Spring Boot

> 简化Spring应用开发
> 整个Spring技术栈的一个大整合
> J2EE开发的一站式解决方案

Spring Boot的优点

- 快速创建独立运行的Spring项目以及与主流框架集成
- 使用嵌入式的Servlet容器，应用无需打成war包
- starter自动依赖与版本控制
- 大量的自动配置，简化开发，也可修改默认值
- 无需配置XML，无代码生成，开箱即用

- 准生产环境的运行时应用监控
- 与云计算的天然集成
- 缺点：入门容易，精通难

准备：

- 掌握Spring框架的使用
- 熟练使用Maven进行项目构建和依赖管理
- 熟练使用Eclipse或者IDEA

helloworld项目通过lifecycle中的package打包成jar包运行

项目文件

pom文件

1. 父项目

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.2</version>
        <relativePath/> <!-- lookup parent from repository -->
</parent>
```

他的父项目：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-dependencies</artifactId>
    <version>2.5.2</version>
</parent>
```

这一项才是真正管理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里面管理的依赖需要声明版本号）

2. 启动器

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

spring-boot-starter-web:

spring-boot-starter-web: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starter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。

主程序类，主入口类

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

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

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.
                @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class
public @interface SpringBootApplication {}
```

@**SpringBootConfiguration**: Spring Boot的配置类；标注在某个类上，表还是这是一个Spring Boot的配置类；
@**Configuration**:配置类上标注这个注解；配置类----配置文件；配置类也是容器中的一个组件，@Component

@**EnableAutoConfiguration**:开启自动配置功能；、
@**AutoConfigurationPackage**:自动配置包；
@import(AutoConfigurationPackage.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。
其功能：将主配置类（@SpringBootApplication标注的类）的所在包及下面所有自包里面的所有组件扫描到Spring容器
@ **import(EnableAutoConfigurationImportSelector.class)**：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）: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；
有了自动配置类，就免去了手动编写配置注入功能组件等的工作

Spring Initializer快速创建Spring Boot项目

- resources文件夹中目录结构
  - static: 保留所有静态资源； js, css, images;
  - templates: 保留所有的模板页面；（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应用配置文件

application.yml

- yaml语法：
  - 使用缩进表示层级关系
  - 以空格的缩进来控制层级关系
  - 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
  - 属性和值大小写敏感
  - Key: value 表示一对键值对（空格必须有）
  - 字符串默认不用加单引号或双引号
- 对象、map：
  - key: value

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

行内写法：

```yaml
friends: {lastName: zhangsan, age: 20}
```

- 数组：

```yaml
pets:
 - cat
 - dog
 - bird
```

行内写法：

```yaml
pets: [cat, dog, bird]
```

配置文件值注入

- 配置文件

```yaml
person:
  lastName: zhangsan
  age: 18
  boss: false
  birth: 2017/12/12
  maps: {k1: v1, k2: v2}
  lists:
    - heihei
    - haha
    - kuakua
  dog:
    name: 小白
    age: 2
```

```java
package com.raiuny.test.bean;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.List;
import java.util.Map;

/**
 * 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Spring Boot将本类中的所有属性和配置文件中相关的配置进行绑定
 * 只有这个组件是容器中的组件，才能用容器提供的@ConfigurationProperties的功能
 */
@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;

    @Override
    public String toString() {
        return "Person{" +
                "lastName='" + lastName + '\'' +
                ", age=" + age +
                ", boss=" + boss +
                ", birth=" + birth +
                ", maps=" + maps +
                ", lists=" + lists +
                ", dog=" + dog +
                '}';
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Integer getAge() {
        return age;
    }
```

```
    public void setAge(Integer age) {
        this.age = age;
    }

    public Boolean getBoss() {
        return boss;
    }

    public void setBoss(Boolean boss) {
        this.boss = boss;
    }

    public Date getBirth() {
        return birth;
    }

    public void setBirth(Date birth) {
        this.birth = birth;
    }

    public Map<String, Object> getMaps() {
        return maps;
    }

    public void setMaps(Map<String, Object> maps) {
        this.maps = maps;
    }

    public List<Object> getLists() {
        return lists;
    }

    public void setLists(List<Object> lists) {
        this.lists = lists;
    }

    public Dog getDog() {
        return dog;
    }

    public void setDog(Dog dog) {
        this.dog = dog;
    }

}
```

需要导入配置文件处理器，因此需要加入此依赖：

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

@Value获取值和@ConfigurationProperties的区别

- @ConfigurationProperties
  - 可以批量注入文件中的属性
  - 支持松散语法绑定：lastName等价于last-name
  - 不支持SpEL
  - 支持JSR303数据校验，@Validated注解Person，注入时候会校验 @Email可以注解某个属性，让其必须是指定格式
- @Value
  - 必须一个个指定
  - 支持SpEL，Spring表达式语言 `${person.lastName}` 但 `${person.maps}` 复杂类型的数据无法提取，而@ConfigurationPropeties可以
  - 不支持JSR303数据校验
- 如果我们只是在某个业务逻辑中需要获取一下配置文件中的某项值，则使用@Value。 `@Value(person.lastName)private String name;` 之后直接使用name即可。
- 如果我们专门编写了一个javaBean来和配置文件进行映射，我们就直接使用@ConfigurationProperties; `@Autowired Person person1;` 来得到配置的那个对象信息，不用一个个注入每一个属性，方便快捷。

@ConfigurationProperties

- 与@Bean结合，为属性赋值
- 与@PropertySource(只能用于properties文件)结合，读取指定文件

@ImportResource 读取外部配置文件

导入Spring的配置文件，让配置文件里面的内容生效

```
@ImportResource(locations = {"classpath:beans.xml"})
@PropertySource(value = {"classpath:person.properties"})
@Component
@ConfigurationProperties(prefix = "person")
public class Person {...}
```

这种方式太麻烦了，Spring有自己推荐的读取配置文件的方式

@PropertySource 加载指定的配置文件

```
@PropertySource(value = {"classpath:person.properties"})
@Component
@ConfigurationProperties(prefix = "person")
public class Person {...}
```

Spring推荐给容器中添加组件的方式

不推荐自己编写Spring的配置文件，而只需要自己编写一个配置类即可。

1. **配置类替代了Spring配置文件beans.xml**
2. **使用@Bean给容器添加组件，组件名为@Bean对应的方法名**

myAppConfig.java文件

```
package com.raiuny.test.config;

import com.raiuny.test.service.HelloService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @Configuration告诉SpringBoot该文件定义的类为一个配置类，来替代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 helloService(){
        System.out.println("配置类，通过@Bean给容器添加组件了");
        return new HelloService();
    }
}
```

需要自己定义一个配置类HelloService.

3. **通过ioc可以查看容器中是否添加了该组件**

```
@Autowired
ApplicationContext ioc;

@Test
public void testHelloService(){
    boolean b = ioc.containsBean("helloService");
    System.out.println(b);
}
```

配置文件占位符

RandomValuePropertySource 配置文件中使用随机数

- 配置文件中可以使用随机数，通过：
  - ${random.value}
  - ${random.int}
  - ${random.long}
  - ${random.int(10)}
  - ${random.int[1024,65536]}

属性配置占位符

```
app.name=MyApp
app.description=${app.name} is a Spring Boot application
```

- 可以在配置文件中引用前面配置过的属性（优先级：前面配置过的这里都能使用）
- ${app.name:默认值}来指定找不到属性时的默认值

Profile

1. 多Profile文件
   - 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；
   - spring.profiles.acive = _profile_name，激活指定profile；
   - 如果用yml文件，可以用代码块来做：

```yaml
server:
  port: 8081
spring:
  profiles:
    active: prod
---
server:
  port: 8083
spring:
  profiles: dev
---
server:
  port: 8090
spring:
  profiles: prod
```

- 命令行激活java -jar xxx.jar --spring.profiles.active=dev（program arguments）
- 虚拟机参数：(VM options) -Dspring.profiles.acive=dev