Spring Boot 的快速入门（图文教程）

大家都知道，Spring Framework 是 Java/Spring 应用程序跨平台开发框架，也是 Java EE（Java Enterprise Edition）轻量级框架，其 Spring 平台为 Java 开发者提供了全面的基础设施支持。 虽然 Spring 基础组件的代码是轻量级，但其配置依旧是重量级的。

那是怎么解决了呢？当然是 Spring Boot，Spring Boot 提供了新的编程模式，让开发 Spring 应用变得更加简单方便。本书将会由各个最佳实践工程出发，涉及 Spring Boot 开发相关的各方面。下面先了解下 Spring Boot 框架。

1.1 Spring Boot 是什么

Spring Boot（Boot 顾名思义，是引导的意思）框架是用于简化 Spring 应用从搭建到开发的过程。应用开箱即用，只要通过一个指令，包括命令行 `java -jar`、`SpringApplication` 应用启动类、Spring Boot Maven 插件等，就可以启动应用了。另外，Spring Boot 强调只需要很少的配置文件，所以在开发生产级 Spring 应用中，让开发变得更加高效和简易。目前，Spring Boot 版本是 2.x 版本。

1.1.1 Spring Boot 2.x 特性

Spring Boot 2.x 具有哪些生产的特性呢？常用特性如下：

- SpringApplication 应用类
- 自动配置
- 外化配置
- 内嵌容器
- Starter 组件

还有对日志、Web、消息、测试及扩展等支持。

1.SpringApplication

`SpringApplication` 是 Spring Boot 应用启动类，在 `main()` 方法中调用 `SpringApplication.run()` 静态方法，即可运行一个 Spring Boot 应用。简单使用代码片段如下：

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

Spring Boot 运行的应用是独立的一个 Jar 应用，实际上在运行时启动了应用内部的内嵌容器，容器初始化 Spring 环境及其组件并启动应用。也可以使用 Spring Boot 开发传统的

应用，只要通过 Spring Boot Maven 插件将 Jar 应用转换成 War 应用即可。

2.自动配置

Spring Boot 在不需要任何配置情况下，就直接可以运行一个应用。实际上，Spring Boot 框架的 `spring-boot-autoconfigure` 依赖做了很多默认的配置项，即应用默认值。这种模式叫做“自动配置”。Spring Boot 自动配置会根据添加的依赖，自动加载依赖相关的配置属性并启动依赖。例如，默认用的内嵌式容器是 Tomcat ，端口默认设置为 8080。

3.外化配置

Spring Boot 简化了配置，在 application.properties 文件配置常用的应用属性。Spring Boot 可以将配置外部化，这种模式叫做“外化配置”。将配置从代码中分离外置，最明显的作用是只要简单地修改下外化配置文件，就可以在不同环境中，可以运行相同的应用代码。

4.内嵌容器

Spring Boot 启动应用，默认情况下是自动启动了内嵌容器 Tomcat，并且自动设置了默认端口为 8080。另外还提供了对 Jetty、Undertow 等容器的支持。开发者自行在添加对应的容器 Starter 组件依赖，即可配置并使用对应内嵌容器实例。

5.Starter 组件

Spring Boot 提供了很多“开箱即用”的 Starter 组件。Starter 组件是可被加载在应用中的 Maven 依赖项。只需要在 Maven 配置中添加对应的依赖配置，即可使用对应的 Starter 组件。例如，添加 `spring-boot-starter-web` 依赖，就可用于构建 REST API 服务，其包含了 Spring MVC 和 Tomcat 内嵌容器等。

开发中，很多功能是通过添加 Starter 组件的方式来进行实现。那么，Spring Boot 2.x 常用的 Starter 组件有哪些呢？

1.1.2 Spring Boot 2.x Starter 组件

Spring Boot 官方提供了很多 Starter 组件，涉及 Web、模板引擎、SQL 、NoSQL、缓存、验证、日志、测试、内嵌容器，还提供了事务、消息、安全、监控、大数据等支持。前面模块会在本书中一一介绍，后面这些模块本书不会涉及，如需自行请参看 Spring Boot 官方文档。

每个模块会有多种技术实现选型支持，来实现各种复杂的业务需求：

- Web：Spring Web、Spring WebFlux 等
- 模板引擎：Thymeleaf、FreeMarker、Groovy、Mustache 等
- SQL：MySQL 、H2 等
- NoSQL：Redis、MongoDB、Cassandra、Elasticsearch 等
- 验证框架：Hibernate Validator、Spring Validator 等
- 日志框架：Log4j2、Logback 等
- 测试：JUnit、Spring Boot Test、AssertJ、Mockito 等

- 内嵌容器：Tomcat、Jetty、Undertow 等

另外，Spring WebFlux 框架目前支持 Servlet 3.1 以上的 Servlet 容器和 Netty，各种模块组成了 Spring Boot 2.x 的工作常用技术栈，如图 1-1 所示。

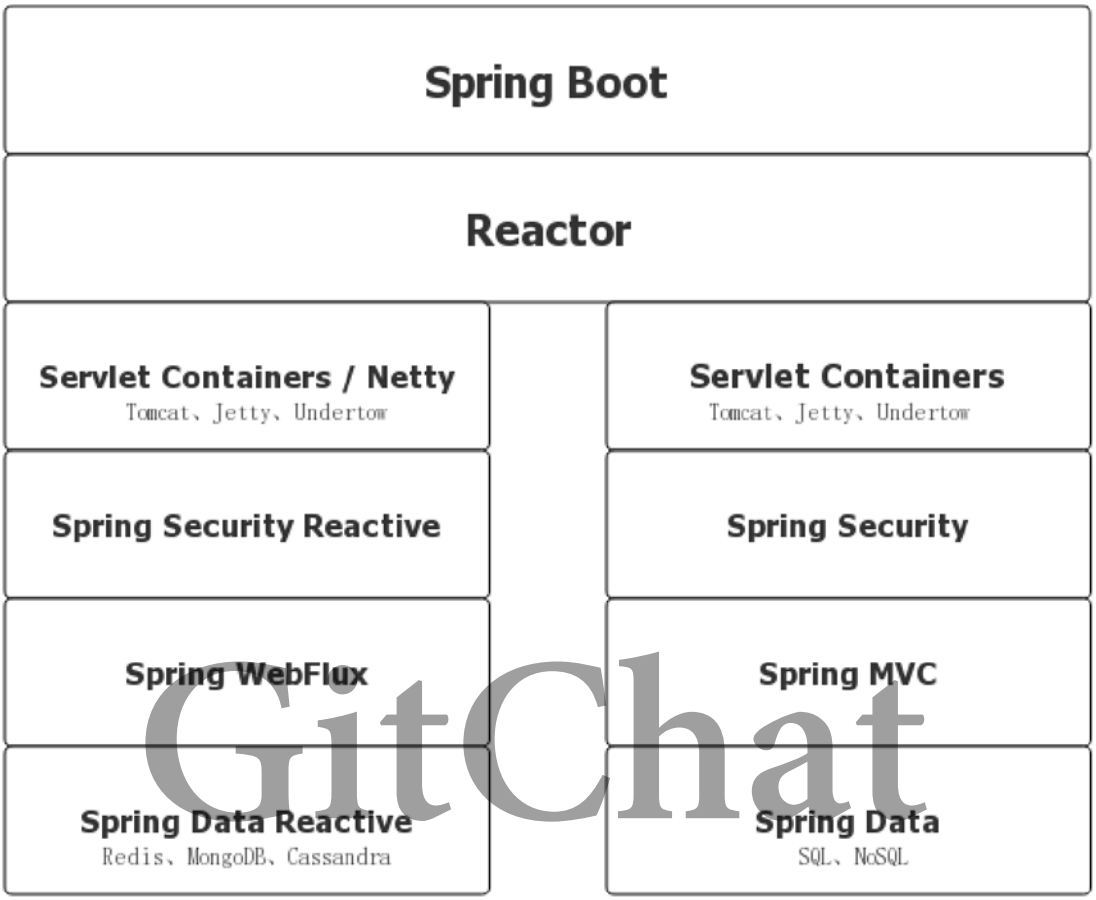


图 1-1 Spring Boot 技术架构

正如白猫黑猫，能抓住老鼠的就是好猫。在上述技术选型中，需要为公司业务的要求和团队技能选择最有效最合适的设计方案、架构和编程模型。

1.1.3 Spring Boot 应用场景

Spring Boot 模块众多，代表着应用场景也非常广泛，包括 Web 应用、SOA 及微服务等。在 Web 应用中，Spring Boot 封装了 Spring MVC 即可以提供 MVC 模式开发传统的 Web，又可以开发 REST API ，来开发 Web、APP、Open API 各种应用。在 SOA 及微服务中，用 Spring Boot 可以包装每个服务，本身可以提供轻量级 REST API 服务接口。也可以整合流行的 RPC 框架（Dubbo 等），提供 RPC 服务接口，只要简单地加入对应的 Starter 组件即可。在微服务实战中，推荐使用 Spring Cloud，是一套基于 Spring Boot 实现分布式系统的工具，适用于构建微服务。

上面从组件和特性两方面介绍了 Spring Boot 2.x，下面快速入门去了解 Spring Boot 2.x 的基本用法。

1.2 快速入门工程

在搭建一个 Spring Boot 工程应用前，需要配置好开发环境及安装好开发工具：

- JDK 1.8+：Spring Boot 2.x 要求 JDK 1.8 环境及以上版本。另外，Spring Boot 2.x 只兼容 Spring Framework 5.0 及以上版本。
- Maven 3.2+：为 Spring Boot 2.x 提供了相关依赖构建工具是 Maven，版本需要 3.2 及以上版本。使用 Gradle 则需要 1.12 及以上版本。本书使用 Maven 对 Spring Boot 工程进行依赖和构建管理。
- IntelliJ IDEA：IntelliJ IDEA（简称 IDEA）是常用的开发工具，也是本书推荐使用的，同样使用 Eclipse IDE，也能完成本书的实践案例。另外，本书的工程都会在 GitHub 上开源，如需要请自行安装 Git 环境。

> 注意：JDK 安装、Maven 安装、Git 安装和 IntelliJ IDEA 开发工具安装详解，见附录 A

安装环境虽然耗时，但磨刀不误砍柴工。下面开发 “Hello Spring Boot” 工程，大致分下面三个步骤：

- 创建工程
- 开发工程
- 运行工程

1.2.1 创建工程 “Hello Spring Boot”

在 IDEA 中，利用 Spring Initializr 插件进行创建名为 chapter-1-spring-boot-quickstart 工程。具体工程创建方式可见 1.3.1 章节。

第一步，打开 IDEA，选择新建工程按钮，然后选择左侧栏 Spring Initializr 选项。默认选择 JDK 版本和 Spring Initializr 的网站地址。如果是封闭式内网开发，也可以搭建一个 Spring Initializr 的内网服务，如图 1-2 所示。

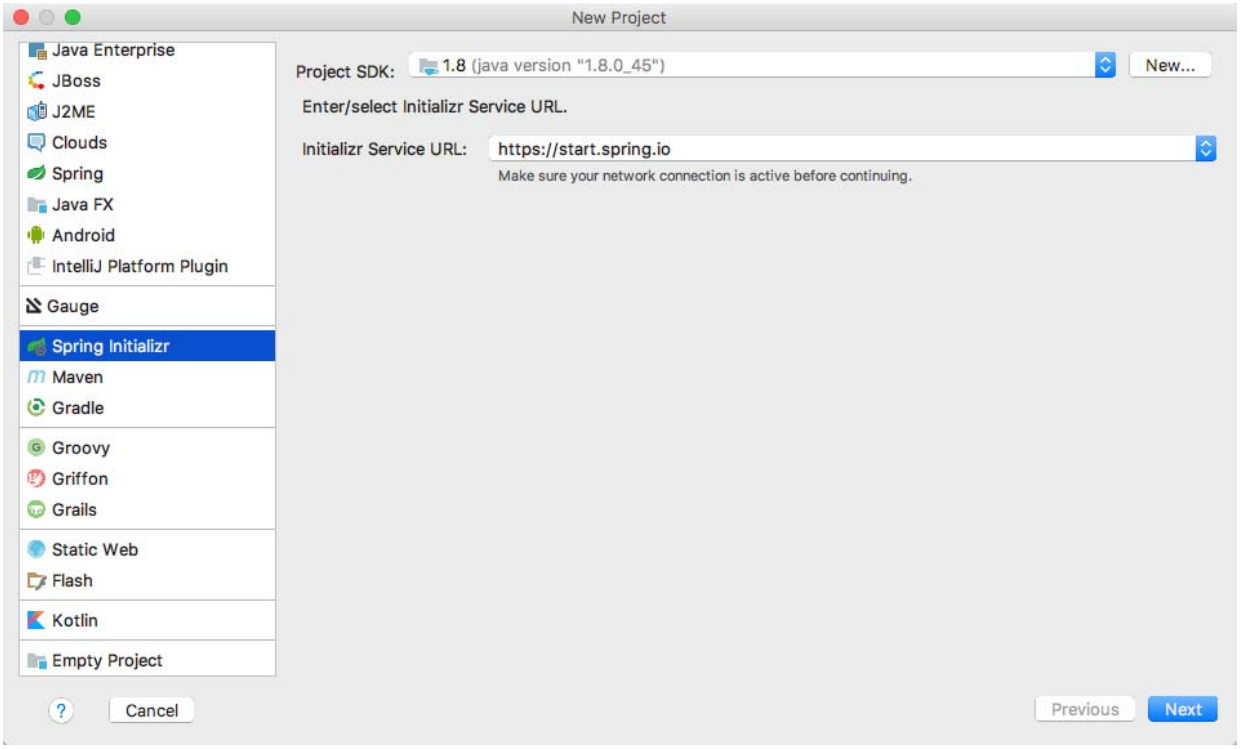


图 1-2 创建工程之选择使用 Spring Initializr

第二步，点击下一步，输入 Maven 工程信息。这里对应的 Maven 信息为：

- groupId：demo.springboot
- artifactId：chapter-1-spring-boot-quickstart
- version：1.0

这里还可以设置工程的名称和描述等，如图 1-3 所示。

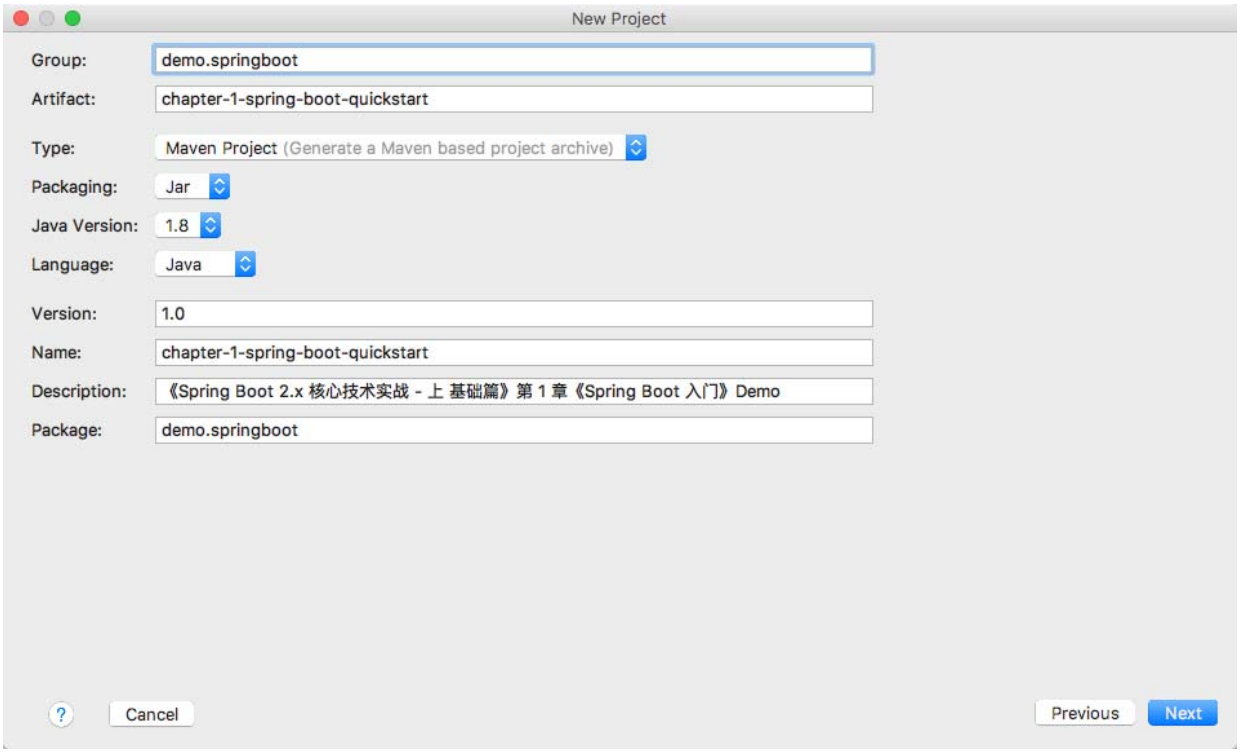


图 1-3 创建工程之新建工程信息

第三步，点击下一步，在依赖选择可视化操作中，下拉选择 Spring Boot 版本号和勾选工程需要的 Starter 组件和其他依赖。这里选择 Web 依赖，为了去实现一个简单的 REST API 服务，即访问 GET:/hello 会返回 “Hello，Spring Boot！” 的字符串，如图 1-4 所示。

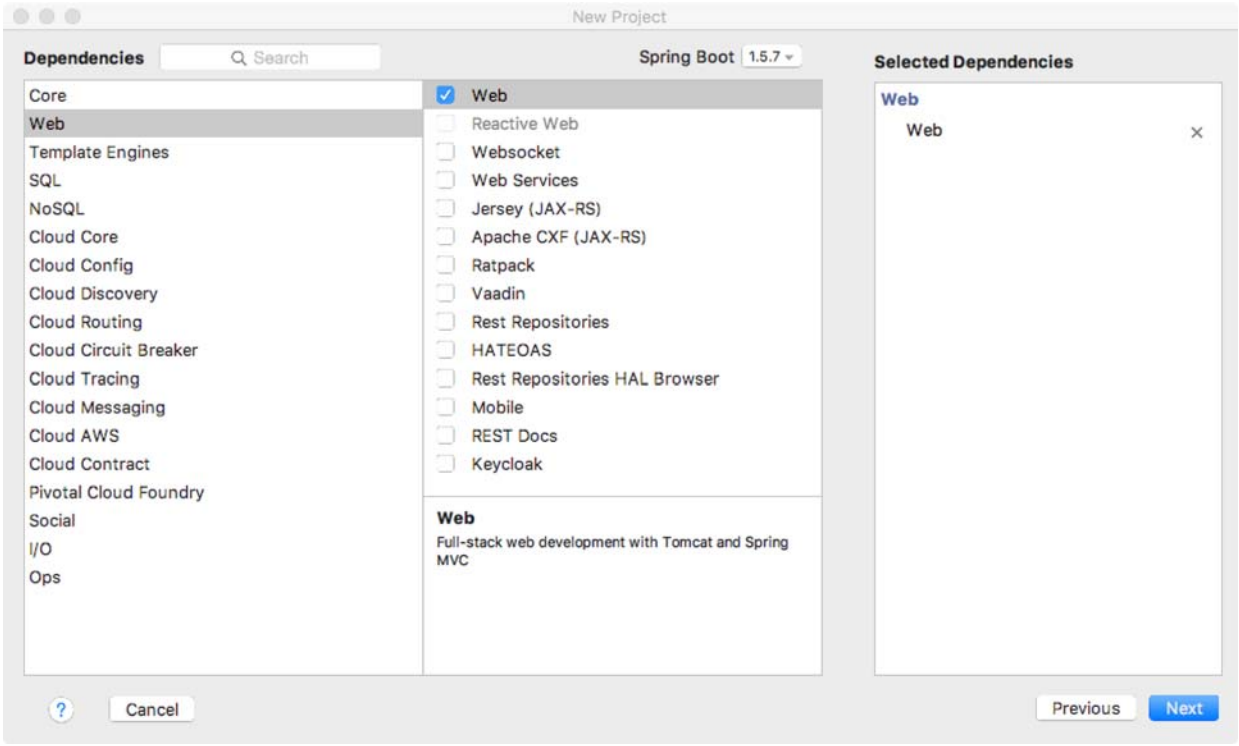


图 1-4 创建工程之选择 Web 依赖

这样就创建好名为 chapter-1-spring-boot-quickstart 工程，使用 IDEA 提示打开工程即可。

1.2.2 开发工程之 Pom 依赖

在 pom.xml 配置文件中， `parent` 节点配置是 Spring Boot 的 Parent 依赖，代码如下：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
    <relativePath/>
</parent>
```

`spring-boot-starter-parent` 依赖，是 Spring Boot 提供的一个特殊的 Starter 组件，本身没有任何依赖。

`spring-boot-starter-parent` 职责，一方面是用于提供 Maven 配置的默认值，即在 Spring Boot 2.x 中设置 JDK 1.8 为默认编译级别、设置 `UTF-8` 编码等。另一方面，其父依赖 `spring-boot-dependencies` 中的 `dependency-management` 节点提供了所有 Starter 组件依赖配置的缺省版本值，但不提供依赖本身的继承。这样的作用是使用各个组件依赖拿来即用，不需要指定 version 。

因为创建工程时，勾选 Web 依赖，Spring Initializr 会自动添加 Web 依赖，代码如下：

```
<!-- Web 依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

正如上面所说，这里只需要加入 groupId 和 artifactId 即可，不需要配置 version。

1.2.3 开发工程之应用启动类

在工程 src 目录中，已经自动创建了包目录 `demo.springboot` ，其包下自动创建了 Spring Boot 应用启动类，代码如下：

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

Spring Boot 应用启动类，是可以用来启动 Spring Boot 应用。其包含了 `@SpringBootApplication` 注解和 `SpringApplication` 类，并调用 `SpringApplication` 类的 `run()` 方法，就可以启动该应用。

1.@SpringBootApplication 注解

`@SpringBootApplication` 注解用标注启动类，被标注的类为一个配置类，并会触发自动配置和 Starter 组件扫描。根据源码可得，该注解配置了 `@SpringBootConfiguration` 、 `@EnableAutoConfiguration` 和 `@ComponentScan` 三个注解， `@SpringBootConfiguration` 注解又配置了 `@EnableAutoConfiguration` 。所以该注解的职责相当于结合了 `@Configuration` , `@EnableAutoConfiguration` 和 `@ComponentScan` 三个注解功能。

`@SpringBootApplication` 注解的职责如下：

- 在被该注解修饰的类中，可以用 `@Bean` 注解来配置多个 Bean 。应用启动时，Spring 容器会加载 Bean 并注入到 Spring 容器。
- 启动 Spring 上下文的自动配置。基于依赖和定义的 Bean 会自动配置需要的 Bean 和类。
- 扫描被 `@Configuration` 修饰的配置类。也会扫描 Starter 组件的配置类，并启动加载其默认配置

2.SpringApplication 类

大多数情况下，在 `main` 方法中调用 `SpringApplication` 类的静态方法 `run(Class, String[])` ，用来引导启动 Spring 应用程序。默认情况下，该类的职责会执行如下步

骤：

- 创建应用上下文 ApplicationContext 实例
- 注册 CommandLinePropertySource ，将命令行参数赋值到 Spring 属性
- 刷新应用上下文，加载所有单例
- 触发所有 CommandLineRunner Bean

在实际开发中如果需要自定义创建高级的配置，可以通过 run(Class, String[]) 方法的第二个参数，并以 String 数组的形式传入。这是 run 几个重载方法的 API 文档：

3.API org.springframework.boot.SpringApplication

（1）static run(Class

1.2.4 开发工程之 Hello 控制层类

在工程中新建包目录 demo.springboot.web ，并在目录中创建名为 HelloController 的控制层类，代码如下：

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloController {

    @RequestMapping(value = "/hello",method = RequestMethod.GET)
    @ResponseBody
    public String sayHello() {
        return "Hello，Spring Boot! ";
    }
}
```


上面定义了简单的 REST API 服务，即 GET:/hello 。表示该 Hello 控制层 sayHello() 方法会提供请求路径为 /hello 和请求方法为 GET 的 HTTP 服务接口。Spring 4.0 的注解 @RestController 支持实现 REST API 控制层。本质上，该注解结合了 @Controller 和 @ResponseBody 的功能。所以将上面 HelloController 可以改造升级成 HelloBookController ，代码如下：

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloBookController {

    @RequestMapping(value = "/book/hello",method =
```

```
    RequestMethod.GET)
        public String sayHello() {
            return "Hello，《Spring Boot 2.x 核心技术实战 - 上 基础
    篇》！";
        }
    }
```

以上核心注解是 Spring MVC 框架的知识点：

（1）@Controller 注解

`@Controller` 对控制层类进行标注，职责是使控制层可以处理 HTTP 请求，简单可以理解为，使控制层能接受请求，处理请求并响应。

（2）@RequestMapping 注解

`@RequestMapping` 对控制层类的方法进行标注，职责是标明方法对应的 HTTP 请求路由的关系映射。参数 value 主要请求映射地址，可接受多个地址。参数 method 标注 HTTP 方法，常用如：GET、POST、HEAD、OPTIONS、PUT、PATCH、DELETE、TRACE。默认是 GET HTTP 方法，在 GET 请求的情况下，可以缩写成 `@RequestMapping(value = "/book/hello")`。Spring 4 支持直接使用 `XXXMapping` 形式的注解，比如上面代码可以写成 `@GetMapping("/book/hello")`。

（3）@ResponseBody 注解

`@ResponseBody` 对控制层类的方法进行标注，职责是指定响应体为方法的返回值。上面代码中，案例是以字符串的形式返回，自然可以使用其他复杂对象作为返回体。

1.2.5 运行工程

一个简单的 Spring Boot 工程就开发完毕了，下面运行工程验证下。

1.Maven 编译工程

使用 IDEA 右侧工具栏，点击 Maven Project Tab ，点击使用下 Maven 插件的 `install` 命令。如图 1-5 所示：

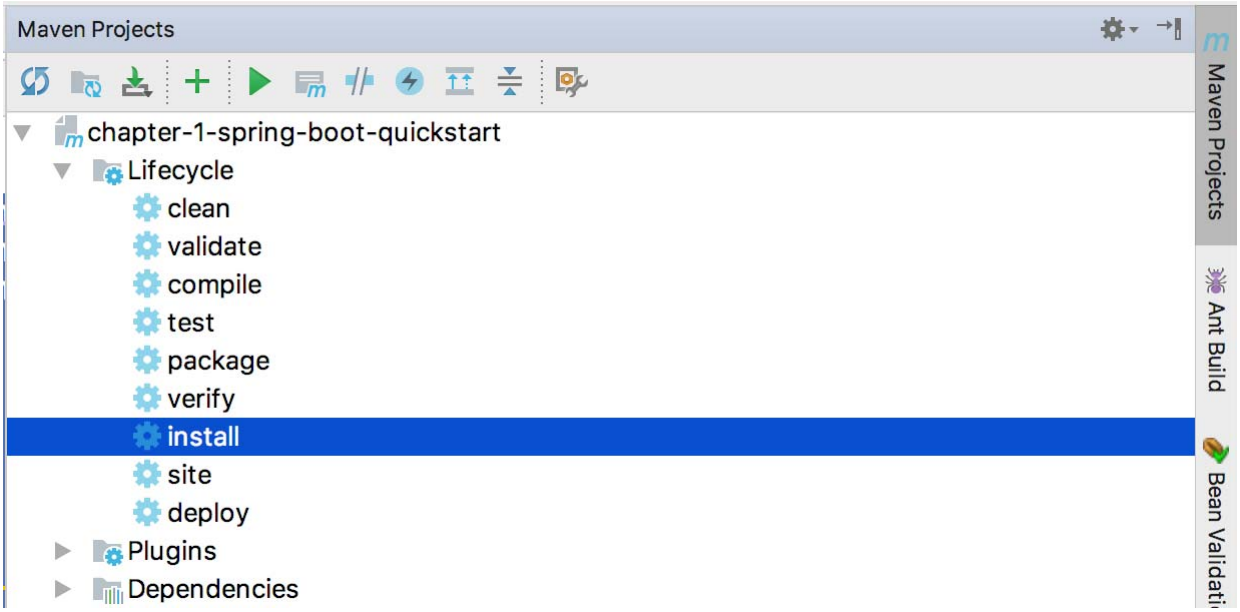


图 1-5 IDEA Maven 工具栏

或者使用命令行的形式，在工程根目录下，执行 Maven 清理和安装工程的指令：

```
cd chapter-1-spring-boot-quickstart
mvn clean install
```


在 target 目录中看到 chapter-1-spring-boot-quickstart-1.0.jar 文件生成，或者在编译的控制台中看到成功的输出：

```
... 省略
[INFO] ------------------------------------------------------------
--------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
--------------
[INFO] Total time: 01:30 min
[INFO] Finished at: 2017-10-15T10:00:54+08:00
[INFO] Final Memory: 31M/174M
[INFO] ------------------------------------------------------------
--------------
```

上面两种方式都可以成功编译工程。

2.运行工程

在 IDEA 中执行 `QuickStartApplication` 类启动，任意正常模式或者 Debug 模式。可以在控制台看到成功运行的输出：

```
... 省略
2017-10-15 10:05:19.994  INFO 17963 --- [           main]
```

```
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on
port(s): 8080 (http)
2017-10-15 10:05:20.000  INFO 17963 --- [           main]
demo.springboot.QuickStartApplication    : Started
QuickStartApplication in 5.544 seconds (JVM running for 6.802)
```

打开浏览器，访问 /hello 地址，会看到如图 1-6 所示的返回结果：

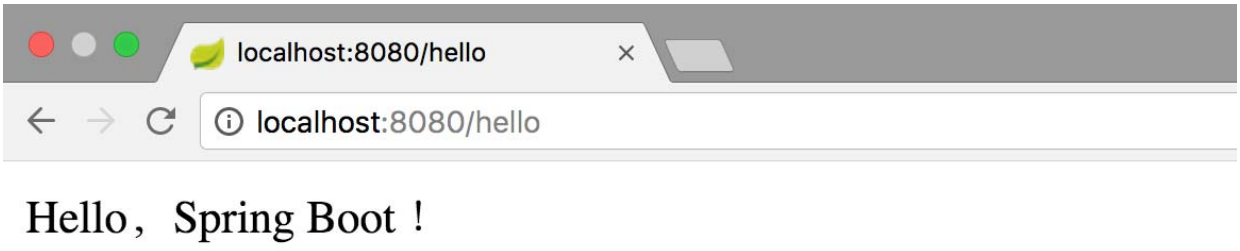


图 1-6 Hello 页面

再访问 /book/hello 地址，会看到如图 1-7 所示的返回结果：

图 1-7 Hello Book 页面

本章工程名为 chapter-1-spring-boot-quickstart，具体代码见本书对应的 GitHub。

1.3 Spring Boot 工程构建

快速入门中还没有详细介绍 Spring Boot 工程构建。工程构建包括创建工程、工程结构和运行工程等。

1.3.1 工程创建方式

Spring Boot Maven 工程，就是普通的 Maven 工程，加入了对应的 Spring Boot 依赖即可。Spring Initializr 则是像代码生成器一样，自动就给你出来了一个 Spring Boot Maven 工程。Spring Initializr 有两种方式可以得到 Spring Boot Maven 骨架工程：

1.start.spring.io 在线生成

Spring 官方提供了名为 Spring Initializr 的网站，去引导你快速生成 Spring Boot 应用。单击这里进入网站，操作步骤如下：

第一步，选择 Maven 或者 Gradle 构建工具，开发语言 Java 、Kotlin 或者 Groovy，最后确定 Spring Boot 版本号。这里默认选择 Maven 构建工具、Java 开发语言和 Spring Boot 2.0.0。

第二步，输入 Maven 工程信息，即项目组 `groupId` 和名字 `artifactId` 。这里对应 Maven 信息为：

- groupId：demo.springboot
- artifactId：chapter-1-spring-boot-quickstart

这里默认版本号 version 为 0.0.1-SNAPSHOT 。三个属性在 Maven 依赖仓库是唯一标识的。

第三步，选择工程需要的 Starter 组件和其他依赖。最后点击生成按钮，即可获得骨架工程压缩包。如图 1-8 所示。

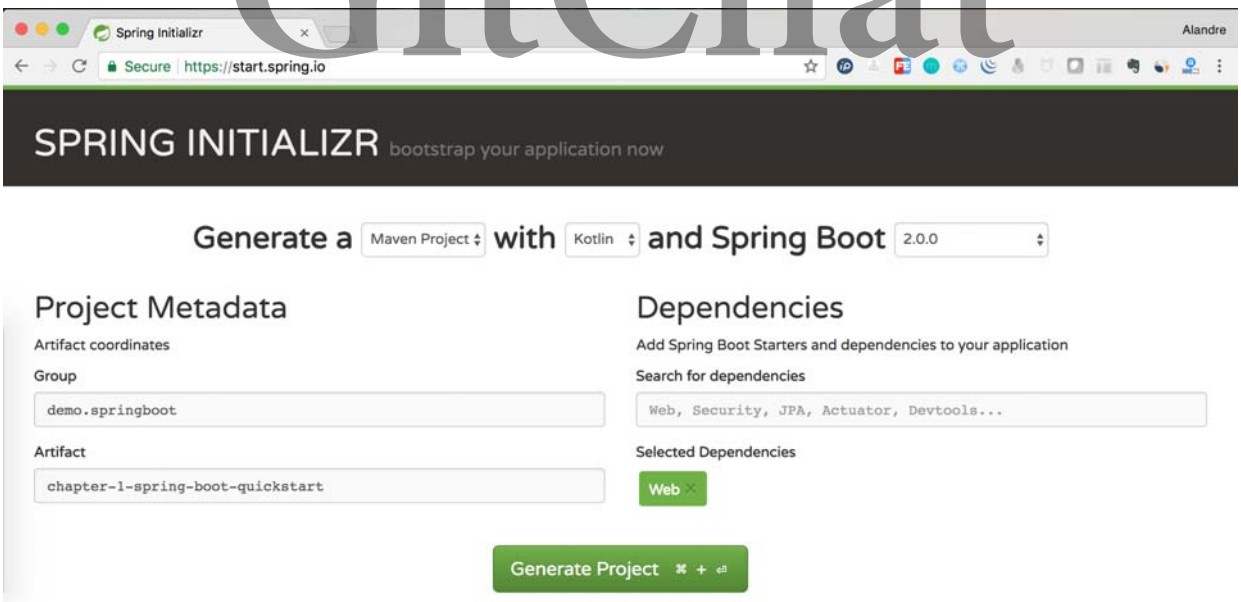


图 1-8 Spring Initializr 在线生成

2.IDEA 支持 Spring Initializr 生成

IDEA 支持 Spring Initializr 生成，这对于开发来说更进一步的省事，也是推荐的创建工程方式。

第一步，打开 IDEA，选择新建工程按钮，然后选择左侧栏 Spring Initializr 选项。默认选择 JDK 版本和 Spring Initializr 的网站地址。如果是封闭式内网开发，也可以搭建一个

Spring Initializr 的内网服务。

第二步，点击下一步，输入 Maven 工程信息。这里对应的 Maven 信息为：

- groupId：demo.springboot
- artifactId：chapter-1-spring-boot-quickstart
- version：1.0

这里还可以设置工程的名称和描述等。

第三步，点击下一步，在可视化操作中，下拉选择 Spring Boot 版本号和勾选工程需要的 Starter 组件和其他依赖。如图 1-9 所示。

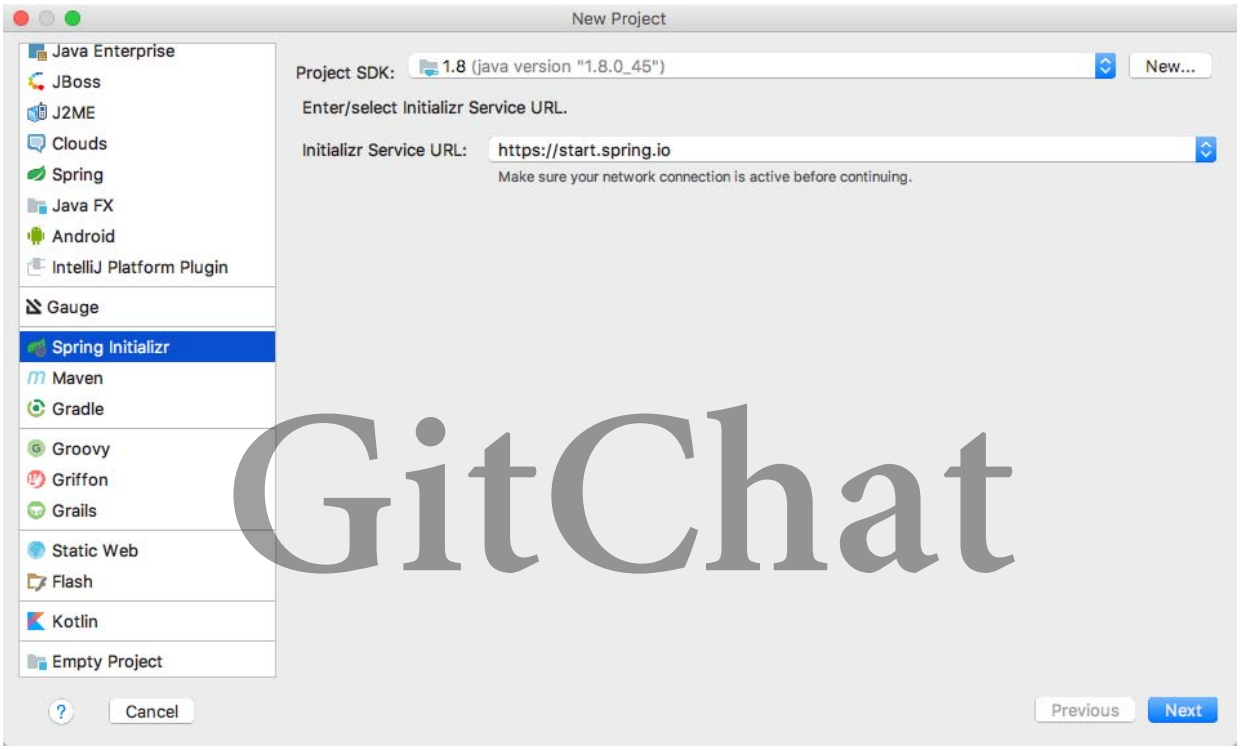


图 1-9 IDEA 支持 Spring Initializr 生成

1.3.2 工程结构

通过工程创建，得到的工程有默认的结构，其目录结构如下：

```
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── demo
    │   │       └── springboot
    │   │           └── QuickstartApplication.java
    │   └── resources
    │       ├── application.properties
    │       ├── static
    │       └── templates
```

```
    └── test
        └── java
            └── demo
                └── springboot
                    └── QuickstartApplicationTests.java
```

这是默认的工程结构，java 目录中是编写代码的源目录，比如三层模型大致会新建三个包目录，web 包负责 web 服务，service 包负责业务逻辑，domain 包数据源服务。对应 java 目录的是 test 目录，编写单元测试的目录。

resources 目录会有 application.properties 应用配置文件，还有默认生成的 static 和 templates 目录，static 用于存放静态资源文件，templates 用于存放模板文件。可以在 application.properties 中自定义配置资源和模板目录。

1.3.3 工程运行方式

上面使用应用启动类运行工程 “Hello Spring Boot”，这是其中一种工程运行方式。 Spring Boot 应用的运行方式很简单，下面介绍下这三种运行方式：

1. 使用应用启动类

在 IDEA 中直接执行应用启动类，来运行 Spring Boot 应用。日常开发中，会经常使用这种方式启动应用。常用的会有 Debug 启动模式，方便在开发中进行代码调试和 bug 处理。自然，Debug 启动模式会比正常模式稍微慢一些。

2. 使用 Maven 运行

通过 Maven 运行，需要配置 Spring Boot Maven 插件，在 pom.xml 配置文件中，新增 `build` 节点并配置插件 spring-boot-maven-plugin，代码如下：

```
<build>
    <plugins>
        <!-- Spring Boot Maven 插件 -->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

在工程根目录中，运行如下 Maven 命令来运行 Spring Boot 应用：

```
mvn spring-boot:run
```

实际调用的是 pom.xml 配置的 Spring Boot Maven 插件 spring-boot-maven-plugin ，上面执行了插件提供的 run 指令。也可以在 IDEA 右侧工具栏的 Maven Project Tab 中，找到

Maven 插件的 spring-boot-maven-plugin，执行相应的指令。所有指令如下：

```
# 生成构建信息文件
spring-boot:build-info
# 帮助信息
spring-boot:help
# 重新打包
spring-boot:repackage
# 运行工程
spring-boot:run
# 将工程集成到集成测试阶段，进行工程的声明周期管理
spring-boot:start
spring-boot:stop
```

3. 使用 Java 命令运行

使用 Maven 或者 Gradle 安装工程，生成可执行的工程 jar 后，运行如下 Java 命令来运行 Spring Boot 应用：

```
java -jar target/chapter-1-spring-boot-quickstart-1.0.jar
```


这里运行了 spring-boot-maven-plugin 插件编译出来的可执行 jar 文件。通过上述三种方式都可以成功运行 Spring Boot 工程，成功运行输出的控制台信息如图 1-10 所示。

图 1-10 控制台成功信息

1.4 安装使用 Spring Boot

在上面工程 “Hello Spring Boot” 中，使用了 Maven 方式去安装使用了 Spring Boot 。使用 Maven 或 Gradle 安装是推荐的方式。新的 Java 程序员或从未有过经验开发 Spring Boot 的开发者，推荐使用 Spring Boot CLI 安装学习更好。下面简单介绍下三种方式。

1.4.1 Maven 方式

Maven 是依赖管理的构建工具。类似其他依赖库使用一样，在 Maven 配置中加入 Spring Boot 相关依赖配置即可安装使用 Spring Boot 。Spring Boot 需要 Maven 3.2 及以上版本的支持。具体 Maven 安装介绍，详见官网 maven.apache.org。

Spring Boot 依赖使用 org.springframework.boot 作为其项目组 groupId 名称，Starter 组件的名字 artifactId 以 spring-boot-starter-xxx 形式命名。

注意，如果不想使用 spring-boot-starter-parent 父 Starter 组件，可以将 spring-boot-dependencies 作为工程依赖管理库，并指定 scope 为 import。代码如下：

```
<dependencyManagement>
      <dependencies>
         <dependency>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-dependencies</artifactId>
             <version>2.0.0</version>
             <type>pom</type>
             <scope>import</scope>
         </dependency>
     </dependencies>
</dependencyManagement>
```


1.4.2 Gradle 方式

Gradle 是依赖管理的构建工具。类似 Maven 方式一样，Spring Boot 需要 Gradle 4 的支持。具体 Gradle 安装介绍，单击这里详见官网。

1.4.3 Spring Boot CLI

Spring Boot CLI 是 Spring Boot Commad Line 的缩写，是 Spring Boot 命令行工具。在 Spring Boot CLI 可以跑 Groovy 脚本，通过简单的 Java 语法就可以快速而又简单的学习 Spring Boot 原型。

1.Spring Boot CLI 安装

打开 Spring Boot CLI 下载页面，下载需要的 spring-boot-cli-2.0.0-bin.zip 或者 spring-boot-cli-2.0.0-bin.tar.gz 依赖，并解压到安装目录，并指定其 bin 目录添加环境变量。
比如 Mac 环境下，代码如下：

```
export PATH=${PATH}:/spring-boot-cli-2.0.0.RELEASE/bin
```

比如 Windows 环境下，代码如下：

```
set PATH=D:\spring-boot-cli-2.0.0.RELEASE\bin;%PATH%
```

执行下面指令能输出对应的版本，用来验证是否安装成功，代码如下：

```
spring --version
```

在控制台中会出现成功的输出：

```
Spring CLI v2.0.0
```

另外，也支持 Homebrew、MacPorts 进行安装。

2.使用 Spring Boot CLI

安装好 Spring Boot CLI 基础环境后，运行 “Hello Spring Boot” 就更加简单了，新建 hello.groovy 文件，代码如下：

```
@RestController
public class HelloController {

    @RequestMapping(value = "/hello")
    public String sayHello() {
        return "Hello, Spring Boot! ";
    }
}
```


然后执行下面指令，进行编译运行应用：

```
spring run hello.groovy
```

也可以，通过 `--` 去外化配置属性值。比如配置端口号为 8081：`spring run hello.groovy -- --server.port=9000`，等控制台成功输出，打开浏览器，访问 /hello 地址，可以得到 “Hello，Spring Boot！” 的结果。

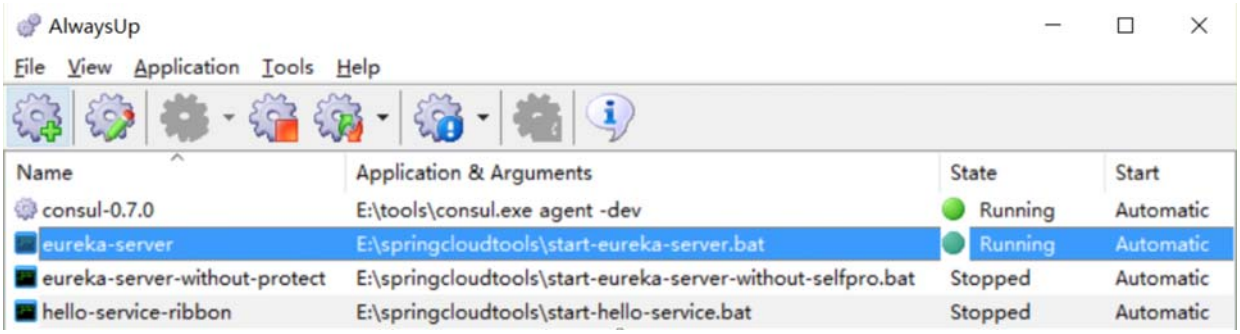
> 注意：具体如何使用 Spring CLI，详见官方使用文档。

1.5 服务器部署方式

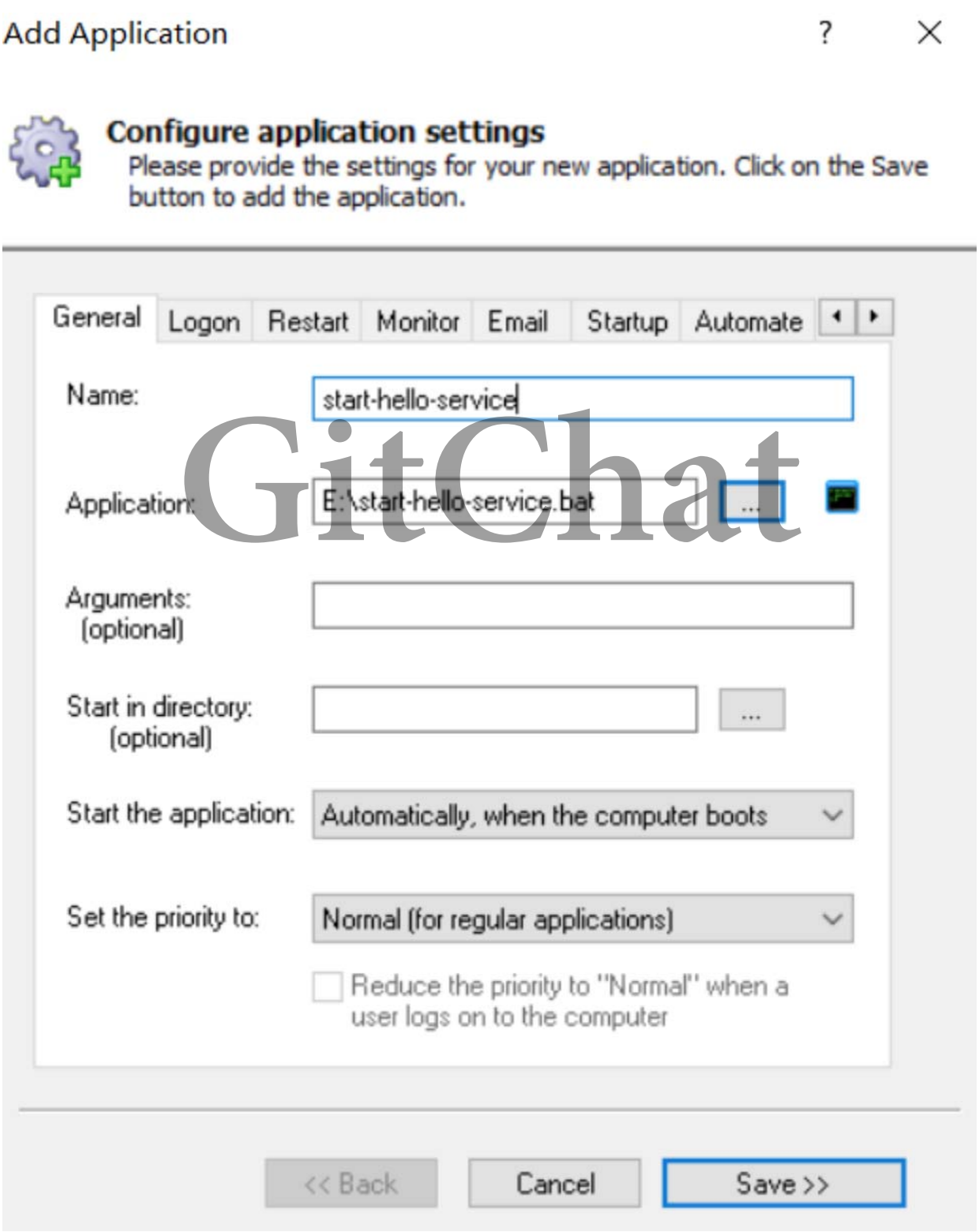
基础环境安装如上面说的，需要 JDK 环境、Maven 环境等。

1.5.1 Win 服务器

推荐使用 AlwaysUp：



使用方式也很简单：



1.5.2 Linux 服务器

推荐 yum 安装基础环境，比如安装 JDK：

```
yum -y list java*
yum -y install java-1.8.0-openjdk*
java -version
```

安装 Maven：

```
yum -y list apache-maven
sudo wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
sudo yum install -y apache-maven
mvn --v
```

Linux 使用 nohup 命令进行对后台程序的启动关闭。

关闭应用的脚本：stop.sh

```
#!/bin/bash
PID=$(ps -ef | grep yourapp.jar | grep -v grep | awk '{ print $2 }')
if [ -z "$PID" ]
then
    echo Application is already stopped
else
    echo kill $PID
    kill $PID
fi
```


启动应用的脚本：start.sh

```
#!/bin/bash
nohup java -jar yourapp.jar --server.port=8888 &
```

重启应用的脚本：stop.sh

```
#!/bin/bash
echo stop application
source stop.sh
echo start application
source start.sh
```

1.6 小结

从 Spring Boot 介绍开始，包括特性、Starter 组件以及应用场景，又通过快速入门工程出发，讲解了开发 Spring Boot 应用涉及到的 Pom 依赖、应用启动类以及控制层，然后讲解了工程构建周期的创建、结构及运行方式，此外还总结了安装使用 Spring Boot 的三种方式。

示例代码请单击这里下载。