Spring Boot基础

> ## 本节目标
>
> 1.掌握Spring Boot核心注解 2.掌握Spring Boot配置

1. 核心注解

- **@SpringBootApplication**

在入口类上标注该注解，让Spring Boot自动给程序进行必要的配置，这个配置等同于： @Configuration ，@EnableAutoConfiguration 和 @ComponentScan 三个配置。

- **@EnableAutoConfiguration**

Spring Boot自动配置（auto-configuration）：尝试根据你添加的jar依赖自动配置Spring应用。例如：如果classpath下存在H2库，并且没有手动配置任何数据库连接beans，那么将自动配置一个内存型（in-memory）数据库”。

可以将@EnableAutoConfiguration或者@SpringBootApplication注解添加到一个@Configuration类上来选择自动配置。如果发现应用了你不想要的特定自动配置类，你可以使用@EnableAutoConfiguration注解的排除属性来禁用它们。

- **@Configuration**

相当于传统的xml配置文件，如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。

```
@Configuration
@EnableAutoConfiguration
public class AppConfiguration {

    @Bean(value = "hello")
    public String helloString() {
        return "hello";
    }

    @Bean(value = "welcome")
    public String welcomeString() {
        return "welcome";
    }

    @Bean(value = "properties")
    public Properties properties() {
        return new Properties();
    }
}
```

- **@PropertySource**

如果需要有自定义的属性文件需要加载，可以使用该注解进行注入，并用@Value配合使用。

```
@Component
@PropertySource(value = {"classpath:appconfig.properties"})
public class AppConfig {

    @Value("${app.config.host}")
    private String host;

    @Value("${app.config.port}")
    private Integer port;

    //此处忽略setter,getter方法
}
```

```
#src/main/resources/appconfig.properties
app.config.host=localhost
app.config.port=3306
```

- **@ImportResource**

用来加载xml配置文件。

```
@SpringBootApplication
//位于: src/main/resources/application-bean.xml
@ImportResource(locations = {"classpath:application-bean.xml"})
public class ExampleApplication {

    public static void main(String[] args) {
        new SpringApplicationBuilder(ExampleApplication.class)
                .bannerMode(Banner.Mode.CONSOLE)
                .run(args);
    }
}
```

- **@Bean** 用@Bean标注方法等价于XML中配置的bean。

```
@Bean
public Executor executor() {
    return Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors() * 2,
new ThreadFactory() {
        final AtomicInteger bizThreadIndex = new AtomicInteger(0);
        @Override
        public Thread newThread(Runnable r) {
            return new Thread(r, "Biz-Thread-" + bizThreadIndex.getAndIncrement());
        }
    });
}
```

```
@Bean(initMethod = "init", destroyMethod = "destroy")
@Scope(scopeName = "prototype")
public ExampleBean exampleBean() {
    ExampleBean exampleBean = new ExampleBean();
    exampleBean.setName("jack");
    return exampleBean;
}
```

- **Environment**

`org.springframework.core.env.Environment`环境类，Spring3.1以后开始引入。比如JDK环境，Servlet环境，Spring环境等等；每个环境都有自己的配置数据，如System.getProperties()、System.getenv()等可以拿到JDK环境数据；ServletContext.getInitParameter()可以拿到Servlet环境配置数据等等；也就是说Spring抽象了一个Environment来表示环境配置。

在Spring Boot中使用直接用@Resource或@Autowired注入，即可获得系统配置文件application.properties/yml的属性值，如果是自定义的配置文件，则需要预先通过@PropertySource等其他注解注入后，才能获取。获取通过getProperty()方法获取。

```
@Autowired
private Environment environment;

@RequestMapping(value = "/environment", method = {RequestMethod.GET})
public Map<String, Object> environment() {
    Map<String, Object> map = new HashMap<>();
    map.put("app.config.host", environment.getProperty("app.config.host"));
    map.put("java.home", environment.getProperty("java.home"));
    map.put("user.dir", environment.getProperty("user.dir"));
    return map;
}
```

2. 基本配置

2.1 入口类

良好的习惯，通常Spring Boot项目有一个名为 ***Application**的入口类，入口类里有 一个main方法，该main方法其实就是一个标准的Java应用的入口方法。在main方法中使用SpringApplication.run(..)方法来启动Spring Boot应用项目。

```java
package com.bittech.boot;

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

@SpringBootApplication
public class ExampleApplication {

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

@SpringBootApplication是一个组合注解。其中@EnableAutoConfiguration注解让Spring Boot根据类路径中的jar包依赖为当前项目进行自动配置。例如：添加spring-boot-starter-web依赖，就会自动添加Tomcat和Spring MVC的依赖，同时Spring Boot会对Tomcat和Spring MVC进行自动配置。

Spring Boot会自动扫描@SpringBootApplication所在类的同级包（例如：com.bittech.boot）以及下级子包里的Bean。

建议入口类放置在项目根包下，即项目最外层包。

2.2 关闭自动配置

有时候需要关闭特定的自动配置，这时需要使用@SpringBootApplication注解的exclude参数，例如：

```java
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
```

2.3 定制Banner

2.3.1 修改Banner

- 在Spring启动的时候会有一个默认的图案，由Spirng以及版本信息组成

```
.   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
\\/  ___)| |_)| | | | | || (_| |  ) ) ) )
'  |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
:: Spring Boot ::        (v2.0.2.RELEASE)
```

- 在`src/main/resources`下新建一个banner.txt
- 通过Text to ASCII Art Generator网站生成字符，比如：`Bit Tech for Java`将生成的字符复制到banner.txt文件
- 再次启动程序，欢迎图案变成如下所示

```
  |  _ \(_) |    |__    __|       | |       / _|                | |
  | |_) |_| |_      | | ___   ___| |__    | |_ ___   _ __       | | __ ___    ____ _
  |  _ <| | __|     | |/ _ \ / __| '_ \   |  _/ _ \| '__|  _    | |/ _` \ \ / / _` |
  | |_) | | |_      | |  __/ (__| | | |  | || (_) | |    | |__| | (_| |\ V / (_| |
  |____/|_|\__|     |_|\___|\___|_| |_|  |_| \___/|_|     \____/ \__,_| \_/ \__,_|
```

2.3.2 关闭Banner

- 在main方法中构造SpringApplication对象时修改

```
SpringApplication application = new SpringApplication(ExampleApplication.class);
application.setBannerMode(Banner.Mode.OFF);
application.run(args);
```

- 使用Fluent API修改

```
new SpringApplicationBuilder(ExampleApplication.class)
        .bannerMode(Banner.Mode.OFF)
        .run(args);
```

> 扩展：构建者模式（Builder Pattern），IDEA安装Builder Generator插件可以自动生成一个类的构建者模式类。

2.4 Spring Boot的配置

Spring Boot使用一个全局的配置文件`application.properties`或者`application.yml`，放置在`src/main/resources`目录或者类路径的`/config`下。

Spring Boot不仅支持常规的properties配置文件，还支持yml语言的配置文件。yml是以数据为中心的语言，在配置数据的时候具有面向对象的特征。

如下示例演示：将Tomcat的默认端口8080改为8090，并且将默认的访问路径`/`修改为`/springboot`。

- 在application.properties中修改

```
server.port=8090
server.servlet.context-path=/springboot
```

- 在application.yaml中修改

```
server:
      port: 8090
      servlet:
        context-path: /springboot
```

上面两种方式都能满足要求，但是@PropertySource注解是不支持加载yml文件，另外yaml文件的格式要求比较严格，加载时还需要解析，所以在日常开发中，比较习惯用properties文件来配置，所以推荐使用properties进行配置。

2.5 starter pom

Spring Boot提供了简化企业级开发绝大多数场景的starter pom,只要使用了应用场景需要的starter pom，相关的参数（属性）配置将会消除，就可以得到Spring Boot为我们提供的自动配置的Bean。

2.5.1 依赖版本列表

- 查看离线手册spring-boot-reference.pdf的Appendix F. Dependency versions章节
- 查看在线手册中依赖版本章节

2.5.2 官方starter pom

> 注意：下图中部分spring-boot-starter-*的名称可能会因为版本差异由所不同，需要以实际版本的帮助手册为准。

名　称	描　述
spring-boot-starter	Spring Boot 核心 starter，包含自动配置、日志、yaml 配置文件的支持
spring-boot-starter-actuator	准生产特性，用来监控和管理应用
spring-boot-starter-remote-shell	提供基于 ssh 协议的监控和管理
spring-boot-starter-amqp	使用 spring-rabbit 来支持 AMQP
spring-boot-starter-aop	使用 spring-aop 和 AspectJ 支持面向切面编程
spring-boot-starter-batch	对 Spring Batch 的支持
spring-boot-starter-cache	对 Spring Cache 抽象的支持
spring-boot-starter-cloud-connectors	对云平台（Cloud Foundry、Heroku）提供的服务提供简化的连接方式

名　称	描　述
spring-boot-starter-data-elasticsearch	通过 spring-data-elasticsearch 对 Elasticsearch 支持
spring-boot-starter-data-gemfire	通过 spring-data-gemfire 对分布式存储 GemFire 的支持
spring-boot-starter-data-jpa	对 JPA 的支持，包含 spring-data-jpa、spring-orm 和 Hibernate
spring-boot-starter-data-mongodb	通过 spring-data-mongodb，对 MongoDB 进行支持
spring-boot-starter-data-rest	通过 spring-data-rest-webmvc 将 Spring Data repository 暴露为 REST 形式的服务
spring-boot-starter-data-solr	通过 spring-data-solr 对 Apache Solr 数据检索平台的支持
spring-boot-starter-freemarker	对 FreeMarker 模板引擎的支持
spring-boot-starter-groovy-templates	对 Groovy 模板引擎的支持
spring-boot-starter-hateoas	通过 spring-hateoas 对基于 HATEOAS 的 REST 形式的网络服务的支持
spring-boot-starter-hornetq	通过 HornetQ 对 JMS 的支持
spring-boot-starter-integration	对系统集成框架 spring-integration 的支持
spring-boot-starter-jdbc	对 JDBC 数据库的支持
spring-boot-starter-jersey	对 Jersery REST 形式的网络服务的支持
spring-boot-starter-jta-atomikos	通过 Atomikos 对分布式事务的支持
spring-boot-starter-jta-bitronix	通过 Bitronix 对分布式事务的支持
spring-boot-starter-mail	对 javax.mail 的支持
spring-boot-starter-mobile	对 spring-mobile 的支持
spring-boot-starter-mustache	对 Mustache 模板引擎的支持
spring-boot-starter-redis	对键值对内存数据库 Redis 的支持，包含 spring-redis
spring-boot-starter-security	对 spring-security 的支持
spring-boot-starter-social-facebook	通过 spring-social-facebook 对 Facebook 的支持
spring-boot-starter-social-linkedin	通过 spring-social-linkedin 对 Linkedin 的支持
spring-boot-starter-social-twitter	通过 spring-social-twitter 对 Twitter 的支持
spring-boot-starter-test	对常用的测试框架 JUnit、Hamcrest 和 Mockito 的支持，包含 spring-test 模块
spring-boot-starter-thymeleaf	对 Thymeleaf 模板引擎的支持，包含于 Spring 整合的配置
spring-boot-starter-velocity	对 Velocity 模板引擎的支持
spring-boot-starter-web	对 Web 项目开发的支持，包含 Tomcat 和 spring-webmvc
spring-boot-starter-Tomcat	Spring Boot 默认的 Servlet 容器 Tomcat
spring-boot-starter-Jetty	使用 Jetty 作为 Servlet 容器替换 Tomcat
spring-boot-starter-undertow	使用 Undertow 作为 Servlet 容器替换 Tomcat
spring-boot-starter-logging	Spring Boot 默认的日志框架 Logback
spring-boot-starter-log4j	支持使用 Log4J 日志框架
spring-boot-starter-websocket	对 WebSocket 开发的支持

spring-boot-starter-ws	对 Spring Web Services 的支持

2.5.3 第三方starter pom

名　　称	地　　址
Handlebars	https://github.com/allegro/handlebars-spring-boot-starter
Vaadin	https://github.com/vaadin/spring/tree/master/vaadin-spring-boot-starter
Apache Camel	https://github.com/apache/camel/tree/master/components/camel-spring-boot
WRO4J	https://github.com/sbuettner/spring-boot-autoconfigure-wro4j
Spring Batch（高级用法）	https://github.com/codecentric/spring-boot-starter-batch-web
HDIV	https://github.com/hdiv/spring-boot-starter-hdiv
Jade Templates（Jade4J）	https://github.com/domix/spring-boot-starter-jade4j
Actitivi	https://github.com/Activiti/Activiti/tree/master/modules/activiti-spring-boot/spring-boot-starters

- 阿里巴巴Druid数据源 druid-spring-boot-starter
- Mybatis在Spring Boot中应用 mybatis-spring-boot-starter

2.6 使用XML配置

Spring Boot提倡零配置，即无XML配置。但是在实际项目中，可能有一些特殊要求必须使用XML配置，这时候可以通过Spring 提供的@ImportResource来加载XML配置。

```
@ImportResource({"classpath:context-bean.xml"})
```

3. 外部配置

Spring Boot允许使用properties文件，yaml文件或者命令行参数作为外部配置。

3.1 命令行参数配置

Spring Boot可以是基于jar包运行，打成jar包的程序可以直接通过下面命令运行。

```
java -jar xxx.jar
```

可以通过命令行参数修改Tomcat的端口号:

```
java -jar xxx.jar --server.port=8090
```

通过Maven打包之后，SpringBoot的配置文件会被打包的jar包中，要修改配置除了通过上面的命令行参数的方式进行修改之外，还可以通过在jar的包的同级目录或者同级目录下的conf目录中创建application.properties或者application.yml来进行参数的覆盖。

```
springboot-case-1.0.0.war
#外置的配置文件
application.properties
```

3.2 属性配置

3.2.1 常规属性配置

在常规的Spring环境下，注入properties文件里的值的方式，通过@PropertySource指明properties文件的位置，然后通过@Value注入值。

在Spring Boot里，只需要在applicat.properties定义属性，直接使用@Value注入即可。

示例：

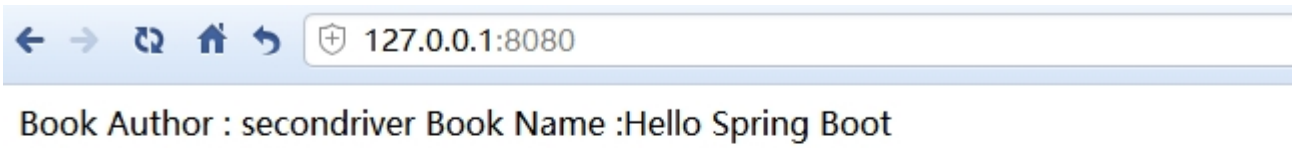
- 在application.properties增加属性：

```
book.author=secondriver
book.name=Hello Spring Boot
```

- 修改入口类

```
@SpringBootApplication
@RestController
public class ExampleApplication {
    @Value("${book.author}")
    private String bookAuthor;
    @Value("${book.name}")
    private String bookName;
    @RequestMapping("/")
    public String index() {
        return "Book Author : " + bookAuthor + " Book Name :" + bookName;
    }
    public static void main(String[] args) {
        SpringApplication.run(ExampleApplication.class, args);
    }
}
```

- 运行服务，访问http://127.0.0.1:8080/接口



3.2.2 类型安全的配置

上面示例中使用@Value注入每个配置在实际项目中会显的格外麻烦，因为我们的配置通常会是许多个，若使用上例的方式则要使用@Value注入很多次。

Spring Boot提供了基于类型安全的配置方式，通过@ConfigurationProperties将properties属性和一个Bean及其属性关联，从而实现类型安全的配置。

示例：

- 创建一个类型安全Bean

```
@ConfigurationProperties(prefix = "app.book")
@Component
public class BookProperties {
    private String author;
    private String name;
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

通过@ConfigurationProperties加载properties文件内的配置，通过prefix属性指定properties的配置的前缀，通过locations属性指定properties文件的位置，如果不指定默认读取application.properties和application.yaml。

- 在application.properties增加属性：

```
app.book.author=secondriver
app.book.name=Hello Spring Boot
```

> 注意：为了能够在增加属性的时候，IDE智能提示，可以在pom中添加如下依赖：

```
<!-- SpringBoot提供的一个类型安全配置的处理器，编译时工作，为IDE智能提示提供支持 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <scope>provided</scope>
</dependency>
```

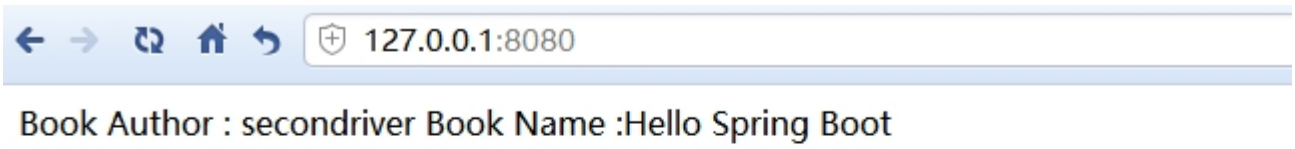
- 修改入口类

```
@SpringBootApplication
@RestController
public class ExampleApplication {
    @Autowired
    private BookProperties bookProperties;

    @RequestMapping("/")
```

```
    public String index() {
        return "Book Author : " + bookProperties.getAuthor() + " Book Name :" +
bookProperties.getName();
    }

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

通过@Autowired注解直接将BookProperties的Bean注入到类的成员。

- 运行服务，访问http://127.0.0.1:8080/接口



4. 日志配置

Spring Boot支持 `Java Util Logging, Log4j, Log4J2` 和 `Logback` 作为日志框架，无论使用哪种日志框架，Spring Boot已为当前使用日志框架的控制台输出以及输出文件做好了配置。

默认情况下，Spring Boot使用Logback作为日志框架。

- 配置日志输出文件

```
logging.file=D:\spring-boot.log
```

- 配置日志级别

配置日志级别的格式为：logging.level.包名=级别（`TRACE,DEBUG,INFO,WARN,ERROR,FATAL,OFF`）

```
logging.level.root=ERROR
logging.level.com.bittech.boot=DEBUG
```

5. Profile配置

Profile是Spring用来针对不同环境的进行不同配置提供的支持。全局Profile配置使用 `application-{profile}.properties`（例如：application-prod.properties）。

通过在application.properties中设置 `spring.profiles.active=prod` 来指定活动的Profile。这里多个活动的Profile时使用逗号分隔即可。

下面示例演示生产环境（prod）和开发环境(dev)，生产环境Tomcat端口号80，开发环境Tomcat端口号8080：

- 创建`application-prod.properties`和`application-dev.properties`

配置文件目录结构：

- 生产和开发环境的配置如下：

application-prod.properties

```
server.port=80
book.name=Hello Spring Boot (prod)
```

application-dev.properties

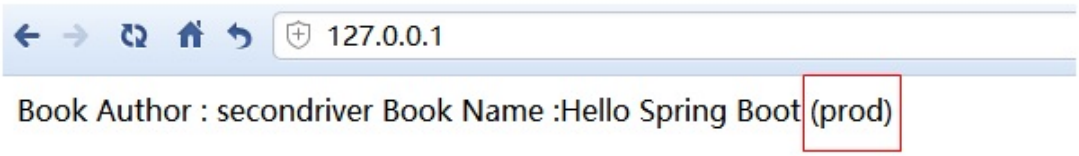
```
server.port=8080
book.name=Hello Spring Boot (dev)
```

application.properties文件配置以及修改profile

```
spring.profiles.active=prod
book.author=secondriver
book.name=Hello Spring Boot
```

- 切换**profile**运行程序，访问地址，效果如下：

生产环境 80端口默认省掉



开发环境

总结

知识块	知识点	分类	掌握程度
SpringBoot注解	1.核心注解类	实战型	掌握
Spring应用	1.基本配置 2.starter理解 3.配置文件 4. 日志配置	实战型	掌握

扩展

- JavaSE课程中学生开发的多人聊天应用改造为基于SpringBoot技术的多人聊天应用。