第 1 章　初识微服务

当今 Java 系统架构已经从传统的单体架构演进到微服务架构，越来越多的公司以微服务架构作为衡量产品架构的标准。微服务是一种软件系统架构模式，提倡将单一应用程序划分成一组松散耦合的小型服务，各个服务之间通过轻量级的协议互相通信，使得整个软件系统更好地支持基于云原生的分布式部署。

1.1　了解软件系统架构的演进

随着互联网的发展，系统应用的规模越来越大，业务越来越复杂，进而系统架构也在不断地进行变化。从总体上看，Java 系统架构大体经历了单体架构、垂直分布式架构、SOA（service-oriented architecture，面向服务架构）架构和微服务架构这样四个阶段。

1.1.1　单体架构

单体架构即一个系统的所有功能都包含在一个 war 包或者 jar 包中，如图 1-1 所示。这种架构节约人力成本，能够快速开发和上线部署、维护成本低，很多初创型公司早期系统大多采用这种架构。在项目初期用户量不大的情况下，单体架构足以支撑系统业务的正常运行。随着业务的发展，单体系统暴露出了一系列问题。

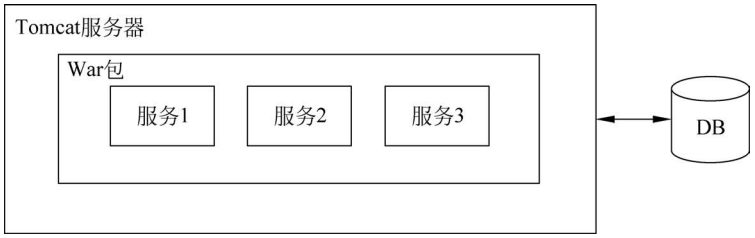

图 1-1　单体架构

（1）用户量越来越大，网站的访问量也不断增大。为满足不同用户需求，系统的业务也越来越复杂。系统内模块高度耦合，对某个模块微小功能的调整，可能会对其他模块带来不可知的影响和潜在的缺陷。

（2）业务越多，系统程序包的容量也越大，系统部署升级的过程也越来越困难。每次发布新版本都是整个系统进行发布，系统部署需要重启整个系统。

（3）单体系统存在性能瓶颈问题，单体架构的服务器 CPU、内存、磁盘等资源都是有限

的。即使添加资源也不可能无限制地扩展。

传统的单体架构已经渐渐地不能满足需求了。需要使用一种新的软件架构来解决上述问题。

1.1.2 垂直分布式架构

为了解决传统架构的问题，人们借用大数据中“分而治之”的思想，同时部署多个系统到多台服务器上，进行 CPU、内存、磁盘存储的负载均衡，以增加系统的服务性能，这就是分布式架构。在分布式架构的使用过程中，人们同时发现并不需要对整个系统进行分布式部署，只有系统中部分访问量很高的模块才需要分布式部署。因此将系统中模块进行垂直拆分，将系统拆分成多个模块。对高访问量的模块分布式部署多个服务器，低访问量的模块还是单服务器部署，这就是垂直分布式架构，如图 1-2 所示。垂直分布式架构实现了流量分担，解决了并发问题，而且可以针对不同模块进行优化和水平扩展。

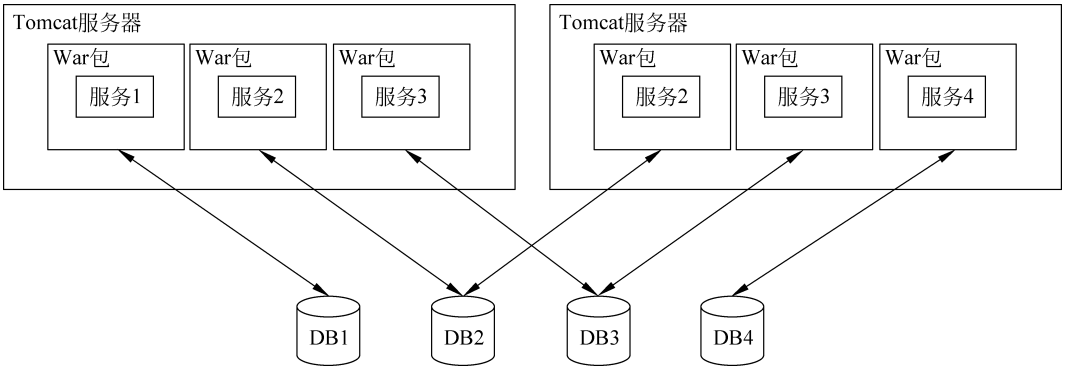

图 1-2 垂直分布式架构

当垂直模块越来越多，重复的业务代码就会越来越多。例如，A 模块需要调用服务 2，B 模块也要调用服务 2，是不是可以将服务 2 这部分重复代码抽取出来，做成统一的业务服务供不同模块调用？这样就产生了 SOA 架构。

1.1.3 SOA 架构

SOA 架构是一种面向服务的架构方式。该架构把一些通用的、被多个服务调用的共享业务服务提取出来，整合成共享的基础服务，这些服务相对来说比较独立，也可以重用。这样一个完整的业务就被划分为一些粗粒度的业务服务和对应的业务流程。当这种服务越来越多时，就需要一个管理者来管理这些服务。因此，一般 SOA 架构中会存在一个服务注册调度中心对集群进行实时管理，如图 1-3 所示。

SOA 架构的服务注册调度中心未实现组件化，功能较为单一，只负责管理服务，在服务治理方面有很多缺陷。例如，一个完整的业务流程需要依次调用多个服务，一旦某个服务调用环节出错，会卡死整个业务流程，造成服务雪崩。同时各服务关系错综复杂，如何统一部署、访问和运维？因此产生了微服务架构。

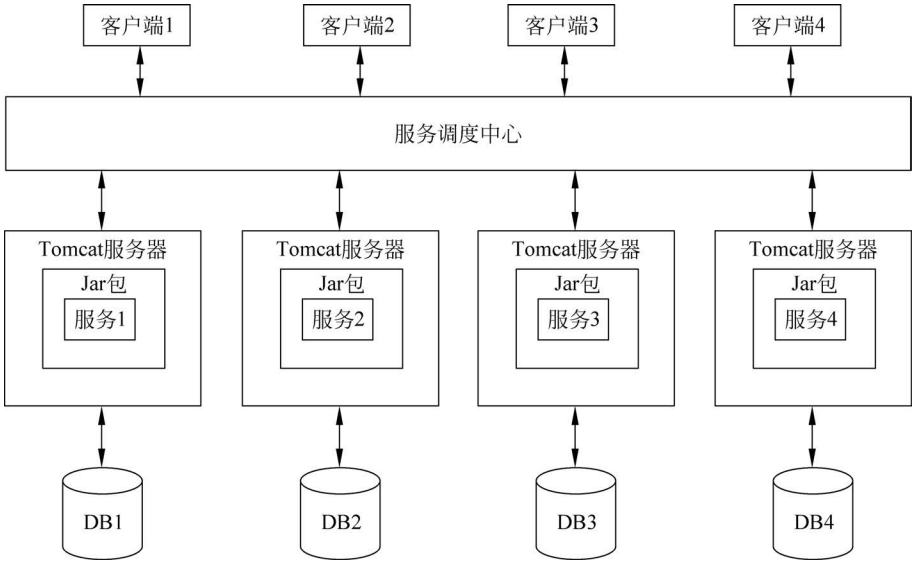

图 1-3　SOA 架构

1.1.4　微服务架构

微服务架构是面向服务架构 SOA 的进一步发展，目的是更好地对各服务进行治理。它将 SOA 服务进行彻底拆分，拆分成更小的细粒度服务，每个服务都是一个可以独立运行的项目。在微服务架构中，各服务由注册中心统一管理，注册中心实现组件化，本身也可以看作是一个微服务。服务之间通信采用基于 HTTP 的 RESTful 接口或基于 TCP 的 RPC 协议。外部客户端通过统一的网关访问内部各个服务，并对服务链路进行日志记录，性能监控和链路追踪，如图 1-4 所示。

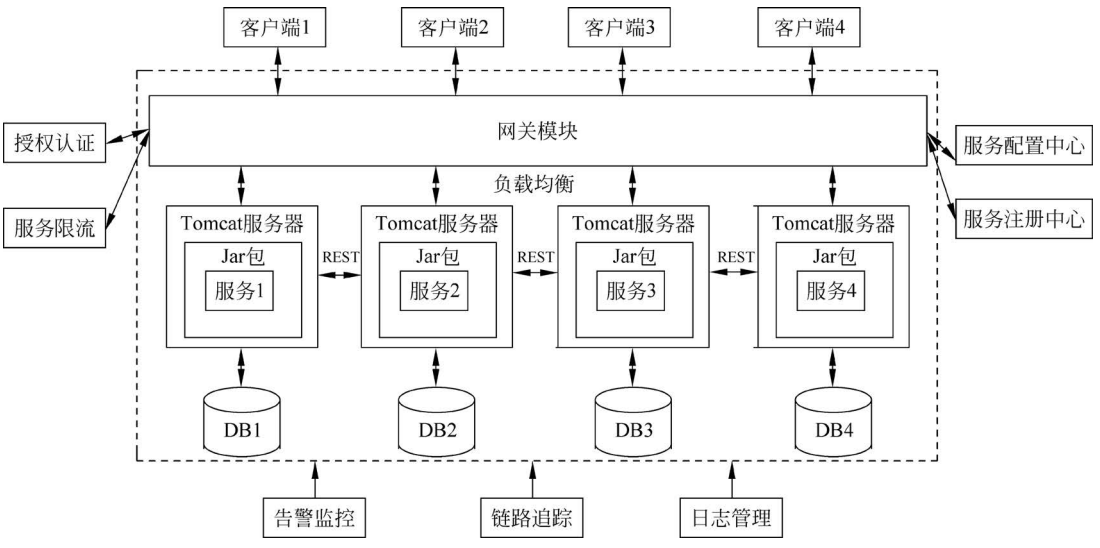

图 1-4　微服务架构

在服务访问过程中，如果某一个服务发生故障，可以通过重试、降级等机制来实现服务的容错，使故障隔离在单个服务中，避免服务雪崩。在系统更新迭代时，每个微服务都是一个独立运行的项目，可以实现快速单独部署，而不需要重新部署整个系统。

微服务架构

1.2 认识 Spring Cloud 微服务框架

在开发中，一般使用现有的微服务框架快速构建微服务应用，目前主流的微服务框架之一就是 Spring Cloud 系列。Spring Cloud 为微服务框架的实现提供了一个标准规范，约定一个微服务框架要提供服务注册发现、服务网关、远程服务通信、负载均衡、服务熔断、分布式消息、配置中心、链路监控等功能组件。实际上一个 Spring Boot 就是一个微服务的最小单元，整个微服务系统可以看作是由很多个 Spring Boot 项目组成。在此基础之上 Spring 团队基于 Spring Boot 整合实现了基于 Spring Cloud 规范的微服务框架，用于对内部各 Spring Boot 项目进行统一管理。基于 Spring Cloud 规范的微服务框架的发展经历了 2 代，分别是 Spring Cloud Netflix 和 Spring Cloud Alibaba。本任务将详细介绍二者的基本组件和异同点。

1.2.1 Spring Cloud Netflix

Spring Cloud Netflix 是第一代微服务架构的开源实现。Netflix 是美国 Netflix 公司开发并开源的一套微服务框架，并在生产环境中稳定运行。

1. Netflix 内部提供的组件

Netflix 内部提供了多种组件，包括服务治理组件注册中心（Eureka）、负载均衡组件（Ribbon）、服务调用组件（Feign）、服务熔断组件（Hystrix）、网关组件（Zuul）、配置管理组件（Archaius）等。Spring 团队对 Netflix 组件进行再次封装和整合，构建了 Spring Cloud Netflix 微服务框架，提升了 Netflix 框架的易用性。Spring Cloud Netflix 整体架构如图 1-5 所示。

（1）Eureka。这是一款基于 Rest 服务的服务治理组件，称为注册中心，用于服务注册与发现，同时实现了云端负载均衡和服务的故障转移功能。

（2）Ribbon。这是一款负载均衡组件，用于实现客户端服务调用的负载均衡。

（3）Hystrix。这是一款容错管理组件，用于实现服务熔断功能，通过断路器模式，隔离故障服务，从而保障微服务系统的高可用性。

（4）Feign。这是一款基于 Ribbon 和 Hystrix 的声明式服务远程通信组件。

（5）Zuul。这是一款微服务网关，用于提供动态路由及访问过滤等服务。

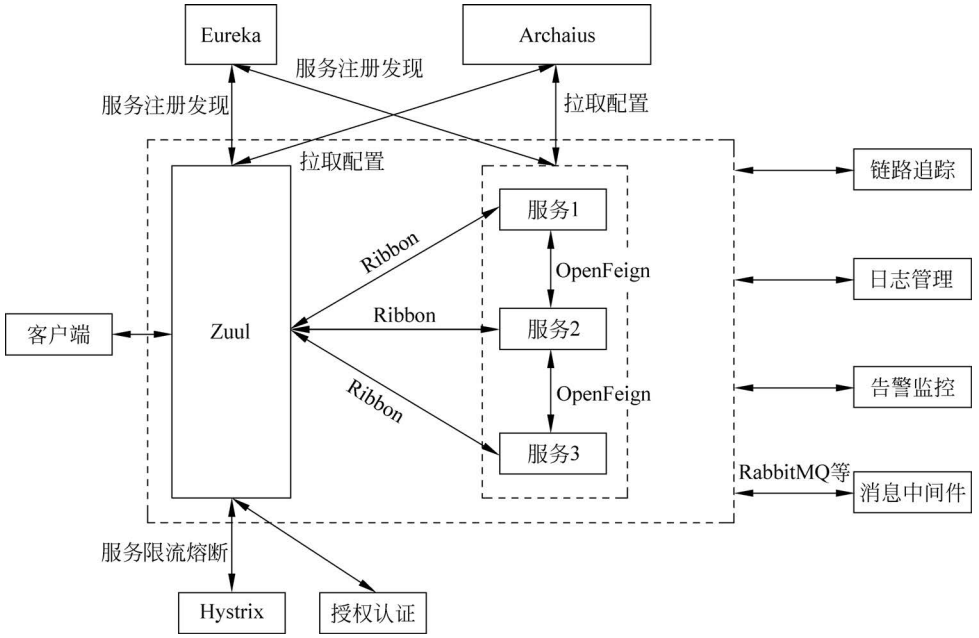

图 1-5　Spring Cloud Netflix 整体架构

(6) Archaius。配置管理 API,包含一系列配置管理 API,提供动态类型化属性、线程安全配置操作、轮询框架、回调机制等功能。

2. Spring Cloud 官方开源的微服务组件

Netflix 微服务框架可整合一些 SpringCloud 官方开源的微服务组件,例如,远程服务通信组件(Spring Cloud ResTemplate 和 Spring Cloud OpenFeign)、API 网关组件(Spring Cloud Gateway)、分布式消息组件(Spring Cloud RabbitMQ)、配置中心(Spring Cloud Config)、负载均衡组件(Spring Cloud LoadBalancer)等。

(1) Spring Cloud ResTemplate:Spring 官方提供的基于 RESTful 的远程服务通信组件,可实现不同微服务之间的调用。RestTemplate 底层提供了对 HTTP 请求及响应的封装,用于简化远程服务通信的实现过程。

(2) Spring Cloud OpenFeign:Spring 官方提供的基于 Feign 的远程服务通信组件。OpenFeign 在 Feign 基础上提供了对 Spring MVC 注解的支持,简化了远程服务通信的实现过程。

(3) Spring Cloud Gateway:Spring 官方提供的网关组件,用于替代旧网关组件 Netflix Zuul。Gateway 旨在为微服务架构提供统一的 API 路由管理方式。Gateway 的基本功能包括路由管理、监控、限流等。

(4) Spring Cloud RabbitMQ:基于 Erlang 语言开发的开源消息通信中间件。Spring 官方集成 RabbitMQ 用于各微服务之间的消息通信。

(5) Spring Cloud Config:Spring 官方提供的分布式配置中心组件,用于各微服务配置文件的统一管理和实时更新。

(6) Spring Cloud LoadBalancer:Spring 官方提供的负载均衡组件,用于替代 Ribbon。

2018 年 12 月，Spring Cloud Netflix 进入维护模式，一些重要组件如注册中心 Euraka、Ribbon 已经不再迭代更新了。但目前仍然有不少开发者还在使用 SpringCloud Netflix 构建微服务应用。

1.2.2 Spring Cloud Alibaba

2019 年 7 月，Spring Cloud Alibaba 成为 Spring 社区正式项目。Spring Cloud Alibaba 是第二代微服务架构的开源实现，内部包含阿里开源组件和商业化组件，以及部分常用的 Spring Cloud 官方开源组件。使用时开发者只需添加少量注解和配置，就可以通过阿里中间件来迅速搭建分布式应用系统。对于个人开发者来说，主要使用阿里开源组件和 Spring Cloud 官方开源组件开发微服务系统，因此，后续介绍内容不涉及阿里商业化组件。

Spring Cloud Alibaba 整体架构

Spring Cloud Alibaba 开源组件整体架构如图 1-6 所示。

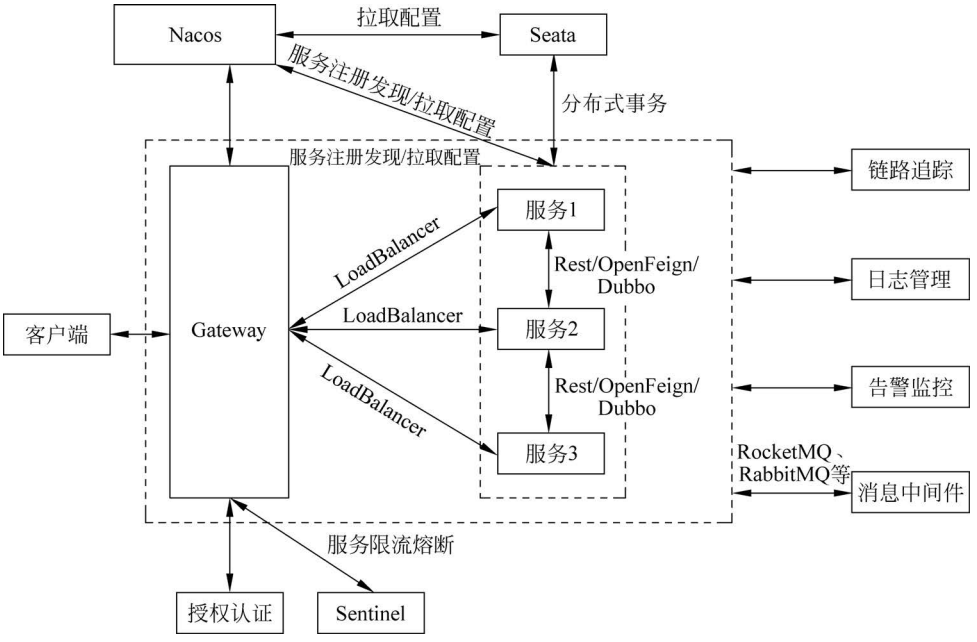

图 1-6　Spring Cloud Alibaba 开源组件整体架构

1. 阿里开源组件

（1）Nacos：一款基于分布式的服务注册发现和配置管理组件。Nacoss 将服务发现和服务配置功能集成整合。

（2）Sentinel：一款面向分布式服务架构的轻量级流量控制组件，用于服务的流量控制、熔断降级、系统负载保护等。

（3）RocketMQ：一款分布式消息组件，该系统基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。

（4）Dubbo：一款高性能 RPC 组件，用于实现服务通信。

（5）Seata：一款分布式事务管理组件。

2. 阿里商业化组件

（1）Alibaba Cloud ACM：阿里云提供的一款对服务配置进行集中管理和推送的配置中心。

（2）Alibaba Cloud OSS：阿里云提供的一款海量、安全、低成本、高可靠的云存储服务。

（3）Alibaba Cloud SchedulerX：阿里云提供的一款分布式任务调度产品，实现基于 Cron 表达式的定时任务调度服务。

（4）Alibaba Cloud SMS：阿里云提供的一款短信服务的产品。

同时，Spring Cloud Alibaba 也兼容常用的 SpringCloud 官方开源组件。例如，使用 Spring Cloud Gateway 作为微服务网关；使用 Open Feign 或 RestTemplate 进行远程服务通信；使用 RabbitMQ 进行服务消息传递；使用 Spring Cloud LoadBalancer 进行负载均衡。

表 1-1 是基于 Spring Cloud Netflix 和 Spring Cloud Alibaba 的微服务架构相关组件对比。

表 1-1　Spring Cloud Netflix 和 Spring Cloud Alibaba 的微服务架构相关组件对比

组件类型	Spring Cloud Netflix	Spring Cloud Alibaba
服务注册发现	Eureka	Nacos
配置中心	Archaius	Nacos
服务熔断	Hystrix	Sentinel
服务调用	Feign/ResTemplate	Feign/ResTemplate/Dubbo
服务网关	Zuul	Spring Cloud Gateway
分布式消息	RabbitMQ	RabbitMQ/RocketMQ
负载均衡	Ribbon	Spring Cloud LoadBalancer/Dubbo LB
分布式事务管理	无	Seata

1.3　搭建 Spring Cloud Alibaba 项目

分布式项目是构建 Spring Cloud Alibaba 项目的基础。本任务将首先介绍如何使用 Idea 搭建一个简单的分布式项目，进而引入 Spring Cloud Alibaba 依赖搭建一个 Spring Cloud Alibaba 项目。

1.3.1 搭建分布式项目

本任务搭建的分布式项目包含一个父 Maven 项目和两个 Spring Boot 子项目。一个 Spring Boot 项目作为服务生产者提供服务；另一个 Spring Boot 项目作为服务消费者，远程调用服务生产者提供的服务。项目中 Idea 版本为 2023，Java 版本为 17，Spring Boot 版本为 3.0.2。

1. 创建父项目

下面创建的父项目是一个 Spring Boot Maven 项目，但是内部没有 src 目录，通过 pom.xml 文件管理所有子项目的版本依赖。

（1）打开 Idea，选择 File→New→Project 命令，如图 1-7 所示。

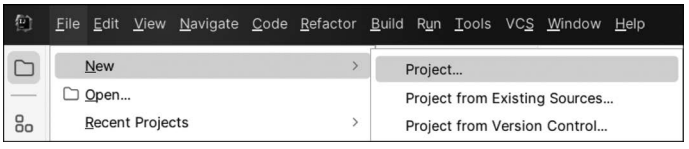

图 1-7　选择 File→New→Project 命令

（2）进入 New Project 对话框，在对话框左侧选中 Spring Initializr 选项，在对话框右侧进行如图 1-8 所示设置。设置 Name 选项为 SpringCloudDemo1；Location 选项可以自定义

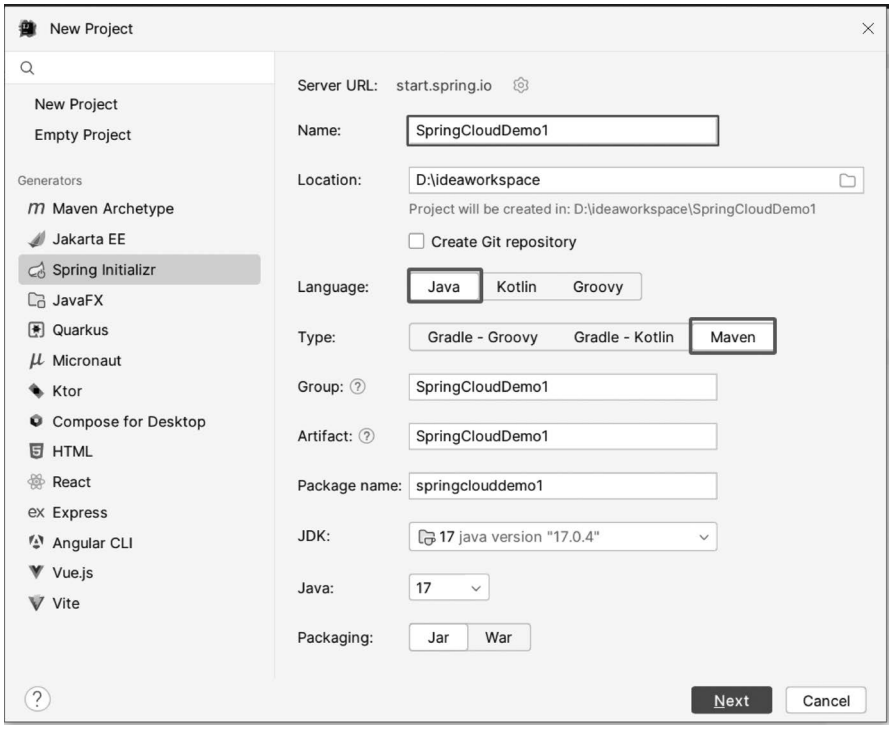

图 1-8　New Project 对话框的 Spring Initializr 界面

设置项目存放位置，这里设置为 D:\ideaworkspace；Language 选项选择 Java；Type 选项选择 Maven；Group 和 Artifact 选项中输入 SpringCloudDemo1；Packagename 选项中输入 springclouddemo1；Java 版本为 17。

（3）设置完毕，单击 Next 按钮，进入图 1-9 所示对话框。在对话框中，默认 Spring Boot 版本为 3.2.4，用户可以根据需要选择导入一些依赖。这里不修改 Spring Boot 版本，也不导入任何依赖，直接单击 Create 按钮，即可完成父项目的创建。

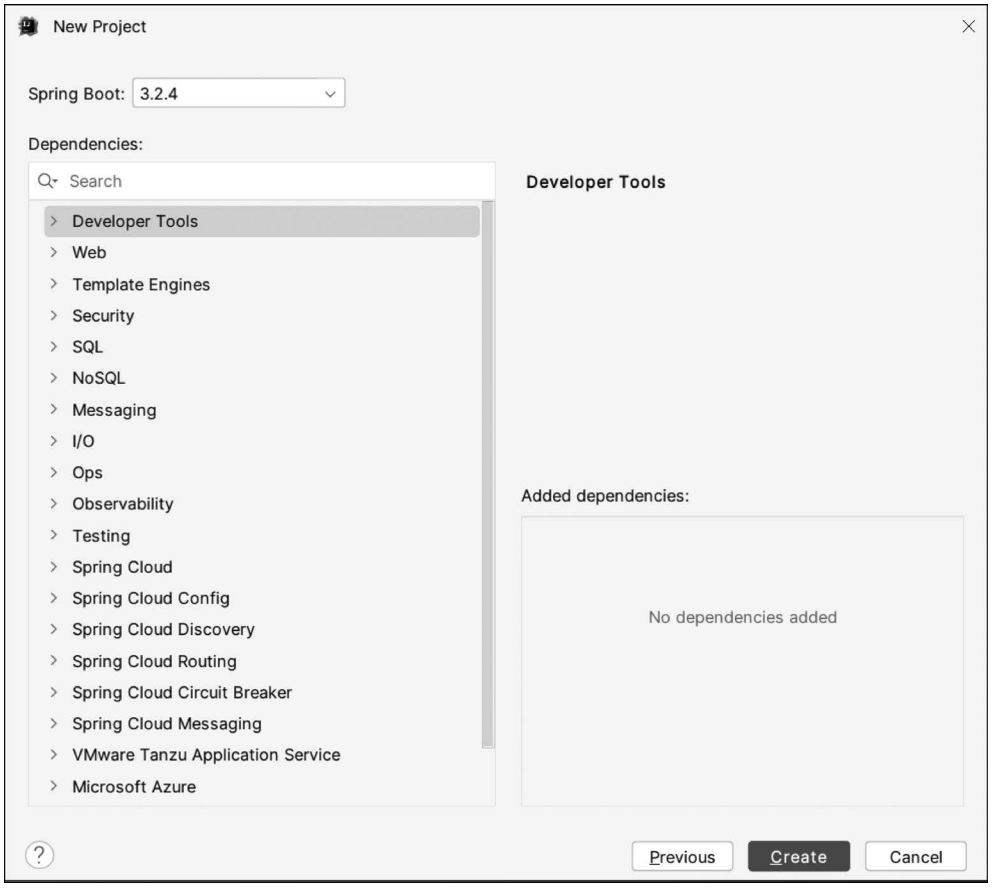

图 1-9　New Project 对话框的 Dependencies 界面

（4）删除父项目 SpringCloudemo1 的 src 文件夹。在 SpringCloudemo1 项目的 pom.xml 文件中设置 spring-boot-starter-parent 的版本为 3.0.2。添加 < packaging > pom </packaging>标签，设置父项目在工程中不会打成 Jar 包。设置打包插件 spring-boot-maven-plugin 版本为 3.0.2。设置父项目编译版本为 java17，这样后面创建的子项目也会自动继承父项目并使用 java17 编译项目。

【pom.xml】

```
<? 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">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>SpringCloudDemo1</groupId>
    <artifactId>SpringCloudDemo1</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>SpringCloudDemo1</name>
    <description>SpringCloudDemo1</description>
    <!-- 为父项目添加 packaging 为 pom，工程打包不会打成 Jar 包 -->
    <packaging>pom</packaging>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <!-- 配置当前项目编译 jdk 版本信息 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```