### JAVA第三阶段—DAY14-JAVA案例

# 1 Spring Boot入门案例

案例需求：请求Controller中的方法，并将返回值响应到页面

（1）依赖管理：

   <!--  
       所用的Spring Boot项目都会直接或者间接的继承spring-boot-starter-parent  
       1.指定项目的编码格式为UTF-8  
       2.指定JDK版本为1.8  
       3.对项目依赖的版本进行管理，当前项目再引入其他常用的依赖时就需要再指定版本号，避免版本冲突的问题  
       4.默认的资源过滤和插件管理  
       -->  
   <parent>  
       <groupId>org.springframework.boot</groupId>  
       <artifactId>spring-boot-starter-parent</artifactId>  
       <version>2.3.4.RELEASE</version>  
   </parent>  
​  
   <dependencies>  
       <!--引入Spring Web及Spring MVC相关的依赖-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-web</artifactId>  
       </dependency>  
   </dependencies>  
​  
   <!--可以将project打包为一个可以执行的jar-->  
   <build>  
       <plugins>  
           <plugin>  
               <groupId>org.springframework.boot</groupId>  
               <artifactId>spring-boot-maven-plugin</artifactId>  
           </plugin>  
       </plugins>  
   </build>

(2)启动类

/\*\*  
\* SpringBoot的启动类通常放在二级包中，比如：com.demo.SpringBootDemo1Application  
\* 因为SpringBoot项目在做包扫描，会扫描启动类所在的包及其子包下的所有内容。  
\*/  
//标识当前类为SpringBoot项目的启动类  
@SpringBootApplication  
public class SpringBootDemo1Application {  
​  
   public static void main(String[] args) {  
       //样板代码  
       SpringApplication.run(SpringBootDemo1Application.class,args);  
  }  
​  
}

(3)Controller

package com.demo.controller;  
​  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
​  
@RestController  
@RequestMapping("/hello")  
public class HelloController {  
​  
   @RequestMapping("/boot")  
   public String helloBoot(){  
       return "Hello Spring Boot";  
  }  
​  
}

# 2 快速构建

简单操作，无代码

# 3 单元测试

1．添加spring-boot-starter-test测试依赖启动器

在项目的pom.xml文件中添加spring-boot-starter-test测试依赖启动器，示例代码如下 ：

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

注意：使用Spring Initializr方式搭建的Spring Boot项目，会自动加入spring-boot-starter-test测试依赖启动器，无需再手动添加

2．编写单元测试类和测试方法

使用Spring Initializr方式搭建的Spring Boot项目，会在src.test.java测试目录下自动创建与项目主程序启动类对应的单元测试类

package com.demo;  
​  
import com.demo.controller.HelloController;  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.junit.runners.JUnit4;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
import org.springframework.test.context.junit4.SpringRunner;  
​  
/\*\*  
\* SpringJUnit4ClassRunner.class:Spring运行环境  
\* JUnit4.class:JUnit运行环境  
\* SpringRunner.class:Spring Boot运行环境  
\*/  
@RunWith(SpringRunner.class) //@RunWith:运行器  
@SpringBootTest //标记为当前类为SpringBoot测试类，加载项目的ApplicationContext上下文环境  
class Springbootdemo2ApplicationTests {  
   /\*\*  
    \* 需求：调用HelloController的hello方法  
    \*/  
​  
   @Autowired  
   private HelloController helloController;  
​  
​  
   @Test  
   void contextLoads() {  
       String result = helloController.hello();  
       System.out.println(result);  
  }  
​  
}

上述代码中，先使用@Autowired注解注入了DemoController实例对象，然后在contextLoads()方法中调用了DemoController类中对应的请求控制方法contextLoads()，并输出打印结果

# 4 热部署

使用步骤：

1）添加SpringBoot的热部署依赖启动器

2）开启Idea的自动编译

3）开启Idea的在项目运行中自动编译的功能

1．添加spring-boot-devtools热部署依赖启动器

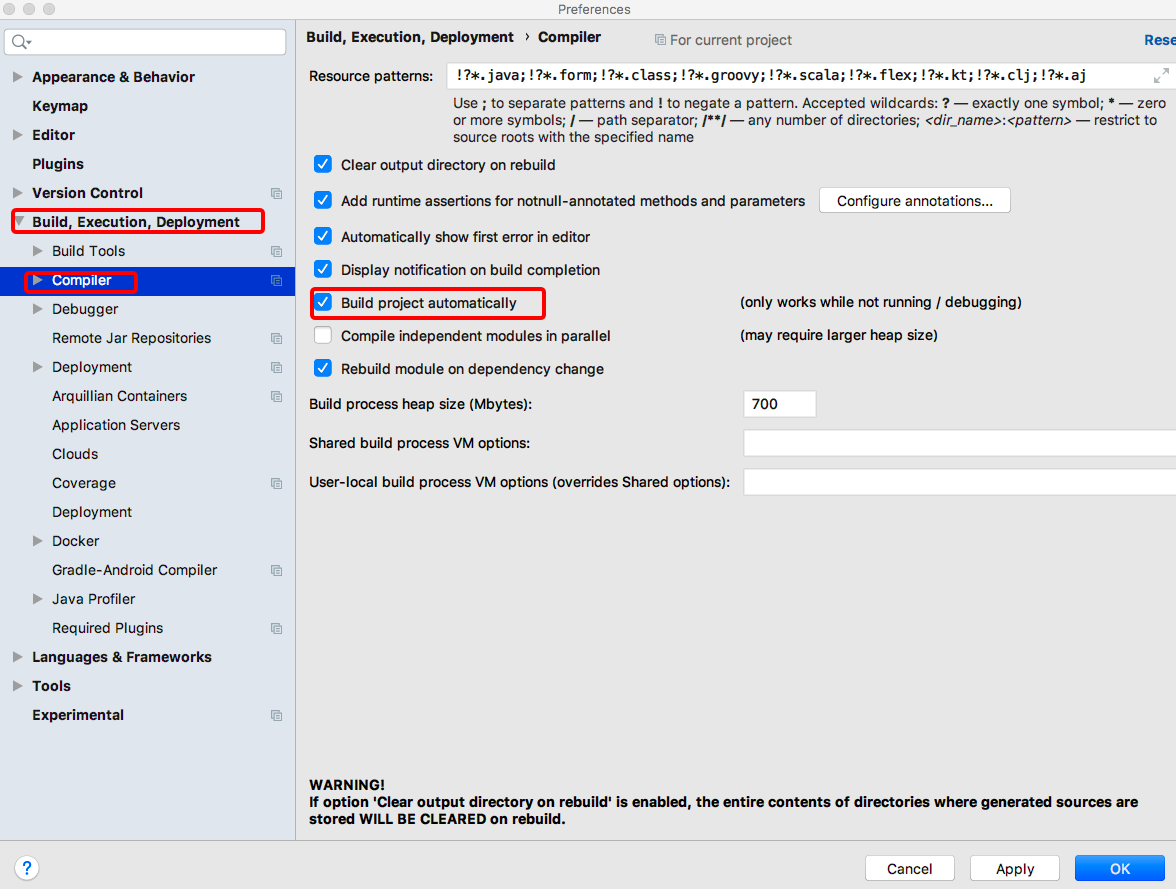
在Spring Boot项目进行热部署测试之前，需要先在项目的pom.xml文件中添加spring-boot-devtools热部署依赖启动器:

<!-- 引入热部署依赖 -->  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-devtools</artifactId>  
</dependency>

由于使用的是IDEA开发工具，添加热部署依赖后可能没有任何效果，接下来还需要针对IDEA开发工具进行热部署相关的功能设置

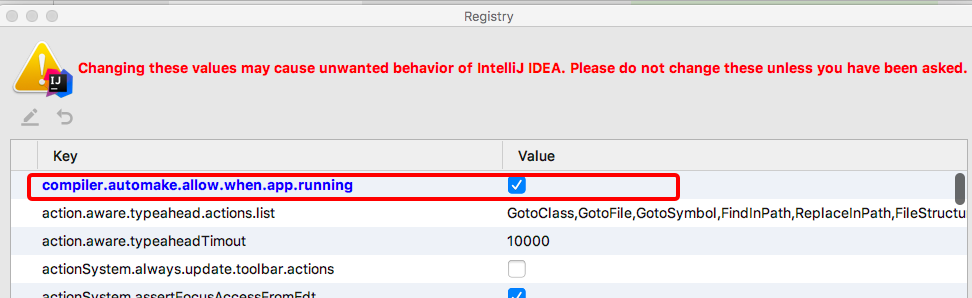
2. IDEA工具热部署设置

选择IDEA工具界面的【File】->【Settings】选项，打开Compiler面板设置页面



选择Build下的Compiler选项，在右侧勾选“Build project automatically”选项将项目设置为自动编译，单击【Apply】→【OK】按钮保存设置

在项目任意页面中使用组合快捷键“Ctrl+Shift+Alt+/”打开Maintenance选项框，选中并打开Registry页面，具体如图1-17所示



列表中找到“compiler.automake.allow.when.app.running”，将该选项后的Value值勾选，用于指定IDEA工具在程序运行过程中自动编译，最后单击【Close】按钮完成设置

# 5 properties配置文件

**server.port**=**8080  
server.servlet.context-path**=**/demo**

# 6 yaml配置文件

server:

port: 8080

servlet:

context-path: /hello

# 7 通过@ConfigurationProperties注入属性值

@Component

//将配置文件中所有以person开头的配置信息注入当前类中

//前提1：必须保证配置文件中person.xx与当前Person类的属性名一致

//前提2：必须保证当前Person中的属性都具有set方法

@ConfigurationProperties(prefix = "person")

public class Person {

private int id; //id

private String name; //名称

private List hobby; //爱好

private String[] family; //家庭成员

private Map map;

private Pet pet; //宠物

}

# 8 使用@Value注解

@Value注解是Spring框架提供的，用来读取配置文件中的属性值并逐个注入到Bean对象的对应属性中，Spring Boot框架从Spring框架中对@Value注解进行了默认继承，所以在Spring Boot框架中还可以使用该注解读取和注入配置文件属性值。使用@Value注入属性的示例代码如下

@Component  
public class Person {  
@Value("${person.id}")  
   private int id;        
}

上述代码中，使用@Component和@Value注入Person实体类的id属性。其中，@Value不仅可以将配置文件的属性注入Person的id属性，还可以直接给id属性赋值，这点是@ConfigurationProperties不支持的

演示@Value注解读取并注入配置文件属性的使用:

（1）在com.demo.pojo包下新创建一个实体类Student，并使用@Value注解注入属性

@Component  
public class Student {  
​  
     @Value("${person.id}")  
 private int id;  
   @Value("${person.name}")  
     private String name; //名称  
       
     //省略toString  
  }

Student类使用@Value注解将配置文件的属性值读取和注入。

从上述示例代码可以看出，使用@Value注解方式需要对每一个属性注入设置，同时又免去了属性的setXX()方法

（2）再次打开测试类进行测试

@Autowired  
private Student student;  
@Test  
public void studentTest() {  
System.out.println(student);  
}