SpringBoot常用注解

课程目标

- 核心注解@SpringBootApplication
- 了解SpringBoot其他常用注解

@SpringBootApplication

@SpringBootApplication

- @SpringBootApplication ：
  - 此注解是一个复合注解，包括
    - @SpringBootConfiguration
    - @EnableAutoConfiguration
    - @ComponentScan
  - 可以了解为以上三个注解(默认属性)的统一作用
  - 用于修饰main方法
  - 作用:
    - 是Sprnig Boot项目的核心注解，目的是开启自动配置

@SpringBootConfiguration

- **@SpringBootConfiguration:**
  - 继承自@Configuration，二者功能也一致.
  - @SpringBootConfiguration作用为标注当前类是配置类，并会将当前类内声明的一个或多个以@Bean注解标记的方法的实例纳入到srping容器中，并且实例名就是方法名。

- **使用@SpringBootConfiguration(@Configuration)和@Bean注解就可以创建一个简单的spring配置类，可以用来替代相应的xml配置文件。**
- 简单的说相当于application.xml配置文件.

- @Configuration注解,标识这个类可以使用Spring IoC容器作为bean定义的来源。

- @Bean注解告诉Spring，一个带有@Bean的注解方法将返回一个对象，该对象应该被注册为在Spring应用程序上下文中的bean。

```xml
<beans>
    <bean id = "car" class="com.test.Car">
        <property name="wheel" ref = "wheel"></property>
    </bean>
    <bean id = "wheel" class="com.test.Wheel"></bean>
</beans>
```

相当于:

```java
@Configuration
public class Conf {
    @Bean
    public Car car() {
        Car car = new Car();
        car.setWheel(wheel());
        return car;
    }
    @Bean
    public Wheel wheel() {
        return new Wheel();
    }
}
```

@EnableAutoConfiguration

- 使用 @Import（EnableAutoConfigurationImportSelector.class), 借助 EnableAutoConfigurationImportSelector类, @EnableAutoConfiguration 可以帮助 SpringBoot 应用将所有符合条件的 @Configuration 配置都加载到当前 SpringBoot 创建并使用的 IoC 容器.
- 简单的说,相当于XML配置中的<import />标签.

- 借助于Spring框架的工具类：SpringFactoriesLoader的支持，@EnableAutoConfiguration可以智能的自动配置,功效才得以大功告成。

- SpringFactoriesLoader属于Spring框架私有的一种扩展方案，其主要功能就是从指定的配置文件META-INF/spring.factories加载配置。

@ComponentScan

- @ComponentScan的功能其实就是自动扫描并加载符合条件的组件或bean定义，最终将这些bean定义加载到容器中。
- 相当于XML配置中的<context:component-scan basePackage="XXX">标签.

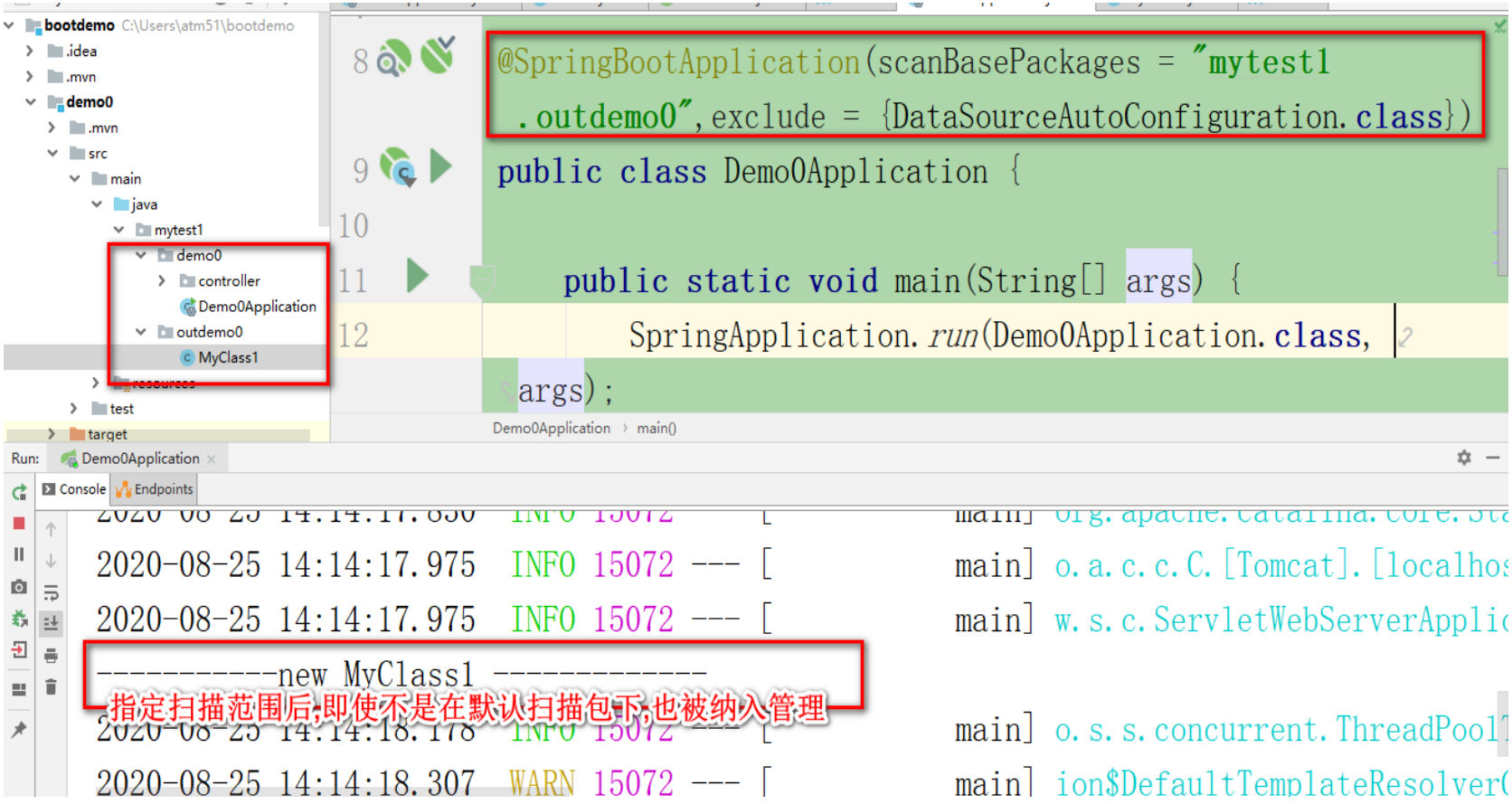
@ComponentScan

- 我们可以通过basePackages等属性指定@ComponentScan自动扫描的范围;如果不指定，则默认Spring框架实现从声明@ComponentScan所在类的包及其子包进行扫描，默认情况下是不指定的，所以SpringBoot的启动类最好放在root package下。

SpringBootApplication的常用参数

SpringBootApplication的常用参数

- scanBasePackages:指定扫描不在默认范围的包
- scanBasePackageClasses:指定扫描不在默认范围的包中的类。
- exclude：指定将扫描范围中的某个类排除在spring容器中


bootdemo C:\Users\atm51\bootdemo
.idea
.mvn
demo0
.mvn
src
main
java
mytest1
demo0
controller
Demo0Application
outdemo0
MyClass1
test
target
@SpringBootApplication(scanBasePackages = "mytest1
.outdemo0",exclude = {DataSourceAutoConfiguration.class})
public class Demo0Application {
public static void main(String[] args) {
SpringApplication.run(Demo0Application.class,
args);
Demo0Application › main()
Run: Demo0Application
Console
Endpoints
2020-08-25 14:14:17.975 INFO 15072 --- [main] o.a.c.c.C.[Tomcat].[localhos
2020-08-25 14:14:17.975 INFO 15072 --- [main] w.s.c.ServletWebServerApplic
----------new MyClass1 ----------
指定扫描范围后,即使不是在默认扫描包下,也被纳入管理
2020-08-25 14:14:18.307 WARN 15072 --- [main] ion$DefaultTemplateResolver(

SpringBoot常用注解

请求映射注解

- **@RestController 复合注解**
  - **用于修饰类**
  - 相当于@ResponseBody+@Controller合在一起的作用,RestController使用的效果是将方法返回的对象直接在浏览器上展示成json格式.

@GetMapping

- **用于将HTTP get请求映射到特定处理程序的方法注解**
- 是@RequestMapping(value = “/xxx”,method = RequestMethod.GET)的简写.

@PostMapping

- 是@RequestMapping(value = “/xxx”,method = RequestMethod.POST)的简写.
- **用于修饰方法**

导入配置文件注解

@PropertySource注解

- **用于修饰类**
- 引入单个properties文件:
  - @PropertySource(value = {"classpath : xxxx/xxx.properties"})
- 引入多个properties文件:
  - @PropertySource(value = {"classpath : xxxx/xxx.properties", "classpath : xxxx.properties"})

@ImportResource导入xml配置文件:用于修饰类

- 可以额外分为两种模式 相对路径classpath，绝对路径（真实路径）file
  - 注意：单文件可以不写value或locations，value和locations都可用
- 相对路径（classpath）
  - 引入单个xml配置文件：@ImportSource("classpath：xxx/xxxx.xml")
  - 引入多个xml配置文件：@ImportSource(locations={"classpath：xxxx.xml" , "classpath：yyyy.xml"})
- 绝对路径（file）
  - 引入单个xml配置文件：@ImportSource(locations= {"file：d:/hellxz/dubbo.xml"})
  - 引入多个xml配置文件：@ImportSource(locations= {"file：d:/hellxz/application.xml" , "file：d:/hellxz/dubbo.xml"})

@Import 导入额外的配置信息用于修饰类

- 功能类似XML配置的，用来导入配置类，可以导入带有@Configuration注解的配置类或实现了ImportSelector/ImportBeanDefinitionRegistrar。

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

全局异常处理

- 使用 @ControllerAdvice 实现全局异常处理，只需要定义类，添加该注解即可定义方式如下：

```
@ControllerAdvice
public class ExceptionController {

    @ExceptionHandler(Exception.class)
    @ResponseBody
    public String handleException(){
        System.out.println("----------发生异常----------");
        return "-------Exception Deal!-------";
    }
```

标注全局异常处理类

标注异常处理方法，可指定具体异常

- 在该类中，可以定义多个方法，不同的方法处理不同的异常，例如专门处理空指针的方法、专门处理数组越界的方法...，也可以直接向上面代码一样，在一个方法中处理所有的异常信息。
- @ExceptionHandler 注解用来指明异常的处理类型，即如果这里指定为 NullpointerException，则数组越界异常就不会进到这个方法中来。

- 其他注解如:
  - @Controller(@Service, @Repository, @Component)
  - @RequestBody
  - @RequestMapping
  - @PathVariable
  - @RequestParam
  - ...
  - 以上注解见Spring的注解详解

谢谢！