**SpringBoot2核心技术**

适配器设计模式：  
一个接口A有1,2,3,4,5，五个方法，

然后类B要实现1,2,3，但是必须把不实现的方法也写上

类C要实现4,5方法，所以使用一个适配器，将A中的五个方法全部实现，然后B，C通过适配器只需要实现想要实现的方法即可

1. 基本概念

SpringBoot是Spring的一个高层框架，它的底层是Spring，是整合Spring生态圈的一站式框架，是简化Spring技术栈的快速开发脚手架（搭建好所有功能，只需要写业务）

**SpringBoot的优点：**

创建独立的Spring应用

内嵌web服务器

自动导入需要的相关依赖，并且保证这些依赖的版本之间相互兼容

自动配置Spring以及第三方功能，各种配置文件都自动配置

提供生产级别的监控，健康检查，外部化配置等等

无代码生成，无需编写XML

缺点：

版本迭代快

封装太深，内部原理复杂，难以精通

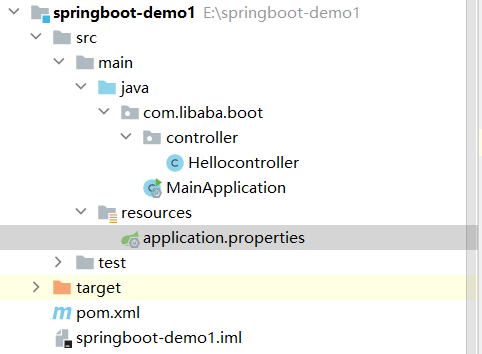
1. 搭建框架

版本：java1.8以上

Maven 3.3以上

添加parent和依赖，参考官方文档；

然后写类：



直接在主类中运行，就可以

在Resources中写一个配置文件即可；application.properties;

使用一个插件，可以将当前项目打成一个jar包，然后在目标服务器上运行即可：

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.7.0</version>

</plugin>

</plugins>

</build>

注意：这里必须配置版本，且版本必须和当前SpringBoot版本一致，否则报错

过程：

1. 创建一个maven工程，
2. 引入依赖

<!--parent和依赖-->

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.2</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

1. 创建主程序

@SpringBootApplication

public class MainApplication {

public static void main(String[] args) {

SpringApplication.run(MainApplication.class,args);

}

}

1. 编写业务

@RestController

public class HelloController {

@RequestMapping("/hello")

public String handle(){

return "Hello World";

}

}

1. 运行
2. 简化配置

只需在resources里写一个配置文件application.properties即可

1. 简化部署

使用下面插件，将项目打包成jar包，直接在目标服务器执行

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

1. SpringBoot的两个功能
2. **依赖管理**

1）父项目的父项目几乎声明了开发中用的所有依赖版本号

2）要自定义修改版本号：

在<parent>下面添加下面标签即可

<properties>

<mysql.version>5.6.3</mysql.version>

</properties>

3）Start场景启动器：所有官方启动器都遵循类似的命名模式；spring-boot-starter-\*，其中\*是特定类型的应用程序。

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

在：<https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using.build-systems.starters>

里查看官方的所有starter

\*-spring-boot-starter :是第三方自定义提供的场景启动器

所有场景启动器最底层的依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<version>2.7.2</version>

<scope>compile</scope>

</dependency>

1. **自动配置**

自动配好Tomcat

○ 引入Tomcat依赖。

○ 配置Tomcat

自动配好SpringMVC

○ 引入SpringMVC全套组件

○ 自动配好SpringMVC常用组件（功能）

自动配好Web常见功能，如：字符编码问题

○ SpringBoot帮我们配置好了所有web开发的常见场景

默认的包结构

○ 主程序所在包及其下面的所有子包里面的组件都会被默认扫描进来

○ 无需以前的包扫描配置

其他包必须和主程序在同一个包里，如果不在，可以使用下面注解配置

@SpringBootApplication(scanBasePackages="com.atguigu")或者@ComponentScan 指定扫描路径

即@SpringBootApplication

等同于

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan("com.atguigu.boot")

各种配置拥有默认值

○ 默认配置最终都是映射到某个类上，如：MultipartProperties

○ 配置文件的值最终会绑定每个类上，这个类会在容器中创建对象

按需加载所有自动配置项（也就是需要那些功能就手动引入这些配置，但是这些配置其实已经有了默认配置，只不过需要引入才能发挥作用）

○ 非常多的starter

○ 引入了哪些场景这个场景的自动配置才会开启

○ SpringBoot所有的自动配置功能都在 spring-boot-autoconfigure 包里面

○

1. **容器功能**

**2.1、组件添加**

**1、@Configuration**

**SpringBoot2响应式编程**