springboot笔记

0. Springboot的发展

0.1 Spring1.x 时代

在Spring1.x时代，都是通过xml文件配置bean，随着项目的不断扩大，需要将xml配置分放到不同的配置文件中，需要频繁的在java类和xml配置文件中切换。

0.2 Spring2.x时代

随着JDK 1.5带来的注解支持，Spring2.x可以使用注解对Bean进行申明和注入，大大的减少了xml配置文件，同时也大大简化了项目的开发。

那么，问题来了，究竟是应该使用xml还是注解呢？

最佳实践： 应用的基本配置用xml，比如：数据源、资源文件等；
业务开发用注解，比如：Service中注入bean等；

0.3 Spring3.x到Spring4.x

从Spring3.x开始提供了Java配置方式，使用Java配置方式可以更好的理解你配置的Bean，现在我们就处于这个时代，并且Spring4.x和Spring boot都推荐使用java配置的方式。

0.4 Spring的Java配置方式

Java配置是Spring4.x推荐的配置方式，可以完全替代xml配置。

@Configuration 和 @Bean

Spring的Java配置方式是通过 @Configuration 和 @Bean 这两个注解实现的：
1、@Configuration 作用于类上，相当于一个xml配置文件；
2、@Bean 作用于方法上，相当于xml配置中的<bean>；

1. Springboot的hello-world

一个功能：

浏览器发送hello请求，服务器接受请求并处理，响应Hello World字符串

1.1 创建一个maven工程； （jar）

1.2 导入spring boot相关的依赖

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.9.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
```

1.3 编写一个主程序；启动Spring Boot应用

```
/**
 *  @SpringBootApplication 来标注一个主程序类，说明这是一个Spring Boot应用
 */
@SpringBootApplication
public class HelloWorldMainApplication {

    public static void main(String[] args) {

        // Spring应用启动起来
        SpringApplication.run(HelloWorldMainApplication.class,args);
    }
}
```

1.4 编写相关的controller

```
@Controller
public class HelloController {

    @ResponseBody
    @RequestMapping("/hello")
    public String hello(){
        return "Hello World!";
    }
}
```

1.5 运行主程序测试

1.6 简化部署

```
<!-- 这个插件，可以将应用打包成一个可执行的jar包； -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

将这个应用打成jar包，直接使用java -jar的命令进行执行；

2. HelloWorld探究

2.1 pom文件

2.1.1 父项目

依赖中有父项目依赖spring-boot-starter-parent

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.9.RELEASE</version>
</parent>
```

该父项目还依赖于其父项目

```
他的父项目是
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-dependencies</artifactId>
  <version>1.5.9.RELEASE</version>
  <relativePath>../../spring-boot-dependencies</relativePath>
</parent>
他来真正管理Spring Boot应用里面的所有依赖版本；
```

Spring Boot的版本仲裁中心；

以后我们导入依赖默认是不需要写版本；（没有在dependencies里面管理的依赖自然需要声明版本号

2.2 启动器

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

spring-boot-**starter-web**：

spring-boot-starter：spring-boot场景启动器；帮我们导入了web模块正常运行所依赖的组件；

Spring Boot将所有的功能场景都抽取出来，做成一个个的starters（启动器），只需要在项目里面引入这些starter相关场景的所有依赖都会导入进来。要用什么功能就导入什么场景的启动器

2.3 主程序类

```
/**
 *  @SpringBootApplication 来标注一个主程序类，说明这是一个Spring Boot应用
 */
@SpringBootApplication
public class HelloWorldMainApplication {

    public static void main(String[] args) {

        // Spring应用启动起来
        SpringApplication.run(HelloWorldMainApplication.class,args);
    }
}
```

2.3.1 @SpringBootApplication

Spring Boot应用标注在某个类上说明这个类是SpringBoot的主配置类，SpringBoot就应该运行这个类的main方法来启动SpringBoot应用

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited    //SpringBootApplication主要包含的三个注解
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

2.3.1.1 @SpringBootConfiguration

@**SpringBootConfiguration**:Spring Boot的**配置类**;

标注在某个类上，表示这是一个Spring Boot的配置类;

@**Configuration**:配置类上来**标注这个注解**;

配置类 ----- 配置文件；配置类也是容器中的一个组件；@Component

2.3.1.2 @EnableAutoConfiguration

以前我们需要配置的东西，Spring Boot帮我们自动配置；@**EnableAutoConfiguration**告诉SpringBoot开启自动配置功能；这样自动配置才能生效;

```java
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

@**AutoConfigurationPackage**：自动配置包

@**Import**(AutoConfigurationPackages.Registrar.class):

Spring的底层注解@Import，给容器中导入一个组件；导入的组件由AutoConfigurationPackages.Registrar.class;

将主配置类（@SpringBootApplication标注的类）的所在包及下面所有子包里面的所有组件扫描到Spring容器;

@**Import**(EnableAutoConfigurationImportSelector.class);

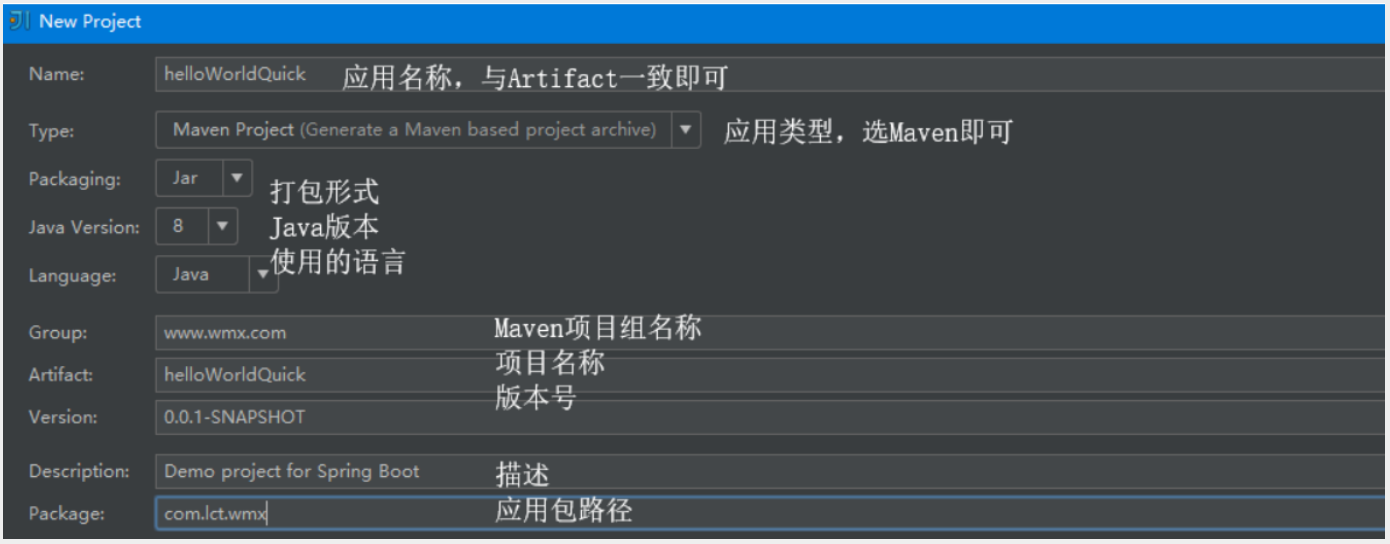
给容器中导入组件?

EnableAutoConfigurationImportSelector：导入哪些组件的选择器;

将所有需要导入的组件以全类名的方式返回；这些组件就会被添加到容器中;

会给容器中导入非常多的自动配置类（xxxAutoConfiguration）；就是给容器中导入这个场景需要的所有组件，并配置好这些组件;

3. 使用Spring Initializer快速创建SpringBoot项目



IDE都支持使用Spring的项目创建向导快速创建一个Spring Boot项目；
选择我们需要的模块；向导会联网创建Spring Boot项目；
默认生成的Spring Boot项目；
主程序已经生成好了，我们只需要我们自己的逻辑

- resources文件夹中目录结构
  - static：保存所有的**静态资源**； js css images；
  - templates：保存所有的**模板页面**； （Spring Boot默认jar包使用嵌入式的Tomcat，默认不支持JSP页 面）；可以使用模板引擎（freemarker、thymeleaf）；

- application.properties：Spring Boot应用的**配置文件**；可以修改一些默认设置；

4. 配置文件

SpringBoot使用一个**全局的配置文件**，配置文件名是固定的；

•application.properties

•application.yml

配置文件的作用：**修改**SpringBoot自动配置的**默认值**；SpringBoot在底层都给我们自动配置好；

YAML（YAML Ain't Markup Language）

YAML A Markup Language：是一个标记语言

YAML isn't Markup Language：不是一个标记语言；

标记语言：

以前的配置文件；大多都使用的是 **xxxx.xml**文件；

YAML：**以数据为中心**，比json、xml等更适合做配置文件；

YAML：配置例子

4.1 YMAL文件

4.1.1 基本语法

k:(空格)v：表示一对键值对（**空格必须有**）；

以**空格**的缩进来控制层级关系；只要是左对齐的一列数据，都是同一个层级的

```
server:
    port: 8081
    path: /hello
```

属性和值也是大小写敏感；

4.1.2 值的写法

4.1.2.1 普通的值的写法

k:(空格)v：字面直接来写；

字符串默认不用加上单引号或者双引号；

""：双引号；不会转义字符串里面的特殊字符；特殊字符会作为本身想表示的意思

name: "zhangsan \n lisi"：输出；zhangsan 换行 lisi

''：单引号；会转义特殊字符，特殊字符最终只是一个普通的字符串数据

name: 'zhangsan \n lisi'：输出；zhangsan \n lisi

4.1.2.2 对象、Map（属性和值）（键值对）

k: v：在下一行来写对象的属性和值的关系；注意缩进

对象还是k: v的方式

```
friends:
        lastName: zhangsan
        age: 20
```

行内写法：

```
friends: {lastName: zhangsan,age: 18}
```

4.1.2.2.3 数组（List、Set）

用**-** 值表示数组中的一个元素

```
pets:
 - cat
 - dog
 - pig
```

行内写法

```
pets: [cat,dog,pig]
```

4.1.2.2.4 配置占位符

```
${random.value}、${random.int}、${random.long}
${random.int(10)}、${random.int[1024,65536]}
```

```
person.last-name=张三${random.uuid}
person.age=${random.int}
person.birth=2017/12/15
person.boss=false
person.maps.k1=v1
person.maps.k2=14
person.lists=a,b,c
person.dog.name=${person.hello:hello}_dog
person.dog.age=15
```

4.2 YMAL文件的使用-给javaBean注入值

4.2.1 javaBean设置

```
/**
 * 将配置文件中配置的每一个属性的值，映射到这个组件中
 * @ConfigurationProperties：告诉SpringBoot将本类中的所有属性和配置文件中相关的配置进行绑定；
 *      prefix = "person"：配置文件中哪个下面的所有属性进行一一映射
 *
 * 只有这个组件是容器中的组件，才能容器提供的@ConfigurationProperties功能；
 *
 */
@Component
@ConfigurationProperties(prefix = "person")
public class Person {

    private String lastName;
    private Integer age;
    private Boolean boss;
    private Date birth;

    private Map<String,Object> maps;
    private List<Object> lists;
    private Dog dog;
```

我们可以导入配置文件处理器，以后编写配置就有提示了

```
<!--导入配置文件处理器，配置文件进行绑定就会有提示-->
        <dependency>
```

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

4.2.2 配置文件编写

```
person:
    lastName: hello
    age: 18
    boss: false
    birth: 2017/12/12
    maps: {k1: v1,k2: 12}
    lists:
      - lisi
      - zhaoliu
    dog:
      name: 小狗
      age: 12
```

4.3 @Value获取值和@ConfigurationProperties两种注入方法的对比

	@ConfigurationProperties	@Value
功能	批量注入配置文件中的属性	一个个指定
松散绑定（松散语法）	支持	不支持
SpEL	不支持	支持
JSR303数据校验	支持	不支持
复杂类型封装	支持	不支持

4.3.1 松散绑定

属性松散绑定 表示**驼峰式、下划线(_)、短横线(-)**

标准方式
person.firstName

方式一
大写用- person.first-name

方式二
大写用_ person.first_name

三种方式，都可以使用 推荐，属性书写方式 PERSON_FIRST_NAME

4.3.2 数据校验

在使用配置文件校验时，必须使用@configurationproperties注解，@value不支持该注解，在类上添加@validated注解，标识该类会被校验@Email表示校验类型为email

```
@Data
   @ConfigurationProperties(prefix = "Persion")
   @Validated
   @Component
   public class Persion implements Serializable {
      // @Value("${Persion.name}")

       @Email
       public String name;
   }
```

检验类型：

@Null	限制只能为null
@NotNull	限制必须不为null
@AssertFalse	限制必须为false
@AssertTrue	限制必须为true
@DecimalMax(value)	限制必须为一个不大于指定值的数字
@DecimalMin(value)	限制必须为一个不小于指定值的数字
@Digits(integer,fraction)	限制必须为一个小数，且整数部分的位数不能超过integer，小数部分的位数不能超过fraction
@Future	限制必须是一个将来的日期
@Max(value)	限制必须为一个不大于指定值的数字
@Min(value)	限制必须为一个不小于指定值的数字
@Past	验证注解的元素值（日期类型）比当前时间早
@Pattern(value)	限制必须符合指定的正则表达式
@Size(max,min)	限制字符长度必须在min到max之间
@NotEmpty	验证注解的元素值不为null且不为空（字符串长度不为0、集合大小不为0）
@NotBlank	验证注解的元素值不为空（不为null、去除首位空格后长度为0），不同于@NotEmpty，@NotBlank只应用于字符串且在比较时会去除字符串的空格
@Email	验证注解的元素值是Email，也可以通过正则表达式和flag指定自定义的email格式

4.4 加载指定的配置文件@PropertySource

不然加载全局配置文件

```
@PropertySource(value = {"classpath:person.properties"})
@Component
@ConfigurationProperties(prefix = "person")
//@Validated
public class Person {
    private String lastName;
    //@Value("#{11*2}")
    private Integer age;
    //@Value("true")
    private Boolean boss;
```

4.5添加组件

4.5.1 @ImportResource导入spring的xml配置文件(不推荐)

Spring Boot里面没有Spring的配置文件，我们自己编写的配置文件，也不能自动识别；

想让Spring的配置文件生效，加载进来；@**ImportResource**标注在一个配置类上

```
@ImportResource(locations = {"classpath:beans.xml"})
//导入Spring的配置文件让其生效
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans htt
p://www.springframework.org/schema/beans/spring-beans.xsd">


    <bean id="helloService" class="com.atguigu.springboot.service.HelloServ
ice"></bean>
</beans>
```

4.5.2 @configuration与@bean（推荐）

1、配置类@Configuration------>Spring配置文件

2、使用@Bean给容器中添加组件

```java
/**
 * @Configuration：指明当前类是一个配置类；就是来替代之前的Spring配置文件
 *
 * 在配置文件中用<bean><bean/>标签添加组件
 *
 */
@Configuration
public class MyAppConfig {

    //将方法的返回值添加到容器中；容器中这个组件默认的id就是方法名
    @Bean
    public HelloService helloService02(){
        System.out.println("配置类@Bean给容器中添加组件了...");
        return new HelloService();
    }
}
```

4.6 多profile(环境)文件的使用和加载顺序

我们在主配置文件编写的时候，文件名可以是application-{profile}.properties/yml
默认使用application.properties的配置；

4.6.1 yml支持多文档块形式(一个yml文件支持配置多个生产环境)