4.12 Spring Boot

4.12.3 约定优于配置

Spring Boot是快速使用Spring的方式。

约定优于配置：即遵守约定

4.12.4 Spring Boot的两大核心

Spring

优点：IOC和AOP

缺点：配置是重量级的，在applicationContext.xml中要编写很多配置；使用Maven工程，会存在依赖冲突的问题

Spring Boot

优点：Spring Boot利用**起步依赖**和**自动配置**很好的解决了Spring上述的问题。

起步依赖：将具备某种功能的坐标打包到一起，并提供一些默认的功能

自动配置：Spring Boot会自动地将一些配置类的Bean标签注入到IOC容器中。“自动”的表现形式就是我们只需要引我们想用功能的包，相关的配置我们完全不用管，springboot会自动注入这些配置bean，我们直接使用这些bean即可

4.12.5 Spring Boot案例实现

使用Spring Initializer（本质上是一个web项目）方式构建Spring Boot项目。

一定要把启动类拖到最外层包，因为Spring Boot只会对启动类所在的包进行注解扫描。

4.12.6 Spring Boot单元测试

Spring Boot对项目的单元测试提供了很好的支持，在使用时，需要提前在项目的pom.xml文件中添加**spring-boot-starter-test**测试依赖启动器，可以通过相关注解实现单元测试

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

4.12.7 Spring Boot热部署

在开发过程中，通常会对一段业务代码不断地修改测试，在修改之后往往需要重启服务，有些服务需要加载很久才能启动成功，这种不必要的重复操作极大的降低了程序开发效率。为此，SpringBoot框架专门提供了进行热部署的依赖启动器，用于进行项目热部署，而无需手动重启项目。

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

并打开build project automatically -> 在Registry中打开“compiler.automake.allow.when.app.running“

4.12.8 全局配置文件-application.properties

Spring Boot使用application.properties或者application.yaml的文件作为全局配置文件，该文件存放在src/main/resource目录或者类路径的/config，一般会选择resource目录。

application.properties

application.yaml

4.12.9 全局配置文件-application.yaml

JSON格式，看起来更加简洁。但优先级没有application.properties高

1. 可以使用.yaml或者.yml
2. key:(空格)value

针对不同类型的值：

1. 普通数据类型（基本数据类型+String）

   ```
   server:
     port: 8080
     servlet:
       context-path: /hello
   ```

2. 数组和单列集合

   ```
   person:
     hobby: [play,read,sleep]
   ```

3. map集合和对象

```yaml
person:
  map: {k1: v1,k2: v2}
```

4.12.10 配置文件属性值的注入

@ConfigurationProperties

Spring Boot提供的@ConfigurationProperties注解用来快速、方便地将配置文件中的自定义属性值**批量注入**到某个Bean对象的多个对应属性中。需要结合@Component

@Value

单个注入

4.12.11 使用@propertySource加载配置文件

4.12.12 使用@Configuration编写自定义配置类

在Spring Boot框架中，推荐使用配置类的方式向容器中添加和配置组件

在Spring Boot框架中，通常使用@Configuration注解定义一个配置类，Spring Boot会自动扫描和识别配置类，从而替换传统Spring框架中的XML配置文件。

当定义一个配置类后，还需要在类中的方法上使用@Bean注解进行组件配置，将方法的返回对象注入到Spring容器中，并且组件名称默认使用的是方法名，当然也可以使用@Bean注解的name或value属性自定义组件的名称

4.12.13 Spring Boot源码环境搭建

4.12.14～15 源码分析-依赖管理（再看）

1. 为什么导入dependency时不需要指定版本?

   因为父工程中已经对版本进行了统一的管理

2. spring-boot-starter-parent父依赖启动器的主要作用是进行版本统一管理，那么项目运行依赖的JAR包是从何而来的?

4.12.16 源码分析-自动配置（在看）

概念：能够在我们添加jar包依赖的时候，自动为我们配置一些组件的相关配置，我们无需配置或者只需要少量配置就能运行编写的项目

1. Spring Boot到底是如何进行自动配置的，都把哪些组件进行了自动配置?

   加载spring.factories，再根据条件过滤，决定配置哪些组件

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

```
@Target({ElementType.TYPE}) //注解的适用范围,Type表示注解可以描述在类、接口、注解或枚举中
@Retention(RetentionPolicy.RUNTIME) //表示注解的生命周期，Runtime运行时
@Documented //表示注解可以记录在javadoc中
@Inherited //表示可以被子类继承该注解
@SpringBootConfiguration // 标明该类为配置类
@EnableAutoConfiguration // 启动自动配置功能
@ComponentScan( // 包扫描器
  excludeFilters = {@Filter(
  type = FilterType.CUSTOM,
  classes = {TypeExcludeFilter.class}
), @Filter(
  type = FilterType.CUSTOM,
  classes = {AutoConfigurationExcludeFilter.class}
)}
)
public @interface SpringBootApplication {
...
}
```

4.12.17 Spring Boot整合Mybatis

4.12.18 注解的方式

4.12.19 XML的方式

4.12.20 整合SSM