第1-4课：写一个 Hello World 来感受 Spring Boot

在学习新技术的时候我们都喜欢先写一个 Hello World 程序，一方面可以验证基础环境的搭建是否正确，另一方面可以快速了解整个开发流程。本节课我们就来学习 Spring Boot 的第一个 Hello World 程序。

什么是 Spring Boot

Spring 在官方首页这样介绍：

> BUILD ANYTHING.Spring Boot is designed to get you up and running as quickly as possible,with minimal upfront configuration of Spring.Spring Boot takes an opinionated view of building production ready applications.

中文翻译：Spring Boot 可以构建一切。Spring Boot 设计之初就是为了最少的配置，以最快的速度来启动和运行 Spring 项目。Spring Boot 使用特定的配置来构建生产就绪型的项目。

来一个 Hello World

（1）可以在 Spring Initializr 上面添加，也可以手动在 pom.xml 中添加：

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

pom.xml 文件中默认有个模块：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

- `<scope>test</scope>`，表示依赖的组件仅仅参与测试相关的工作，包括测试代码的编译和执行，不会被打包包含进去；
- `spring-boot-starter-test` 是 Spring Boot 提供项目测试的工具包，内置了多种测试工具，方便我们在项目中做单元测试、集成测试。

（2）编写 controller 内容

在目录 src\main\java\com\neo\web 下创建 HelloController：

GitChat

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String hello() {
        return "hello world";
    }

}
```

- @RestController 的意思是 Controller 里面的方法都以 JSON 格式输出，不需要有其他额外的配置；如果配置为 @Controller，代表输出内容到页面。
- @RequestMapping("/hello") 提供路由信息，"/hello" 路径的 HTTP Request 都会被映射到 hello() 方法上进行处理。

（3）启动主程序

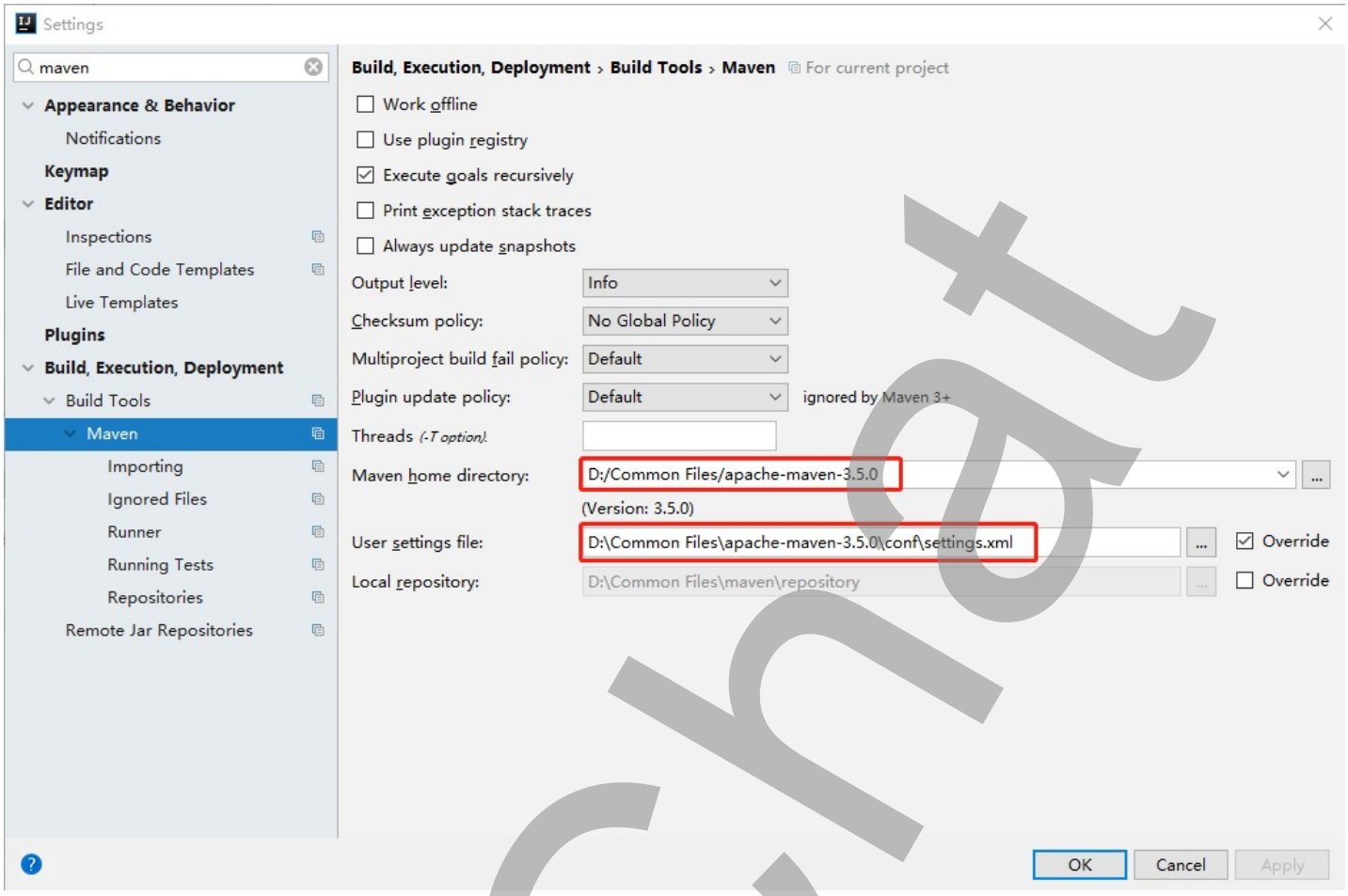
右键单击项目中的 HelloApplication l run 命令，就可以启动项目了，若出现以下内容表示启动成功：

```
2018-09-19 13:33:57.801  INFO 32996 --- [  restartedMain] o.s.b.w.embedded.tomcat.
TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-09-19 13:33:57.802  INFO 32996 --- [  restartedMain] com.neo.hello.HelloAppli
cation           : Started HelloApplication in 0.88 seconds
```

如果启动过程中出现 java ClassNotFoundException 异常，请检查 Maven 配置是否正确，具体如下。

- 检查 Maven 的 settings.xml 文件是否引入正确，请参考模板 settings.xml 文件。
- 检查 IDE 工具中的 Maven 插件是否配置为本机的 Maven 地址，如下图：

GitChat



Spring Boot 还提供了另外两种启动项目的方式：

- 在项目路径下，使用命令行 mvn spring-boot:run 来启动，其效果和上面“启动主程序”的效果是一致的；
- 或者将项目打包，打包后以 Jar 包的形式来启动。

```
# 进行项目根目录
cd ../hello
# 执行打包命令
mvn clean package
# 以 Jar 包的形式启动
java -jar target/hello-0.0.1-SNAPSHOT.jar
```

启动成功后，打开浏览器输入网址：http://localhost:8080/hello，就可以看到以下内容了：

```
hello world
```

> 开发阶段建议使用第一种方式启动，便于开发过程中调试。

（4）如果我们想传入参数怎么办？

请求传参一般分为 URL 地址传参和表单传参两种方式，两者各有优缺点，但基本都以键值对的方式将参数传递到后端。作为后端程序不用关注前端采用的那种方式，只需要根据参数的键来获取值，Spring 提供了很多种参数接收方式，今天我们了解最简单的方式：通过 URL 传参。

GitChat

只要后端处理请求的方法中存在参数键相同名称的属性，在请求的过程中 Spring 会自动将参数值赋值到属性中，最后在方法中直接使用即可。下面我们以 hello() 为例进行演示。

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String hello(String name) {
        return "hello world, " +name;
    }
}
```

重新启动项目，打开浏览器输入网址 http://localhost:8080/hello?name=neo，返回如下内容。

```
hello world, neo
```

到这里，我们的第一个 Spring Boot 项目就开发完成了，有没有感觉很简单？经过测试发现，修改 Controller 内相关的代码，需要重新启动项目才能生效，这样做很麻烦是不是？别着急，Spring Boot 又给我们提供了另外一个组件来解决。

热部署

热启动就需要用到我们在一开始就引入的另外一个组件：spring-boot-devtools。它是 Spring Boot 提供的一组开发工具包，其中就包含我们需要的热部署功能，在使用这个功能之前还需要再做一些配置。

添加依赖

在 pom.xml 文件中添加 spring-boot-devtools 组件。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-Devtools</artifactId>
  <optional>true</optional>
</dependency>
```

在 plugin 中配置另外一个属性 fork，并且配置为 true。

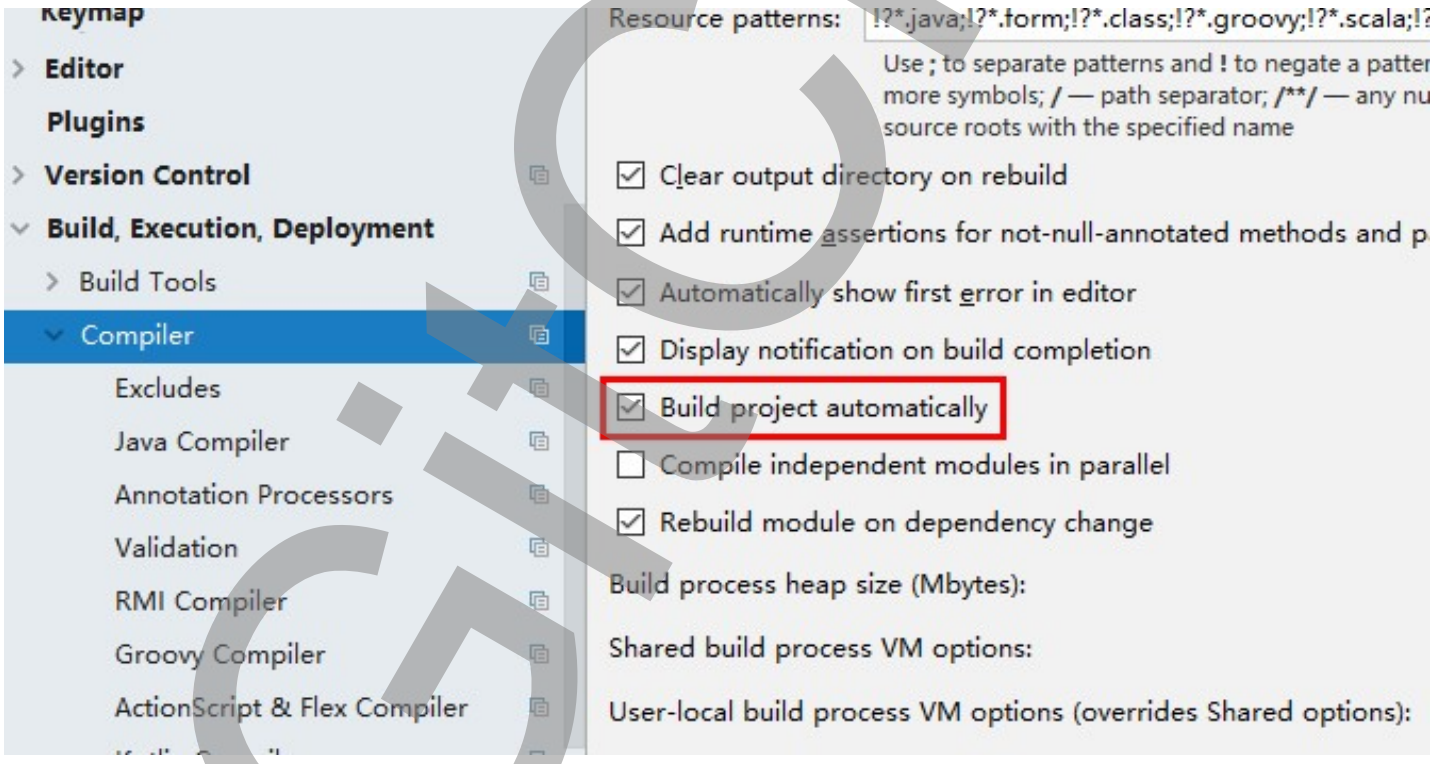
gitchat

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <fork>true</fork>
            </configuration>
        </plugin>
</plugins>
</build>
```

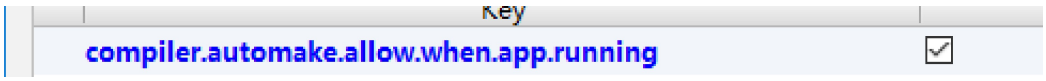
OK，以上的配置就完成了，如果你使用的是 Eclipse 集成开发环境，那么恭喜你大功告成了；如果你使用的是 IDEA 集成开发环境，那么还需要做以下配置。

配置 IDEA

选择 File | Settings | Compiler 命令，然后勾选 Build project automatically 复选框，低版本的 IDEA 请勾选 make project automatically 复选框。



使用快捷键 Ctrl + Shift + A，在输入框中输入 Registry，勾选 compile.automake.allow.when.app.running 复选框：



全部配置完成后，IDEA 就支持热部署了，大家可以试着去改动一下代码就会发现 Spring Boot 会自动重新加

载，再也不需要手动单击重新部署了。

为什么 IDEA 需要多配置后面这一步呢？因为 IDEA 默认不是自动编译的，需要我们手动去配置后才会自动编译，而热部署依赖于项目的自动编译功能。

> 该模块在完整的打包环境下运行的时候会被禁用，如果你使用 java-jar 启动应用或者用一个特定的 classloader 启动，它会认为这是一个“生产环境”。

单元测试

单元测试在我们日常开发中必不可少，一个优秀的程序员，单元测试开发也非常完善。下面我们看下 Spring Boot 对单元测试又做了哪些支持?

如果我们只想运行一个 hello World，只需要一个 @Test 注解就可以了。在 src/test 目录下新建一个 HelloTest 类，代码如下：

```
public class HelloTest {
    @Test
    public void hello(){
        System.out.println("hello world");
    }
}
```

右键单击“运行”按钮，发现控制台会输出：hello world。如果需要测试 Web 层的请求呢？Spring Boot 也给出了支持。

以往我们在测试 Web 请求的时候，需要手动输入相关参数在页面测试查看效果，或者自己写 post 请求。在 Spring Boot 体系中，Spring 给出了一个简单的解决方案，使用 MockMVC 进行 Web 测试，MockMVC 内置了很多工具类和方法，可以模拟 post、get 请求，并且判断返回的结果是否正确等，也可以利用 print() 打印执行结果。

GitChat

```
@SpringBootTest
public class HelloTest {

    private MockMvc mockMvc;

    @Before
    public void setUp() throws Exception {
        mockMvc = MockMvcBuilders.standaloneSetup(new HelloController()).build();
    }

    @Test
    public void getHello() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/hello?name=小明")
          .accept(MediaType.APPLICATION_JSON_UTF8)).andDo(print());
    }

}
```

- @Before 注解的方法表示在测试启动的时候优先执行，一般用作资源初始化。
- `.accept(MediaType.APPLICATION_JSON_UTF8))` 这句是设置 JSON 返回编码，避免出现中文乱码的问题。
- 注意导包需要加入以下代码，

```
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.*;
```

因为 print() 等方法都是 MockMvcResultHandlers 类的静态方法。

在类的上面添加 @SpringBootTest，系统会自动加载 Spring Boot 容器。在日常测试中，可以注入 bean 来做一些局部的业务测试。MockMvcRequestBuilders 可以支持 post、get 请求，使用 print() 方法会将请求和相应的过程都打印出来，如下：

GitChat

```
MockHttpServletRequest:
      HTTP Method = POST
      Request URI = /hello
       Parameters = {name=[小明]}
          Headers = {Accept=[application/json;charset=UTF-8]}
             Body = <no character encoding set>
    Session Attrs = {}

Handler:
             Type = com.neo.hello.web.HelloController
           Method = public java.lang.String com.neo.hello.web.HelloController.hell
o(java.lang.String)

...

MockHttpServletResponse:
           Status = 200
    Error message = null
          Headers = {Content-Type=[application/json;charset=UTF-8], Content-Length
=[19]}
     Content type = application/json;charset=UTF-8
             Body = hello world ,小明
    Forwarded URL = null
   Redirected URL = null
          Cookies = []
```

从返回的 `Body = hello world ,neo` 可以看出请求成功了。

当然每次请求都看这么多返回结果，不太容易识别，MockMVC 提供了更多方法来判断返回结果，其中就有判断返回值。我们将上面的 getHello() 方法稍稍进行改造，具体如下所示：

```
@Test
public void getHello() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post("/hello?name=小明")
            .accept(MediaType.APPLICATION_JSON_UTF8))/*.andDo(print())*/
            .andExpect(MockMvcResultMatchers.content().string(Matchers.containsStr
ing("小明")));
}
```

把刚才的 .andDo(print()) 方法注释掉，添加以下代码：

```
.andExpect(MockMvcResultMatchers.content().string(Matchers.containsString("小明")));
```

- MockMvcResultMatchers.content()，这段代码的意思是获取到 Web 请求执行后的结果；
- Matchers.containsString("小明")，判断返回的结果集中是否包含“小明”这个字符串。

GitChat

我们先将上面代码改为：

```
.andExpect(MockMvcResultMatchers.content().string(Matchers.containsString("微笑")));
```

然后执行 test 即可返回以下结果：

```
java.lang.AssertionError: Response content
Expected: a string containing "微笑"
     but: was "hello world ,小明"
...
```

> 抛出异常说明：期望的结果是包含“微笑”，结果返回内容是“hello world ,小明”，不符合预期。

接下来我们把代码改回：

```
.andExpect(MockMvcResultMatchers.content().string(Matchers.containsString("小明")));
```

再次执行测试方法，测试用例执行成功，说明请求的返回结果中包含了“小明”字段。

以上实践的测试方法，只是 spring-boot-starter-test 组件中的一部分功能，spring-boot-starter-test 对测试的支持是全方位的，后面会一一介绍到。

对比

我们简单做一下对比，使用 Spring Boot 之前和使用之后。

使用 Spring Boot 之前：

- 配置 web.xml，加载 Spring 和 Spring MVC
- 配置数据库连接、配置 Spring 事务
- 配置加载配置文件的读取，开启注解
- 配置日志文件
- …
- 配置完成之后部署 Tomcat 调试
- …

现在非常流行微服务，如果项目只是简单发送邮件的需求，我们也需要这样操作一遍。

使用 Spring Boot 之后，仅仅三步即可快速搭建起一个 Web 项目：

- 页面配置导入到开发工具中
- 进行代码编写
- 运行

GitChat

通过对比可以发现 Spring Boot 在开发阶段做了大量优化，非常容易快速构建一个项目。

总结

本课学习了如何搭建 Spring Boot 开发项目所需的基础环境，了解了如何开发一个 Spring Boot 的 Hello World 项目。使用 Spring Boot 可以非常方便地快速搭建项目，不用关心框架之间的兼容性、组件版本差异化等问题；项目中使用组件，仅仅添加一个配置即可，所以使用 Spring Boot 非常适合构建微服务。

> 点击这里下载源码。

GitChat