

## 2.4.1 Tomcat 配置

### 1. 常规配置

在 Spring Boot 项目中，可以内置 Tomcat、Jetty、Undertow、Netty 等容器。当开发者添加了 spring-boot-starter-web 依赖之后，默认会使用 Tomcat 作为 Web 容器。如果需要对 Tomcat 做进一步的配置，可以在 application.properties 中进行配置，代码如下：

```
1 server.port=8081
2 server.error.path=/error
3 server.servlet.session.timeout=30m
4 server.servlet.context-path=/chapter02
5 server.tomcat.uri-encoding=utf-8
6 server.tomcat.max-threads=500
7 server.tomcat.basedir=/home/sang/tmp
```

代码解释：

- server.port 配置了 Web 容器的端口号。
- error.path 配置了当项目出错时跳转去的页面。
- session.timeout 配置了 session 失效时间，30m 表示 30 分钟，如果不写单位，默认单位是秒。  
由于 Tomcat 中配置 session 过期时间以分钟为单位，因此这里单位如果是秒的话，该时间会被转换为一个不超过所配置秒数的最大分钟数，例如这里配置了 119，默认单位为秒，则实

际 session 过期时间为 1 分钟。

- context-path 表示项目名称，不配置时默认为/。如果配置了，就要在访问路径中加上配置的路径。
- uri-encoding 表示配置 Tomcat 请求编码。
- max-threads 表示 Tomcat 最大线程数。
- basedir 是一个存放 Tomcat 运行日志和临时文件的目录，若不配置，则默认使用系统的临时目录。

### Http 重新定向到https

这是因为 Spring Boot 不支持同时在配置中启动 HTTP 和 HTTPS。这个时候可以配置请求重定向，将 HTTP 请求重定向为 HTTPS 请求。配置方式如下：

```
1 @Configuration
2 public class TomcatConfig {
3     @Bean
4     TomcatServletWebServerFactory tomcatServletWebServerFactory() {
5         TomcatServletWebServerFactory factory = new TomcatServletWebServerFactory(){
6             @Override
```

```

7     protected void postProcessContext(Context context) {
8         SecurityConstraint constraint = new SecurityConstraint();
9         constraint.setUserConstraint("CONFIDENTIAL");
10        SecurityCollection collection = new SecurityCollection();
11        collection.addPattern("/*");
12        constraint.addCollection(collection);
13        context.addConstraint(constraint);
14    }
15}
16factory.addAdditionalTomcatConnectors(createTomcatConnector());
17return factory;
18}
19private Connector createTomcatConnector() {
20    Connector connector = new
21 Connector("org.apache.coyote.http11.Http11NioProtocol");
22    connector.setScheme("http");
23    connector.setPort(8080);
24    connector.setSecure(false);
25    connector.setRedirectPort(8081);
26    return connector;
27}
28}

```

Spring Boot 项目中的 application.properties 配置文件一共可以出现在如下 4 个位置：

- 项目根目录下的 config 文件夹中。
- 项目根目录下。
- classpath 下的 config 文件夹中。
- classpath 下。

如果这 4 个位置中都有 application.properties 文件，那么加载的优先级从 1 到 4 依次降低，如图 2-10 所示。Spring Boot 将按照这个优先级查找配置信息，并加载到 Spring Environment 中。

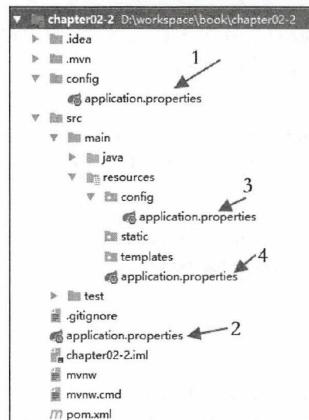


图 2-10

如果开发者在开发中未使用 application.properties，而是使用了 application.yml 作为配置文件，那么配置文件的优先级与图 2-10 一致。

默认情况下，Spring Boot 按照图 2-10 的顺序依次查找 application.properties 并加载。如果开发者不想使用 application.properties 作为配置文件名，也可以自己定义。例如，在 resources 目录下创建一个配置文件 app.properties，然后将项目打成 jar 包，打包成功后，使用如下命令运行：

```
1 | java -jar chapter02-2-0.0.1-SNAPSHOT.jar --spring.config.name=app
```

在运行时再指定配置文件的名字。使用 spring.config.location 可以指定配置文件所在目录（注意需要以/结束），代码如下：

```
1 | java -jar chapter02-2-0.0.1-SNAPSHOT.jar --spring.config.name=app
--spring.config.location=classpath:/
```

YAML 是 JSON 的超集，简洁而强大，是一种专门用来书写配置文件的语言，可以替代 application.properties。在创建一个 Spring Boot 项目时，引入的 spring-boot-starter-web 依赖间接地引入了 snakeyaml 依赖，snakeyaml 会实现对 YAML 配置的解析。YAML 的使用非常简单，利用缩进来表示层级关系，并且大小写敏感。在 Spring Boot 项目中使用 YAML 只需要在 resources 目录下创建一个 application.yml 文件即可，然后向 application.yml 中添加如下配置：

```
1 server:  
2   port: 80  
3   servlet:  
4     context-path: /chapter02  
5   tomcat:  
6     uri-encoding: utf-8
```

这一段配置等效于 application.properties 中的如下配置：

```
1 server.port=80  
2 server.servlet.context-path=/chapter02  
3 server.tomcat.uri-encoding=utf-8
```

此时可以将 resources 目录下的 application.properties 文件删除，完全使用 YAML 完成文件的配置。

*在 Spring Boot 中使用 YAML 虽然方便 但是 YAML 也有一些缺陷 例如无法使用 @PropertySource 注解加载 YAML 文件*

## 2.8 Profile

开发者在项目发布之前，一般需要频繁地在开发环境、测试环境以及生产环境之间进行切换，这个时候大量的配置需要频繁更改，例如数据库配置、redis 配置、mongodb 配置、jms 配置等。频繁修改带来了巨大的工作量，Spring 对此提供了解决方案（@Profile 注解），Spring Boot 则更进一步提供了更加简洁的解决方案，Spring Boot 中约定的不同环境下配置文件名称规则为 application-{profile}.properties，profile 占位符表示当前环境的名称，具体配置步骤如下：

### 1. 创建配置文件

首先在 resources 目录下创建两个配置文件：application-dev.properties 和 application-prod.properties，分别表示开发环境中的配置和生产环境中的配置。其中，application-dev.properties 文件的内容如下：

```
1 server.port=8080
```

application-prod.properties 文件的内容如下：

```
1 server.port=80
```

这里为了简化问题并且容易看到效果，两个配置文件中主要修改了一下项目端口号。

## 2. 配置 application.properties

然后在 application.properties 中进行配置：

```
1 spring.profiles.active=dev
```

这个表示使用 application-dev.properties 配置文件启动项目，若将 dev 改为 prod，则表示使用 application-prod.properties 启动项目。项目启动成功后，就可以通过相应的端口进行访问了。

## 3. 在代码中配置

对于第二步在 application.properties 中添加的配置，我们也可以在代码中添加配置来完成，在启动类的 main 方法上添加如下代码，可以替换第二步的配置：

```
1 SpringApplicationBuilder builder = new  
2 SpringApplicationBuilder(Chapter013Application.class);  
3 builder.application().setAdditionalProfiles("prod");  
4 builder.run(args);
```

## 4. 项目启动时配置

对于第 2 步和第 3 步提到的两种配置方式，也可以在将项目打成 jar 包后启动时，在命令行态指定当前环境，示例命令如下：

```
1 java -jar chapter01-3-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod
```

顾名思义，@ControllerAdvice 就是@Controller 的增强版。@ControllerAdvice 主要用来处理全局数据，一般搭配@ExceptionHandler、@ModelAttribute 以及@InitBinder 使用。

@ControllerAdvice 是一个全局数据处理组件，因此也可以在@ControllerAdvice 中配置全局数据，使用@ModelAttribute 注解进行配置，代码如下：

```
1 @ControllerAdvice  
2 public class GlobalConfig {  
3     @ModelAttribute(value = "info")  
4     public Map<String, String> userInfo() {  
5         HashMap<String, String> map = new HashMap<>();  
6         map.put("username", "罗贯中");  
7         map.put("gender", "男");  
8         return map;  
9     }  
10 }
```

代码解释：

- 在全局配置中添加 userInfo 方法，返回一个 map。该方法有一个注解@ModelAttribute，其中的 value 属性表示这条返回数据的 key，而方法的返回值是返回数据的 value。
- 此时在任意请求的 Controller 中，通过方法参数中的 Model 都可以获取 info 的数据。

```
16  @Controller
17  public class IndexController {
18
19      @Value("${school.name}")
20      private String schoolName;
21
22      @Value("http://www.bjpowernode.com")
23      private String website;
24
25      @RequestMapping(value = "/say")
26      public @ResponseBody String say() { return "Hello:" + schoolName + ":" +
27  }
```

现在我们要使用springboot集成jsp, 手动指定jsp最后编译的路径  
而且springboot集成jsp编译jsp的路径是springboot规定好的位置  
META-INF/resources

```
-->
<resources>
    <resource>
        <!--源文夹-->
        <directory>src/main/webapp</directory>
        <!--指定编译到META-INF/resources-->
        <targetPath>META-INF/resources</targetPath>
        <!--指定源文件夹中的哪个资源要编译进行-->
        <includes>
            <include>.*.</include>
        </includes>
    </resource>
</resources>
```

```
StudentMapper.java × StudentMapper.java × StudentMapper.xml ×
<resultMap id="BaseResultMap" type="com.bjpowernode.springboot.model.Student">
    <!--id 标签只能修改主键字段-->
    <!--result 除了主键以外的字段-->
    <!--
        column 数据库中的字段名称
        property 映射对象的属性名称
        jdbcType 列中数据库中字段的类型(可以省略不写)
    -->
    <id column="id" jdbcType="INTEGER" property="id"/>
    <result column="name" jdbcType="VARCHAR" property="name"/>
    <result column="age" jdbcType="INTEGER" property="age"/>
</resultMap>
```

```
<!--
    column 数据库中的字段名称
    property 映射对象的属性名称
    jdbcType 列中数据库中字段的类型(可以省略不写)
-->
<!--
    resultMap作用：
    1.当数据库中字段名称与实体类对象的属性名不一致时，可以进行转换
    2.当前查询的结果没有对应一个表的时候，可以自定义一个结果集
-->

<!--
    数据库表字段名称      实体对象属性名称
    user_name            userName
    product_type         productType
-->
<!--
    如果数据库中字段名称由多个单词构成，通过MyBatis逆向工程生成的对象属性名称
    会按照驼峰命名法规则生成属性名称
    其中：数据库中字段名称由多个单词构成的时候必须使用_下划线分隔
-->
<id column="id" jdbcType="INTEGER" property="id"/>
<result column="name" jdbcType="VARCHAR" property="name"/>
```

数据库添加依赖: (1) mysql (2) mybatis集成Springboot框架起步依赖

```
<!--MySQL驱动-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<!--MyBatis集成SpringBoot框架起步依赖-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.0.0</version>
</dependency>
```

```
import ...

@SpringBootApplication //开启spring配置
@MapperScan(basePackages = "com.bipowernode.springboot.mapper") //开启扫描
public class Application {

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

```
4 3. 使用springboot框架的核心配置文件application.properties
5 4. 使用springboot框架的核心配置文件application.yml或者application.yaml
6 5. springboot框架的核心配置application.properties和application.yaml或者application.yml
7 6. 多环境下核心配置文件的使用(properties),工作中开发的环境有哪此:开发环境,测试环境,;
8 7. 多环境下核心配置文件的使用(yaml或yml),工作中开发的环境有哪此:开发环境,测试环境,准
9 8. springboot在核心配置文件application.properties自定义配置一个一个获取@Value
10 9. springboot在核心配置文件将自定义配置映射到一个对象
11 10. SpringBoot集成jsp
12 11. SpringBoot集成MyBatis
13     a. 添加mybatis依赖,MySQL驱动
14     b. 使用MyBatis提供的逆向工程生成实体bean,映射文件,DAO接口
15 12-15: SpringBoot集成MyBatis,最主要的是两个注解@Mapper,@MapperScan
16 @Mapper 需要在每一个Mapper接口类上添加,作用扫描dao接口
17 @MapperScan 是在SpringBoot启动入口类上添加的,它是扫描所有的包
18
19 9. 关于Mapper映射文件存放的位置的写法有以下两种:
20 1. 将Mapper接口和Mapper映射文件存放到src/main/java同一目录下,
21     还需要在pom文件中手动指定资源文件夹路径resources
22
23 2. 将Mapper接口和Mapper映射文件分开存放
24     Mapper接口类存放到src/main/java目录下
25     Mapper映射文件存放到resources(类路径)
26     在springboot核心配置文件中指定mapper映射文件存放位置
```