

(5) 负载均衡: Ribbon。

(6) 流控、降级和保护：Hystrix* 、Sentinel、Resilience4J。

(7) 服务通信：Stream、RabbitMQ、RocketMQ。

提示：上述组件中带有 * 标记的表示已停止更新，不建议再使用。

1.1.3　Spring Boot 与 Spring Cloud 的关系

Spring Boot 和 Spring Cloud 是 Spring 生态系统中两个不同但相关的项目。Spring Boot 是一种基于 Spring 的快速开发框架，它提供了一系列的快速配置脚手架，使得开发者可以快速、方便地开发单体服务；而 Spring Cloud 是一个基于 Spring Boot 的、用于构建分布式系统的工具集合，它提供了一系列的组件，用于解决分布式系统中的常见问题，如服务发现、配置管理、负载均衡和熔断器等。

Spring Boot 和 Spring Cloud 的关系可以理解为：Spring Boot 主要用于单个服务的快速开发和部署，而 Spring Cloud 则专注于整个微服务架构的全局管理和治理。因此，Spring Boot 是构建应用程序的基础，而 Spring Cloud 则是在 Spring Boot 的基础上构建分布式系统的解决方案，Spring Cloud 需依赖于 Spring Boot 来开发和部署微服务。

基于 Spring Boot 和 Spring Cloud 之间的关系，我们将基于 Spring Boot 所开发的由单体服务构成的应用程序称为 Spring Boot 单体微服务，而将基于 Spring Boot 和 Spring Cloud 所构建的、由多个微服务构成的分布式系统称为 Spring Cloud 微服务，并将前者归为后者的特例。

任务 1.2　Spring Cloud 微服务部署方式

部署是软件项目开发流程最后一个环节，也是一个非常重要的内容。软件项目部署的任务是指将开发好的软件应用程序从开发环境或测试环境中转移到生产环境中，并确保其能够正常地运行，以便提交给用户使用。部署通常会涉及配置管理、版本控制和自动化脚本等方面的处理，而这些是软件应用程序在生产环境中稳定运行的重要保障。

由 1.1.3 小节可知，Spring Cloud 微服务是基于 Spring Boot 开发的应用程序(简称 Spring Boot 应用程序)，这类应用程序的部署方式主要有以下几种。

(1) JAR 包部署。JAR(Java Archive)包部署是将 Spring Boot 应用程序打包并构建成可执行的.jar 文件，其中包含了 Java 类文件、资源文件以及应用程序的所有依赖等，而后开发者可通过命令行或脚本运行.jar 文件。.jar 文件是一种 Java 应用程序打包和分发的文件格式，主要用于 Java 应用程序的类文件、资源文件和依赖库文件的打包，以及 Java 库的分发。这里的 Spring Boot 应用程序将以内嵌方式启动一个嵌入式的 Web 服务器，如 Tomcat、Jetty 或 Undertow，而无须依赖外部的部署环境。这种方式具有简单、方便的特点，非常适用于 Spring Boot 单体微服务的部署。

(2) Web 服务器部署。Web 服务器部署，即 WAR(Web Application Archive)包部署，是先将 Spring Boot 应用程序打包成.war 文件，其文件内容与 JAR 包部署中的.jar 相类似；再将.war 文件部署到外部的 Web 服务器，如 Tomcat、Nginx＋Tomcat 或 Jetty 等；最后启动 Web 服务器，即可运行该应用程序。.war 也是一种 Java 应用程序打包和分发的文件格式，且专用于 Java

Web 应用程序。这种方式适用于需要利用外部 Web 服务器的全功能管理和其他服务能力,或是需要与其他 Web 应用共享容器资源的场景。

(3) Docker 容器部署。Docker 容器部署是将 Spring Boot 应用程序打包成 Docker 镜像,再通过 Docker 容器化技术在 Docker 容器中进行部署和管理,之后就可以在容器中运行该应用程序了。这种方式能够提供更好的隔离性、可移植性和可扩展性,同时简化了应用程序的部署和管理流程,适用于需要快速、可靠、可移植的应用程序部署的场景,特别适于 Spring Cloud 微服务的部署。

(4) 容器编排平台部署。容器编排平台部署是将 Spring Boot 应用程序打包并构建成 Docker 镜像;而后利用容器编排平台,如 Kubernetes、Docker Swarm 等,在 Docker 容器中实现应用程序的自动化部署、扩展和管理;最终应用程序是在 Docker 容器中运行的。这种方式可以实现自动部署,提供服务发现和负载均衡等扩展功能,适用于管理和部署大规模的 Spring Cloud 微服务集群。

总的来说,JAR 包部署和 Web 服务器部署均为传统的部署方式,主要适用于小规模 Spring Cloud 微服务(如 Spring Boot 单体微服务)项目,而 Docker 容器部署和容器编排平台部署则属于容器化的部署方式,更加适用于大规模 Spring Cloud 微服务项目。

任务 1.3　搭建基础开发环境

本任务将在 Ubuntu(版本 20.04.2 LST)操作系统下,介绍 Spring Boot 应用程序开发所需的最基本开发环境的安装方法,后续项目会在此基础上根据内容需要陆续增加其他中间件。IDEA(IntelliJ IDEA)是一个用于 Java 语言开发的集成环境,它将代码编写、编译、执行和调试等多种功能综合到一起,是目前业界公认的最好的 Java 程序开发工具,同时它也非常适于 Spring Boot 应用程序的开发。

1.3.1　安装 JDK

本书采用 Spring 6 和 Spring Boot 3 进行案例开发,对应的 JDK 版本至少是 17.0.10。在 Ubuntu 系统中,打开命令终端,进入系统根目录,以 root 用户身份输入以下命令进行 JDK 的安装:

```
apt update                                    #更新软件包列表
apt install openjdk-17-jdk                    #安装版本为 17 的 JDK
```

命令执行完后,安装好的 JDK 将位于/usr/lib/jvm 目录下。执行 java -version 命令,当看到如图 1-3 所示的 JDK 版本信息时,表示 JDK 安装和配置完成。

```
gll@ubuntu:/$ java -version
openjdk version "17.0.10" 2024-01-16
OpenJDK Runtime Environment (build 17.0.10+7-Ubuntu-120.04.1)
OpenJDK 64-Bit Server VM (build 17.0.10+7-Ubuntu-120.04.1, mixed mode, sharing)
```

图 1-3　获取 JDK 版本信息

1.3.2　安装 IDEA

在 IDEA 官网首页单击 Download 选项，下载基于 Linux 系统的 IDEA 安装包，如图 1-4 所示。

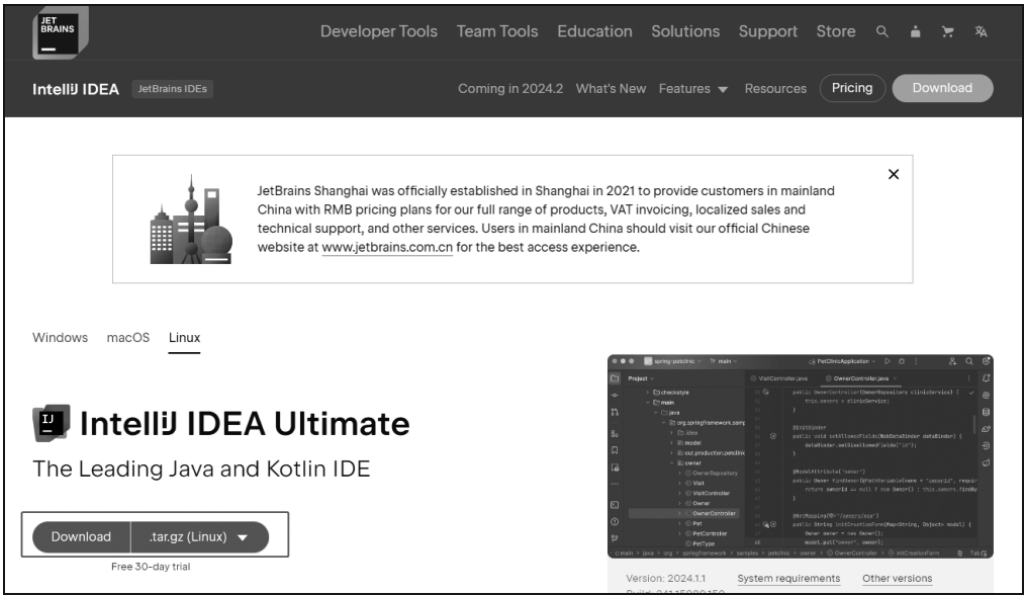

图 1-4　IDEA 官网首页

将安装包复制到 Ubuntu 系统/usr/local/idea 目录下，打开命令行终端窗口，输入以下命令对其进行解压：

```
tar -zxvf ideaIU-2024.1.1.tar.gz
```

解压后自动生成 idea-IU-241.15989.150 目录，该目录即为 IDEA 安装目录，如图 1-5 所示。

```
gll@ubuntu:/usr/local/idea$ ls
ideaIU-2024.1.1.tar.gz  idea-IU-241.15989.150
gll@ubuntu:/usr/local/idea$
```

图 1-5　解压后生成的 IDEA 安装目录

进入/usr/local/idea/idea-IU-241.15989.150/bin 目录，输入以下命令启动 IDEA：

```
./idea.sh
```

执行上述命令后，进入如图 1-6 所示 IDEA 界面，表示安装成功。

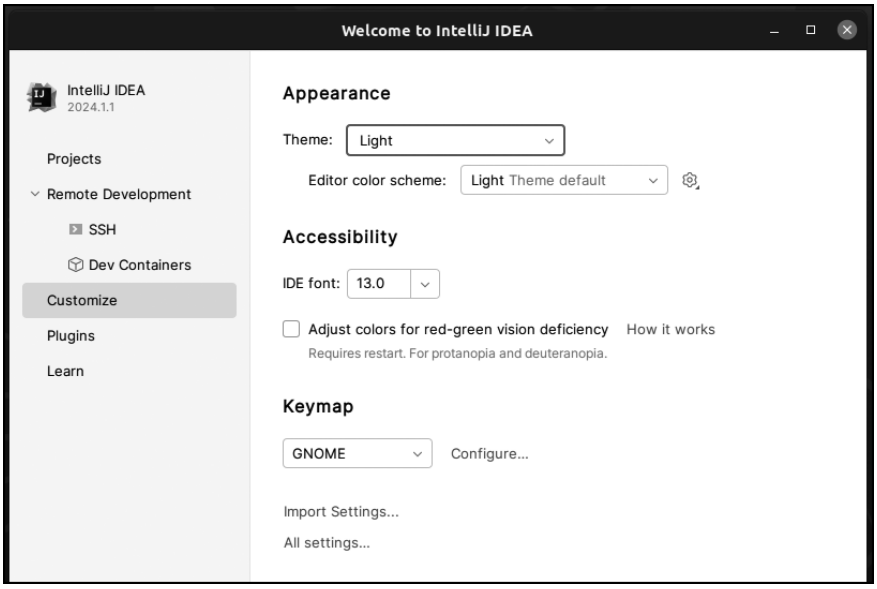

图 1-6　IDEA 界面

任务 1.4　综合案例：Spring Boot 单体微服务搭建和 JAR 包部署实践

在了解了 Spring Cloud 微服务相关的基础知识后，本任务将通过一个简单 Spring Boot 单体微服务项目的开发和部署，介绍如何应用 IDEA 工具开发 Spring Boot 应用程序，以及采用 JAR 包方式对其进行部署和运行。

1.4.1　案例目标

任务内容：利用 IDEA 工具构建一个 Spring Boot 单体微服务项目，编写 Controller 类，对于 HTTP 请求进行处理，实现输出信息为“这是一个 Spring Boot 单体微服务”的功能，此外，还需要采用 JAR 包方式实现项目的部署和运行。搭建完成后的项目结构和运行效果如图 1-7 所示。

1.4.2　任务分析

本任务的目标是搭建和部署 Spring Boot 单体微服务项目，以实现对 HTTP 请求的处理。我们将利用 Spring Boot 框架技术和 IDEA 工具提供的功能，基于 Spring Boot 创建一个 Web 应用程序，实现接收 HTTP 请求并返回字符串的简单功能，同时使用 Maven 管理并自动构建该应用程序，从而达到任务要求。

由于是首个案例，有必要先介绍一下 Spring Boot 项目的组成结构。由图 1-7(a)可知，

(a) 项目结构　　(b) 运行效果

图 1-7　项目结构和运行效果

项目结构的组成要素包括业务处理部分(com.example.springbootdemo)、启动类(SpringBootDemoApplication)、Maven 项目配置文件(pom.xml)、Spring Boot 项目配置文件(application.properties),其中业务处理部分位于项目的 java 目录下,通常包括处理请求和响应的控制器类(controller)、实现业务功能的服务类(service)和处理数据操作的数据访问类(DAO)。启动类位于项目最顶部的包路径下,用于启动 Spring Boot 应用程序,具有自动开启配置和组件扫描等特性;pom.xml 位于项目根目录下,用于管理项目的依赖和插件;application.properties 位于项目 resources 目录下,负责 Spring Boot 项目的相关配置,如服务器端口和数据库连接信息等,当默认配置无法满足需要时就可在该文件中进行修改。

在了解了 Spring Boot 项目结构之后,确定任务的实现思路如下。

(1) 使用 Spring Boot 搭建 Web 项目。

(2) 编写业务处理部分、启动类和配置文件。

(3) 利用 Maven 命令进行 JAR 包部署,生成可执行的 Spring Boot 应用程序。

(4) 利用 java 命令运行 Spring Boot 应用程序。

1.4.3　任务实施

步骤一:使用 Spring Boot 搭建 Web 项目。

打开 IDEA 工具,单击 New Project 按钮,进入新建项目界面,如图 1-8 所示,选择左侧菜单中的 Spring Boot 选项,使用该选项只需填写一组属性,即可快速完成基于 Spring Boot 的项目搭建工作。接着逐个填写界面右侧的各个属性:Name(项目名称)属性中输入 spring-boot-demo;Location(项目位置)属性中输入项目保存的路径;Language(开发语言)属性选择 Java;Type(项目类型)属性选择 Maven;Group(组织机构)、Artifact(项目名称)和 Package name(包名称)属性会自动生成,可选择默认值;JDK 和 Java 属性选择 17;

Packaging(打包形式)属性选择 Jar。然后单击 Next 按钮,进入项目依赖项界面(图 1-9):Spring Boot 属性选择 3.2.9,Dependencies(依赖项)属性选择 Spring Web 以便构建 Web 应用程序。最后,单击 Create 按钮完成项目的新建工作。

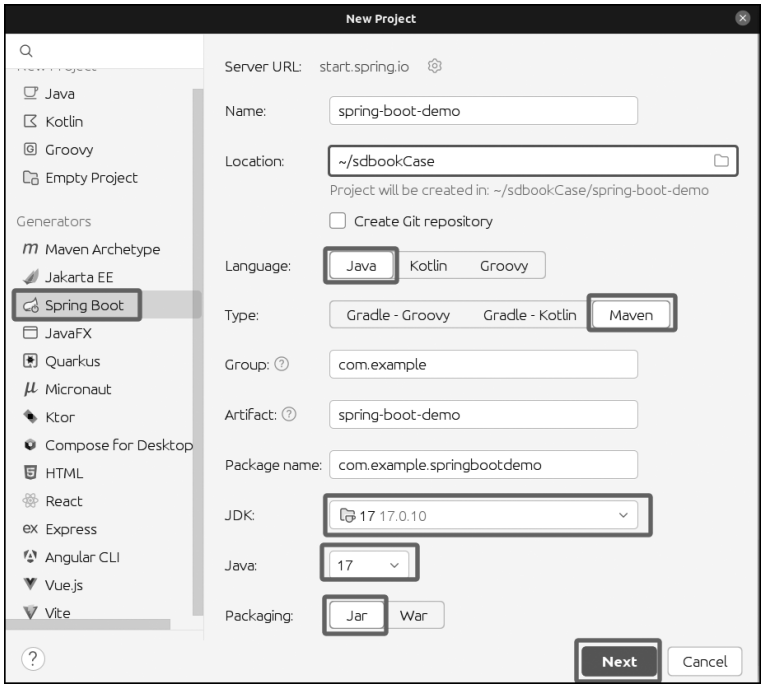

图 1-8　新建项目界面

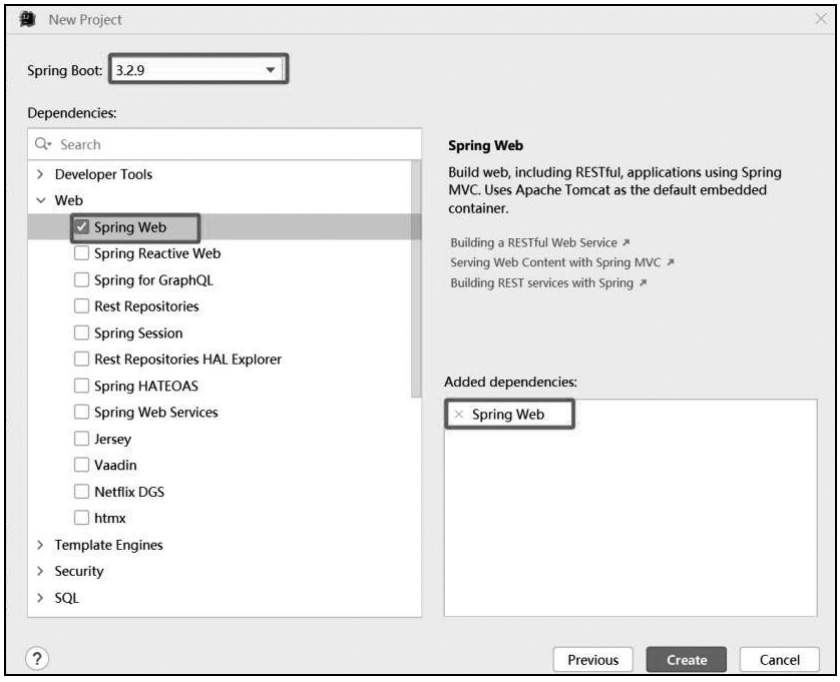

图 1-9　项目依赖项界面

步骤二：编写 pom.xml 文件。

本任务所搭建的 Spring Boot 项目是一个 Web 应用程序，需要引入 Spring Web 依赖项（步骤一已完成）。pom.xml 文件完整代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
<!--pom 版本-->
<modelVersion>4.0.0</modelVersion>
<!--父级项目信息-->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.9</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<!--组织机构域名的倒序-->
<groupId>com.example</groupId>
<!--本项目 id-->
<artifactId>spring-boot-demo</artifactId>
<!--本项目版本-->
<version>0.0.1-SNAPSHOT</version>
<!--项目打包方式-->
<packaging>jar</packaging>
<name>spring-boot-demo</name>
<description>Demo project for Spring Boot</description>
<!--变量声明-->
<properties>
    <java.version>17</java.version>
</properties>
<!--项目依赖项-->
<dependencies>
    <!--Spring Web 依赖包-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

```
42  <!--项目构建配置-->
43  <build>
44      <!--生成文件的最终文件名-->
45      <finalName>SpringBootDemo</finalName>
46      <!--插件配置-->
47      <plugins>
48          <!--Spring Boot 插件-->
49          <plugin>
50              <groupId>org.springframework.boot</groupId>
51              <artifactId>spring-boot-maven-plugin</artifactId>
52          </plugin>
53      </plugins>
54    </build>
55  </project>
```

代码说明：第 22 行表示项目打包形式为 JAR，它是默认打包形式，因此，可以不配置 packaging 标签。第 32～35 行表示引入 Spring Web 依赖包，它包含了创建 Web 应用程序所需的基本依赖集合，包括 Spring MVC、Tomcat 服务器和 Jackson 等。第 49～52 行表示 Spring Boot 插件配置，它用于将 Spring Boot 项目打包成 JAR 或 WAR 包。如果是 JAR 包，则可以执行，通过执行 JAR 包可启动 Spring Boot 项目工程。

步骤三：编写 Controller 类。

在 Spring Boot 项目中，Controller 类用于接收 HTTP 请求和返回业务处理结果。由于本任务仅要求接收 HTTP 请求后返回字符串，因此，业务处理部分仅需编写一个 Controller 类 MainController 即可，其代码如下。

```
1   @RestController
2   public class MainController {
3       @GetMapping("/demoTest")
4       public String test(){
5           return "这是一个 Spring Boot 单体微服务";
6       }
7   }
```

代码说明：第 1 行@RestController 组合了@Controller 和@ResponseBody 两个注解，用于标识 MainController 类为一个 Restful 风格控制器类，使得该类可处理 HTTP 请求，并返回 JSON 或其他格式的响应结果数据。第 3 行@GetMapping 注解标识 test()方法为接收 HTTP GET 请求的处理方法，且其请求路径 URL 为/demoTest。第 4～6 行所定义的 test()方法将返回 String 类型的结果数据。

步骤四：编写启动类。

启动类是整个 Spring Boot 应用程序的入口点，使用 Spring Initializer 模板构建 Spring Boot 项目会自动生成该类。本项目的启动类 SpringBootDemoApplication 的代码如下。

```
1   @SpringBootApplication
2   public class SpringBootDemoApplication {
3       public static void main(String[] args) {
```