

Spring

Spring 是 Java EE 编程领域的一款轻量级的开源框架，由被称为“Spring 之父”的 Rod Johnson 于 2002 年提出并创立，它的目标就是要简化 Java 企业级应用程序的开发难度和周期。

Spring 自诞生以来备受青睐，一直被广大开发人员作为 Java 企业级应用程序开发的首选。时至今日，Spring 俨然成为了 Java EE 代名词，成为了构建 Java EE 应用的事实标准。

| 项目名称 | 描述 |
|-----------------|--|
| Spring Data | Spring 提供的数据库访问模块，对 JDBC 和 ORM 提供了很好的支持。通过它，开发人员可以使用一种相对统一的方式，来访问位于不同类型数据库中的数据。 |
| Spring Batch | 一款专门针对企业级系统中的日常批处理任务的轻量级框架，能够帮助开发人员方便的开发出健壮、高效的批处理应用程序。 |
| Spring Security | 前身为 Acegi，是 Spring 中较成熟的子模块之一。它是一款可以定制化的身份验证和访问控制框架。 |
| Spring Mobile | 是对 Spring MVC 的扩展，用来简化移动端 Web 应用的开发。 |
| Spring Boot | 是 Spring 团队提供的全新框架，它为 Spring 以及第三方库一些开箱即用的配置，可以简化 Spring 应用的搭建及开发过程。 |
| Spring Cloud | 一款基于 Spring Boot 实现的微服务框架。它并不是某一门技术，而是一系列微服务解决方案或框架的有序集合。它将市面上成熟的、经过验证的微服务框架整合起来，并通过 Spring Boot 的思想进行再封装，屏蔽调其中复杂的配置和实现原理，最终为开发人员提供了一套简单易懂、易部署和易维护的分布式系统开发工具包。 |

Spring Framework 的特点

Spring 框架具有以下几个特点。

方便解耦，简化开发

Spring 就是一个大工厂，可以将所有对象的创建和依赖关系的维护交给 Spring 管理。

方便集成各种优秀框架

Spring 不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如 Struts2、Hibernate、MyBatis 等）的直接支持。

降低 Java EE API 的使用难度

Spring 对 Java EE 开发中非常难用的一些 API（JDBC、JavaMail、远程调用等）都提供了封装，使这些 API 应用的难度大大降低。

方便程序的测试

Spring 支持 JUnit4，可以通过注解方便地测试 Spring 程序。

AOP 编程的支持

Spring 提供面向切面编程，可以方便地对程序进行权限拦截和运行监控等功能。

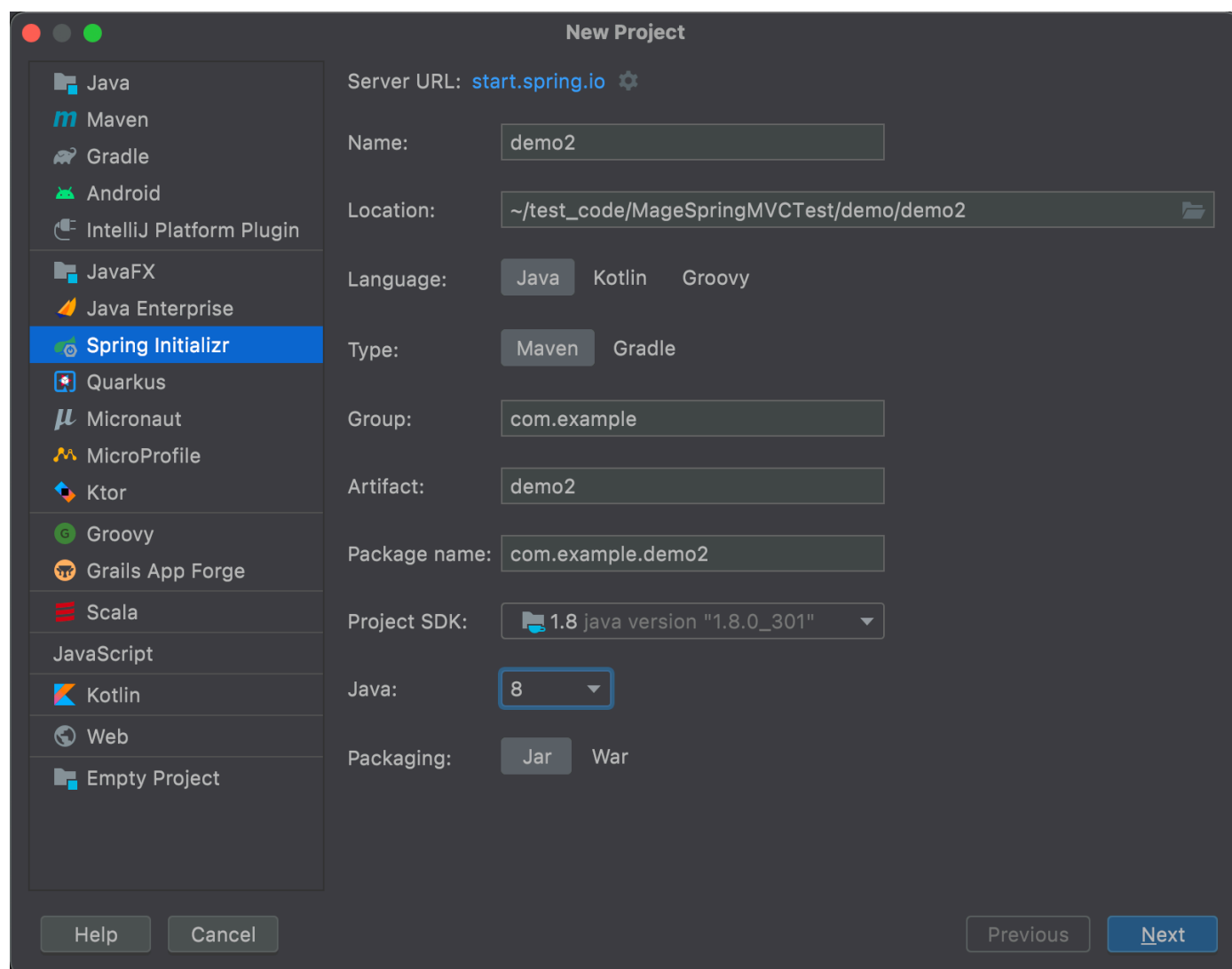
声明式事务的支持

只需要通过配置就可以完成对事务的管理，而无须手动编程。

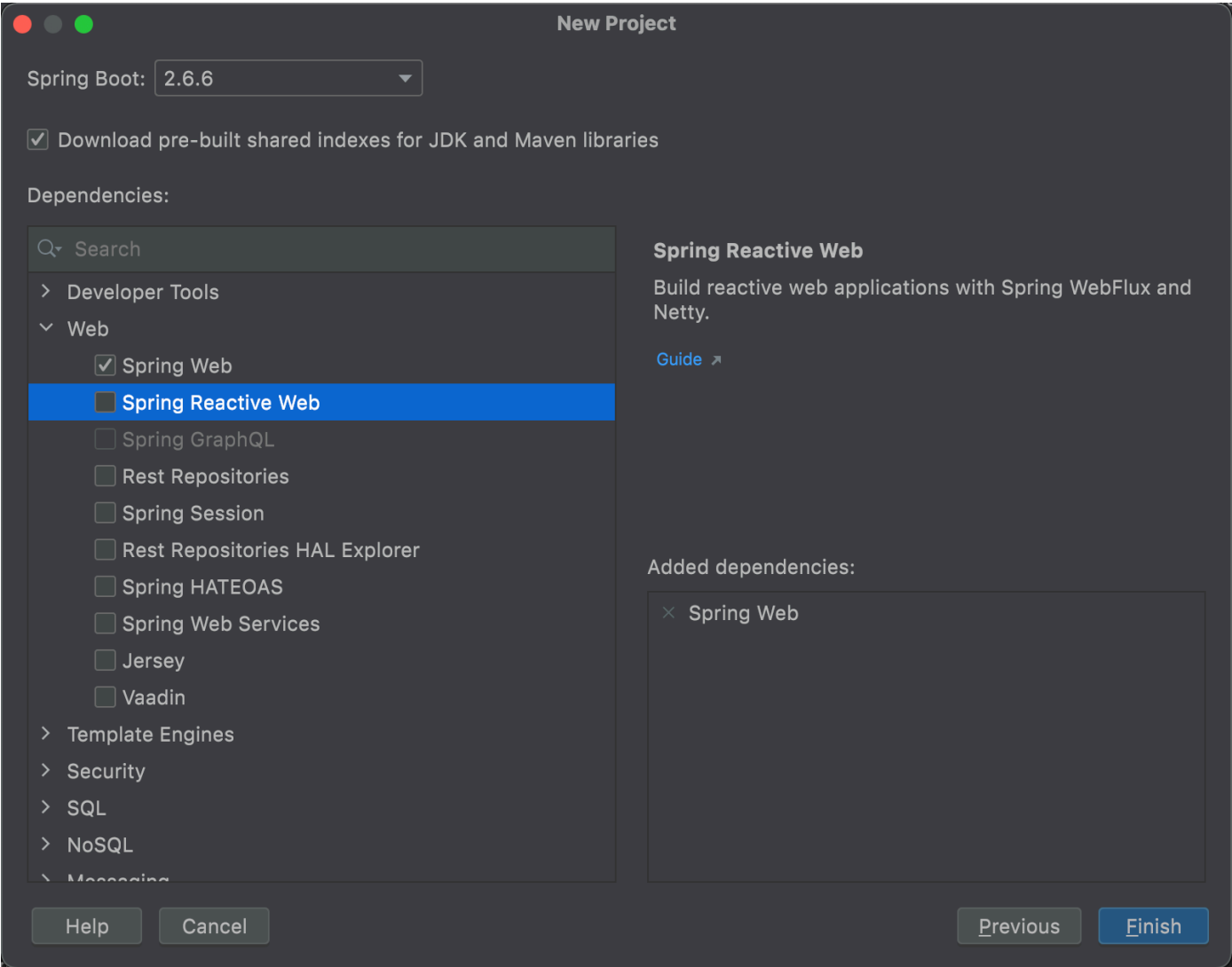
创建一个SpringBoot程序

初始化

新建项目，选择 Spring Initializr，自定义名称之后，点击下一步



选择其中的 Spring Web



创建成功后，可以看到一个初始的spring项目架构



创建接口

接下来，我们使用Spring创建一个 HelloWorld 项目

```
package com.example.springdemo1;

import org.springframework.boot.SpringApplication;
```

```

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@SpringBootApplication
public class SpringDemolApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringDemolApplication.class, args);
    }

    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String
name) {
        return String.format("Hello %s!", name);
    }
}

```

`hello()` 我们添加的方法旨在采用名为的String参数 `name`，然后将此参数与 `"Hello"` 代码中的单词组合。这意味着，如果您 `"Amy"` 在请求中将姓名设置为，则响应为 `"Hello Amy"`。

该 `@RestController` 注解告诉Spring，这个代码描述应该可在网上的端点。该 `@GetMapping("/hello")` 告诉Spring使用我们的 `hello()` 方法来回答这个问题被发送到请求 `http://localhost:8080/hello` 的地址。最后，`@RequestParam` 告诉Spring期望 `name` 请求中的值，但是如果不存在，默认情况下它将使用单词“World”。

访问接口 `hello`，并给参数为 `mage`

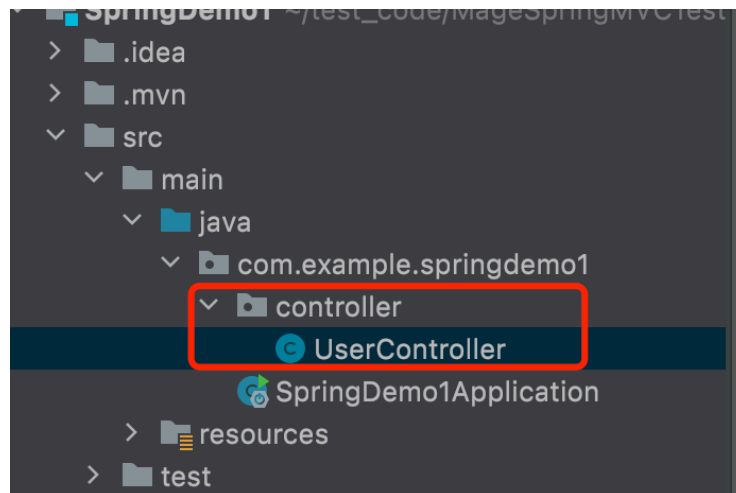
`http://127.0.0.1:8080/hello?name=mage`

← → ↻ ⓘ 127.0.0.1:8080/hello?name=mage

Hello mage!

对接口层进行拆分

新建 controller 目录



在 UserController 文件中写入 接口逻辑

```
package com.example.springdemo1.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String
name) {
        return String.format("Hello %s!", name);
    }
}
```

入口文件

```
package com.example.springdemo1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringDemo1Application {

    public static void main(String[] args) {
        SpringApplication.run(SpringDemo1Application.class, args);
    }
}
```

替换接口注解

使用PostMapping注解

```
@RestController
public class UserController {
    @PostMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String
name) {
        return String.format("Hello %s!", name);
    }
}
```

使用 `@PostMapping` 则只能使用 POST 方法，使用 GET 请求会报错

The image shows two side-by-side screenshots. The left screenshot is a web browser displaying a 'Whitelabel Error Page' with the message: 'This application has no explicit mapping for /error, so you are seeing this page. There was an unexpected error (type=Method Not Allowed, status=405)'. The right screenshot is a REST client interface showing a POST request to 'http://127.0.0.1:8080/hello' with a parameter 'name' set to 'mage'. The response body is 'Hello mage!'.

使用RequestMapping注解

```

@RestController
public class UserController {
    @RequestMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String
name) {
        return String.format("Hello %s!", name);
    }
}

```

使用 `RequestMapping` 注解时，可同时接收 GET 和 POST 请求

The screenshot shows a web browser on the left with the address bar at `127.0.0.1:8080/hello` and the page content "Hello World!". On the right is a REST client interface. The top bar shows a list of requests, with the current one being a POST to `http://127.0.0.1:8080/hello`. The "Body" tab is selected, showing a table of request parameters:

| KEY | VALUE | DESCRIPTION |
|--|-------|-------------|
| <input checked="" type="checkbox"/> name | mage | |
| Key | Value | Description |

Below the table, the "Body" tab shows the response in "Pretty" format:

```

1 Hello mage!

```

The status bar at the bottom indicates "Status: 200 OK", "Time: 50 ms", and "Size: 175 B".