SpringBoot

引言

> 互联网目前仍处于高速发展中，各行各业的信息化发展也是如火如荼。不管是信息化的第一代产品，还是软件的迭代更新，都离不开框架的支持。Servlet、SSH（Spring + SpringMVC + Hibernate | Spring + Struts + Hibernate）、SSM （Spring + SpringMVC + MyBatis）作为早期流行的框架，使得软件开发的进度进一步提速。不断的发展过程中，开发人员发现，有新的需求时，重构项目搭建环境是非常繁琐的。Spring 公司也意识到这个问题，并且研发并制定了一整套的开发流程，开发人员只需要遵守流程，就可以快速上手构建企业级的框架，快速开发，简化重复的环境配置工作的同时也可以顺应大数据时代发展。

1.1 SpringBoot 入门

1.1.1 课程目标

- 了解 SpringBoot 概念和优点
- 熟练创建 SpringBoot 工程
- 掌握 SpringBoot 自动配置理论

1.1.2 知识架构树

- SpringBoot 简介
- 创建 SpringBoot 工程
- SpringBoot 自动配置

1.1.3 理论知识

1.1.3.1 SpringBoot 简介

SpringBoot 简介

> Spring 应用原始的开发流程复杂，配置繁多，开发效率低，第三方应用继承难度大。Spring 公司为了解决这个问题，研发了SpringBoot。SpringBoot 来简化 Spring 应用的开发，**约定大于配置**，去繁从简，很容易就可以生成一个企业级别的应用。

SpringBoot优点

- 快速创建独立运行的 Spring 项目以及主流框架集成
- 使用嵌入式的 Servlet 容器，应用无需生成 war 包
- Starters（启动器）自动依赖与版本控制
- 大量的自动配置，简化开发，也可以修改默认配置
- 无需配置 XML ，无代码生成，开箱即用
- 准生产环境的运行实时应用监控

- 与云计算的天然集成

开发环境准备

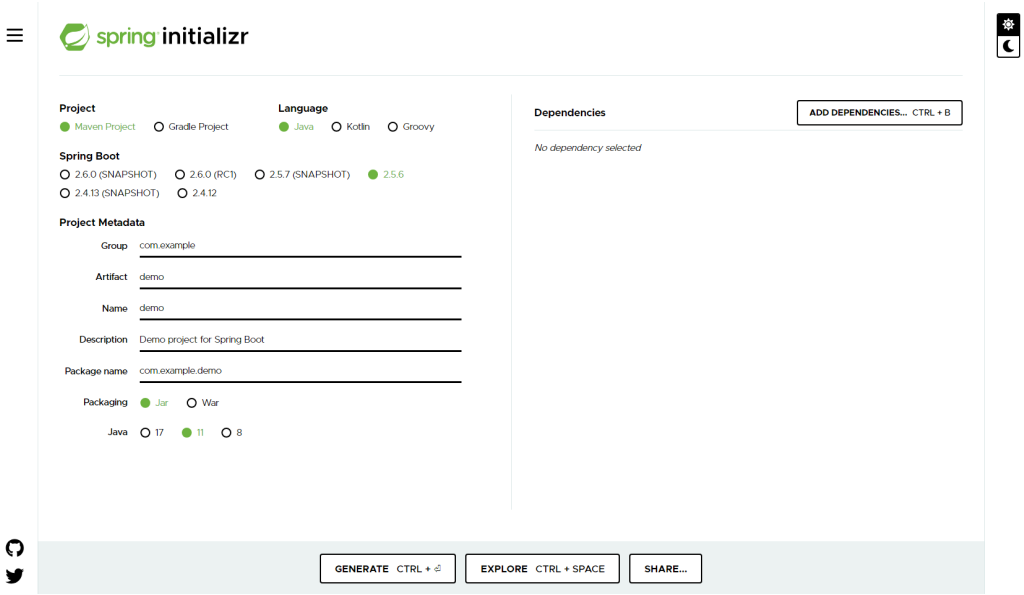
- JDK： 1.8 及以上环境，官方推荐 1.7 以上；
- Maven ： 方便项目管理和 Jar 包管理；
- 基础技术：Spring、SpringMVC；
- 拓展技术：MyBatis、日志、事务、权限等；
- IDE 开发工具：IDEA，STS（Eclipse）；

1.1.3.2 创建 SpringBoot 工程

> SpringBoot 项目启动完成不会自动打开浏览器。
>
> SpringBoot 请求的地址默认是不带项目名称的

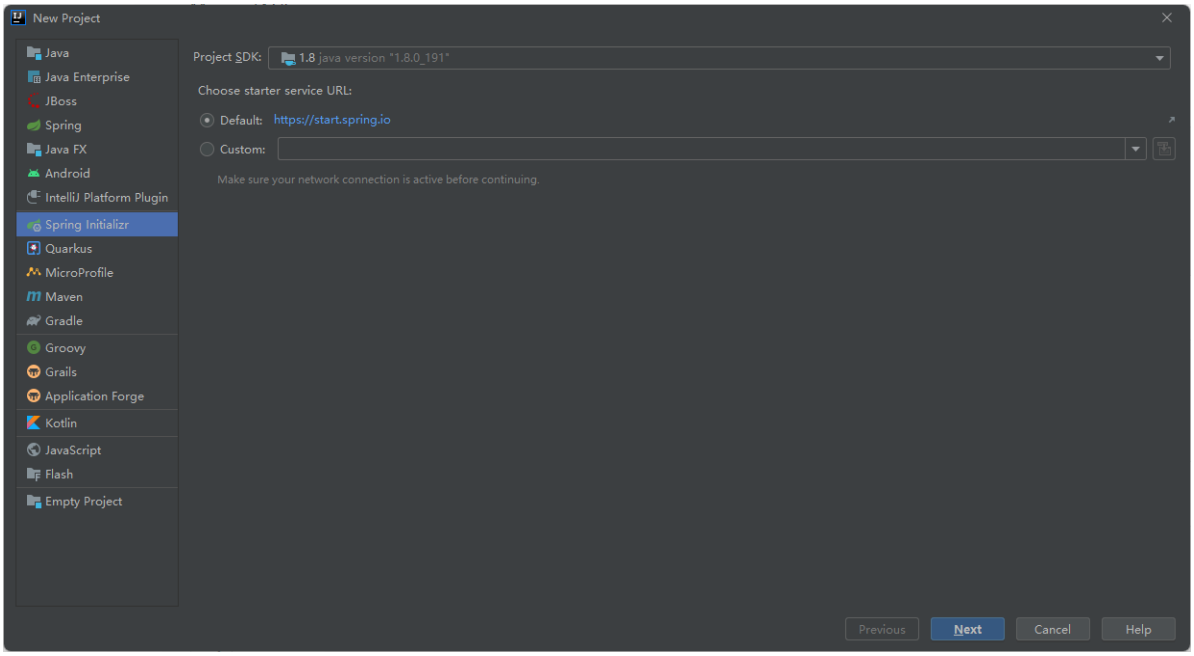
1.1.3.2.1 官网构建项目

> 初始化地址（https://start.spring.io/）



1.1.3.2.2 IDE 构建

> 本质上借助还是官方的初始化工具



1.1.3.2.3 Maven 自己搭建

需求：浏览器发送请求，服务器接收请求，响应给浏览器Hello

1）导入SpringBoot的父项目

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.1</version>
</parent>
```

2）导入需要的场景启动器，比如：要开发web项目

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

3）编写启动类：**建议启动类放到基础包下**

```
package com.soft;

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

/**
 * 通过Main函数去启动SpringBoot应用
 * @SpringBootApplication
 *      标识这是一个SpringBoot的应用
 **/
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        // 要启动Spring应用
        // 参数1：  @SpringBootApplication所在类的类对象
        // 参数2：  main函数的参数
        SpringApplication.run(Application.class, args);
    }
}
```

4）编写核心业务代码

```
@Controller
@RequestMapping("/hello")
public class HelloController {

    @RequestMapping(value = "/sayHi")
    public @ResponseBody String hello(){
        return "hello !";
    }
}
```

5）运行启动类，可以看到 Spring 的标识。

1.1.3.2.4 目录结构

> 目录结构是一个标准的 Maven 目录结构，在此基础上，Spring Boot 又规定了几个目录。

```
工程名称
    src
        main
            java
                java代码
            resources
                static：静态资源目录，js、css、img等；
                public：静态资源目录，js、css、img等；
                resources：静态资源目录，js、css、img等；

                templates：模板页面（html）目录，模板引擎（Thymeleaf），可以理解为之前的 webapp/WEB-INF/ 目录

                application.properties：SpringBoot 配置文件，默认是空的可以修改默认配置
                application.yml：SpringBoot 配置文件，默认是空的可以修改默认配置
        test
```

1.1.3.2.5 项目打包

> 项目打包就是将开发完的项目进行整合导出，形成一个包。这个包可以直接通过服务器运行。
>
> 包分为两个类型，war包和jar 包；war包是Tomcat服务器能识别的包。jar包就是jdk能直接运行的包。
>
> 简化部署，把项目生成jar包

```
<!-- 导入maven打包插件，通过Maven的package命令打包工程 -->
<build>
    <plugins>
        <!-- 将项目打包成一个可执行的jar包 -->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

通过`java -jar jar包名`命令启动SpringBoot应用

1.1.3.3 SpringBoot 自动配置

1.1.3.3.1 pom文件

（1）SpringBoot 父项目

> 做了大量的依赖版本控制，自动导入对应版本依赖，避免出现因版本不对应而发生问题的情况发生。

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

```
    <version>xxx</version>
</parent>

父项目的父项目：用于管理版本号
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-dependencies</artifactId>
    <version>xxx</version>
    <relativePath>../../spring-boot-dependencies</relativePath>
</parent>
spring-boot-dependencies依赖中包含了所有相关依赖jar包的版本；
可以理解为SpringBoot的版本控制中心，以后导入依赖默认是不需要写版本的。
```

（2）导入场景启动器的依赖，自动注入到Spring容器中。

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

1.1.3.3.2 启动类

```
package com.soft;

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

/**
 * 通过Main函数去启动SpringBoot应用
 * @SpringBootApplication
 *      标识这是一个SpringBoot的应用
 **/
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        // 要启动Spring应用
        // 参数1：  @SpringBootApplication所在类的类对象
        // 参数2：  main函数的参数
        SpringApplication.run(Application.class, args);
    }
}
```

@SpringBootApplication：标注在类上，说明这个类是SpringBoot的主配置类，SpringBoot要运行这个类的main方法来启动Spring应用

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

- **@SpringBootConfiguration**：标注在类上，表示这个类是SpringBoot的配置类，把当前类注入到 Spring 容器中；
  - @Configuration：标注在类上，表示这个类是一个配置类；用于替换配置文件；
    - @Component：由于配置类也是容器中的一个组件，所以需要添加该注解进行管理
- **@EnableAutoConfiguration**：开启自动配置功能；以前需要手动配置的东西，现在都不需要配置，由SpringBoot自动配置注入到容器中可以直接使用；
  - **@AutoConfigurationPackage**：自动配置包；

> `@Import(AutoConfigurationPackages.Registrar.class)`：Spring的底层注解，用于给容器导入组件。由`Registrar.class`控制。
>
> 将主配置类（@SpringBootApplication 注解标注的类）所在包及子包下的所有组件扫描到 Spring 容器中。

```java
    /**
     * {@link ImportBeanDefinitionRegistrar} to store the base package from the importing
     * configuration.
     */
    static class Registrar implements ImportBeanDefinitionRegistrar, DeterminableImports {

        @Override
        public void registerBeanDefinitions(AnnotationMetadata metadata, BeanDefinitionRegistry registry) {
            register(registry, new PackageImports(metadata).getPackageNames().toArray(new String[0]));
        }

        @Override
        public Set<Object> determineImports(AnnotationMetadata metadata) {
            return Collections.singleton(new PackageImports(metadata));
        }

    }
```

- **@Import(AutoConfigurationImportSelector.class)**：给容器中导入已经扫描到的组件；

> `AutoConfigurationImportSelector`：自动配置导入的扫描器，将所需要的组件以全类名数组的方式返回，组件就会添加到容器中。会给容器中导入很多配置类（xxxAutoConfiguration），就是给容器中导入场景需要的所有组件并配置好这些组件。
>
> ```
> 1 SpringFactoriesLoader.loadFactoryNames(getSpringFactoriesLoaderFactoryClass(), getBeanClassLoader())
> ```

老版本：

> SpringBoot在启动时从类路径下的`META-INF/spring.factories`中获取`EnableAutoConfiguration`指定的值（类全限定名），将这些值对应的类导入到spring容器中，就可以帮助完成自动配置工作。（spring-boot-autoconfigure-2.3.4.RELEASE.jar）

新版本：

> SpringBoot在启动时从类路径下的`META-INF/spring/xxxx.imports`中读取每一行配置的自动注入类的全限定名称。

1.1.4 FAQ 手册

1.1.5 作业实践

1. 简述 SpringBoot 优点。
2. 使用开发工具构建 SpringBoot 项目。
3. 简述 SpringBoot 自动配置的原理。

1.1.6 面试宝典

1.1.7 拓展资料

1.2 SpringBoot 配置文件解析

1.2.1 课程目标

- 掌握 SpringBoot 配置文件
- 掌握 SpringBoot 多环境配置

1.2.2 知识架构树

1.2.3 理论知识

1.2.3.1 全局配置文件

> Spring Boot 启动会扫描 `file:./config/`、`file:./`、`classpath:/config/`、`classpath:/` 位置的 `application.properties` 或者`application.yml`文件作为 Spring Boot 的默认配置文件。按照优先级从高到低的顺序，所有位置的文件都会被加载，高优先级配置内容会覆盖低优先级配置内容。可以通过配置`spring.config.location`改变默认配置 。推荐使用`application.yml`作为项目的配置文件。

> YAML 是 "YAML Ain't a Markup Language"（YAML 不是一种标记语言）的递归缩写。在开发的这种语言时，YAML 的意思其实是："Yet Another Markup Language"（仍是一种标记语言）。YAML的语法和其他高级语言类似，并且可以简单表达**清单、散列表，标量**等数据形态。它使用**空白符号**缩进和大量依赖外观的特色，特别适合用来表达或编辑数据结构、各种配置文件、倾印调试内容、文件大纲（例如：许多电子邮件标题格式和YAML非常接近）。YAML 的配置文件后缀为 .yml，如：runoob.yml 。

1.2.3.2 YML 配置文件构建

1） 语法结构

```yaml
# key和value中间冒号（英文状态）后面必须添加"半角空格"
key: value

# yaml 文件中以空格的形式（个数不限）表示层级关系，只要是左对齐的空格都属于同一个层级。
key1:
    key1-1: value
    key1-2: value
    ......
key2:
    key2-1: value
    key2-2: value
    ......
```

```properties
# 对比properties写法
key=value

key1.key1-1=value
key1.key1-2=value

key2.key2-1=value
key2.key2-2=value
```

2） YML 描述数据

> **属性和值的大小写敏感**

（1）字面量：数字、字符串、布尔

```yaml
age: 28

# 字符串不用添加引号
# "" -> 会把字符中的特殊字符按照自身的含义表示
# '' -> 会把字符中的特殊字符按照字符串表示
name: tom
name: "tom \n jack"
    tom
    jack
name: 'tom \n jack'
    tom \n jack

boolean: true
```

（2）对象/Map

```yaml
# 普通写法
student:
    name: tom
    age: 18

# 行内写法
student: {name: tom, age: 18}
```

(3) 数组/List/Set

```yaml
# 普通写法
list:
    - v1
    - v2
    - v3

# 行内写法
list: [v1, v2, v3]
```

1.2.3.3 获取配置文件中的值

> JDBC 配置文件、错误信息配置文件.....
>
> 这类文件一般定义好之后很难去改动，所以要作为配置文件存在，读取配置文件也就是合情合理了！

1.2.3.3.1 获取默认配置文件的值

配置文件：

application.yml

```yaml
# 描述学生对象
student:
  # 字面量
  name: tom@123.com
  age: 18
  sex: 男
  tel: 12345678901
  # 集合
  hobby:
    - game
    - video
    - music
  # Map
  grades:
    chinese: 100
    english: 150
  # 对象
  major:
    mno: M-0001001
    mname: 软件工程
```

application.properties

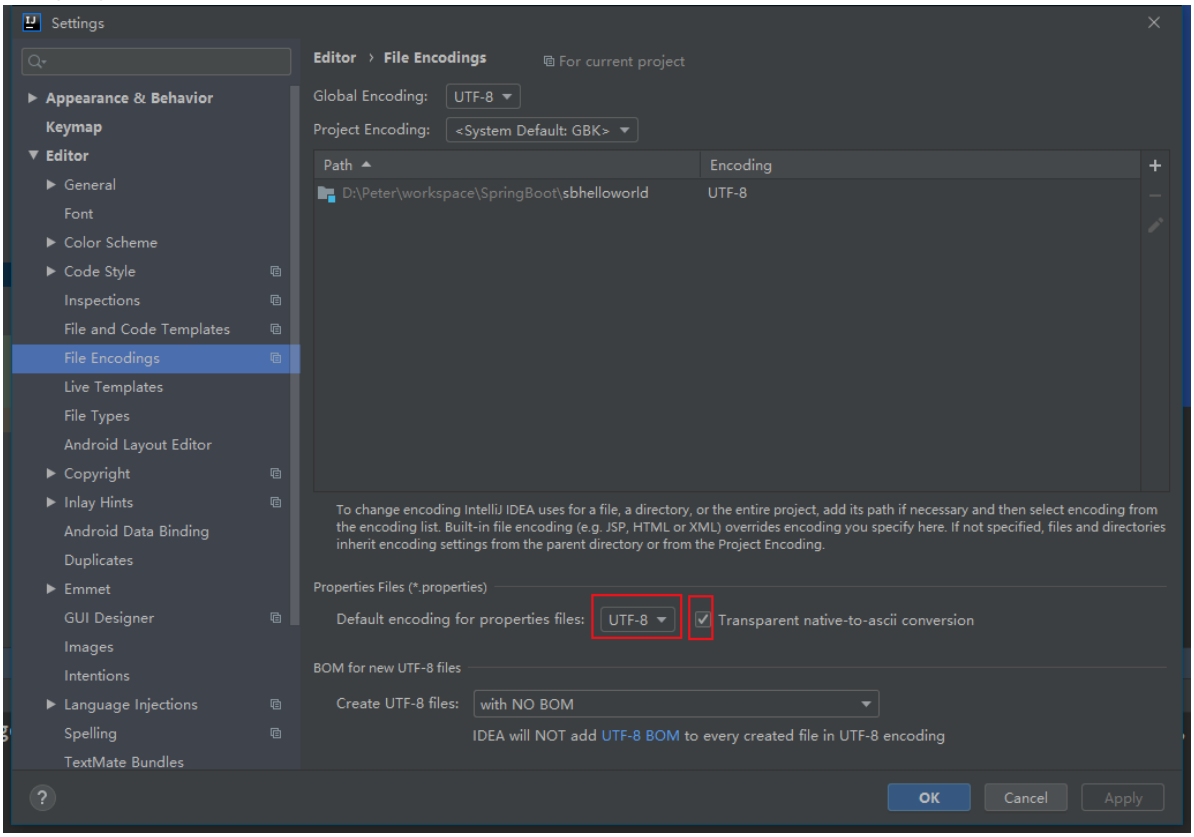
```properties
# 描述学生对象 乱码问题修改properties编码，并且把中文转码
student.name: tom@123.com
student.age: 18
student.sex: 男
student.tel: 12345678901

student.hobby:game,video,music

student.grades.chinese: 100
student.grades.english: 150

student.major.mno: M-0001001
student.major.mname: 软件工程
```

解决properties中中文乱码问题：



1） 使用 @ConfigurationProperties

Student.java

```java
/**
 * @ConfigurationProperties(prefix = "student")
 *       本类中所有的属性和配置文件中的属性绑定，prefix是配置文件中标识该对象的前缀
 *
 * @Component
 *       只有在spring容器中的组件才能使用@ConfigurationProperties功能，所以要添加该注解
 *
 * @Validated
 *       数据校验
 */
@Component
@ConfigurationProperties(prefix = "student")
```

```java
@Validated
public class Student {
    @Email // 验证是否符合邮箱规则
    private String name;
    private String age;
    private String sex;
    @Pattern(regexp="[0-9A-z!-/]")
    private String tel;

    private List<String> hobby;

    private Map<String, String> grades;

    private Major major;

    // 省略set/get
}
```

校验导包：

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.5.Final</version>
</dependency>
```

校验常用注解：

验证注解	验证的数据类型	说明
@AssertFalse	Boolean,boolean	验证注解的元素值是false
@AssertTrue	Boolean,boolean	验证注解的元素值是true
@NotNull	任意类型	验证注解的元素值不是null
@Null	任意类型	验证注解的元素值是null
@Min(value=值)	BigDecimal，BigInteger, byte,short, int, long，等任何Number或CharSequence（存储的是数字）子类型	验证注解的元素值大于等于@Min指定的value值
@Max（value=值)	和@Min要求一样	验证注解的元素值小于等于@Max指定的value值
@DecimalMin(value=值)	和@Min要求一样	验证注解的元素值大于等于@ DecimalMin指定的value值
@DecimalMax(value=值)	和@Min要求一样	验证注解的元素值小于等于@ DecimalMax指定的value值

验证注解	验证的数据类型	说明
@Digits(integer=整数位数, fraction=小数位数)	和@Min要求一样	验证注解的元素值的整数位数和小数位数上限
@Si***=下限, max=上限)	字符串、Collection、Map、数组等	验证注解的元素值的在min和max（包含）指定区间之内，如字符长度、集合大小
@Past	java.util.Date,java.util.Calendar;Joda Time类库的日期类型	验证注解的元素值（日期类型）比当前时间早
@Future	与@Past要求一样	验证注解的元素值（日期类型）比当前时间晚
@NotBlank	CharSequence子类型	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首位空格
@Length(min=下限, max=上限)	CharSequence子类型	验证注解的元素值长度在min和max区间内
@NotEmpty	CharSequence子类型、Collection、Map、数组	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）
@Range(min=最小值, max=最大值)	BigDecimal,BigInteger,CharSequence, byte, short, int, long等原子类型和包装类型	验证注解的元素值在最小值和最大值之间
@Email(regexp=正则表达式,flag=标志的模式)	CharSequence子类型（如String）	验证注解的元素值是Email，也可以通过regexp和flag指定自定义的email格式
@Pattern(regexp=正则表达式,flag=标志的模式)	String，任何CharSequence的子类型	验证注解的元素值与指定的正则表达式匹配
@Valid	任何非原子类型	指定递归验证关联的对象；如用户对象中有个地址对象属性，如果想在验证用户对象时一起验证地址对象的话，在地址对象上加@Valid注解即可级联验证

自定义类在yml中的提示

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

2） 使用 @Value

```
@Component
public class Student {
    @Value("tom")
    private String name;
    @Value("18")
    private String age;
    @Value("1")
    private String sex;
    @Value("tom")
    private String tel;

    private List<String> hobby;

    private Map<String, String> grades;

    private Major major;
    // 省略set/get
}
```

@ConfigurationProperties和@Value区别：

	@ConfigurationProperties	@Value
功能	批量注入	需要在每一个属性上配置
松散绑定	支持	不支持
SpEL（SpringEL）	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

> 松散绑定可以将驼峰命名、下划线隔开、横行隔开三种方式进行等值识别。

松散绑定值得是给属性赋值时的写法：

写法一：标准驼峰命名 -> person.firstName 写法二：大写使用下划线表示 -> person.first_name 写法三：大写使用横杠表示 -> person.first-name

1.2.3.3.2 获取自定义配置文件的值

1） 使用 @PropertySource

student.properties

```properties
# 描述学生对象 乱码问题修改properties编码，并且把中文转码
student.name: tom@123.com
student.age: 18
student.sex: 男
student.tel: 12345678901

student.hobby:game,video,music

student.grades.chinese: 100
student.grades.english: 150

student.major.mno: M-0001001
student.major.mname: 软件工程
```

Student.java

```java
// 加载资源路径下自定义文件，注意只支持 properties 文件
@PropertySource(value = {"classpath:student.properties"})
@Component
@ConfigurationProperties(prefix = "student")
public class Student {
    private String name;
    private String age;
    private String sex;
    private String tel;

    private List<String> hobby;

    private Map<String, String> grades;

    private Major major;

    // 省略set/get
}
```

2）使用 @ImportResource

Student.java

```java
public class Student {
    private String name;
    private String age;
    private String sex;
    private String tel;

    private List<String> hobby;

    private Map<String, String> grades;

    private Major major;

    // 省略set/get
}
```

Spring 配置文件：配置给 Student 属性赋值

```xml
<?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
        https://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean class="com.soft.entity.Student">
        <!-- 字面量 -->
        <property name="name" value="Jack"/>
        <property name="age" value="22"/>
        <property name="sex" value="男"/>
        <property name="tel" value="11111111111"/>
        <!-- List集合 -->
        <property name="hobby">
            <list>
                <value>game</value>
                <value>video</value>
                <value>music</value>
            </list>
        </property>
        <!-- Map集合 -->
        <property name="grades">
            <map>
                <entry key="chinese" value="100"/>
                <entry key="english" value="150"/>
            </map>
        </property>
        <!-- 对象 -->
        <property name="major" ref="major"/>
    </bean>

    <bean id="major" class="com.soft.entity.Major">
        <property name="mno" value="M0001003"/>
        <property name="mname" value="软件工程"/>
    </bean>
</beans>
```

配置类

> SpringBoot 要求约定大于配置，不推荐使用 xml 形式的配置文件，也不会加载 `resources` 目录下的xml文件。
>
> 特殊情况下也非常需要使用 xml 配置文件，SpringBoot 提供了配置类的概念，在配置类中加载 xml 文件。配置类是 `@Configuration` 注解标识的类。
>
> 在配置类上使用注解 `@ImportResource(locations={classpath:spring-bean.xml})` 加载资源路径下自定义的 xml 配置文件

```java
@Configuration // 加载类上，标识当前类为配置类
@ImportResource(locations = {"classpath*:spring-bean.xml"}) // 当前注解一定要添加到配置类上，获取资源路径下的 xml 配置文件
public class StudentConfig {
}
```

3.3 使用 @Configuration

> 使用以上注解引入配置文件的形式确实不符合 SpringBoot 设计理念，所以 Spring 额外提供了一种向容器中注入对象的方式。同样需要依据配置类。

编写一个配置类（把原来的xml使用Java代码描述），使用@Bean给容器中添加组件

```
<bean id="student" class="com.soft.entity.Student"></bean>
```

原来 xml 的写法，转换为配置类的写法

```
@Configuration // 加载类上，标识当前类为配置类
public class StudentConfig{
    /*
     * 目标是使用 Java 代码形式替换掉 xml 文件。
     *      <bean id="student" class="com.soft.entity.Student"></bean>
     *
     * 方法的返回值标识容器中注入的对象类型， 也就是 xml 中的 class
     * 方法名称表示容器中当前对象的唯一标识，也就是 xml 中的 id
     *
     * @Bean 接收返回值，并注入到容器中
     */
    @Bean
    public Student student(){
        // 按需可以给对象赋值并返回
        Student student = new Student();

        student.setName("Lily");
        student.setAge("23");
        student.setTel("66666666666");

        return student;
    }
}
```

1.2.3.4 配置文件占位符

1）随机值

```
${random.int}
${random.value}
${random.uuid}
```

2）表达式获取配置的值

```
student:
  # 字面量
  first-name1: ${random.int}
  first_name2: ${random.value}
  firstName3: ${random.uuid}
  # 应用对象，当对象不存在时赋予默认值
  name: ${student.first-name1:defaultValue} - tom@123.com
  age: 18
  sex: 男
```

```yaml
  tel: 12345678901
  # 集合
  hobby:
    - game
    - video
    - music
  # Map
  grades:
    chinese: 100
    english: 150
  # 对象
  major:
    mno: M-0001001
    mname: 软件工程
```

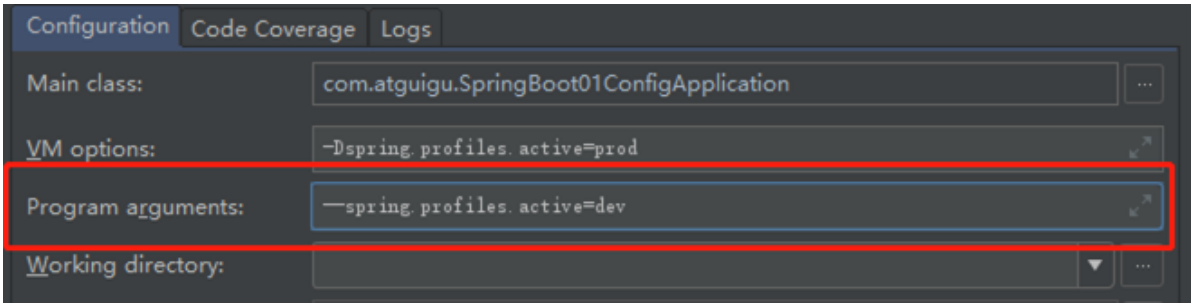
1.2.3.5 profile 多环境

> profile 是 Spring 对不同环境提供不同配置功能的支持，可以通过激活、指定参数等方式快速切换环境。
>
> 环境分为：开发环境、测试环境、生产环境

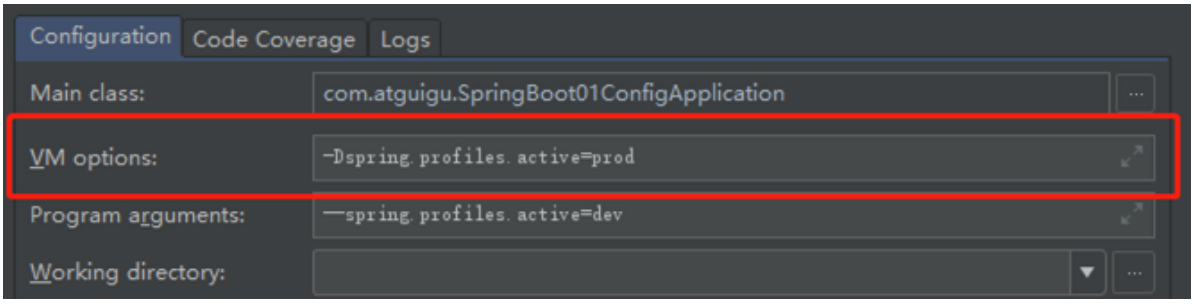
- **多 profile 文件模式**

```
文件命名格式：application-{profile}.yml
    application-dev.xml
    application-prod.xml
```

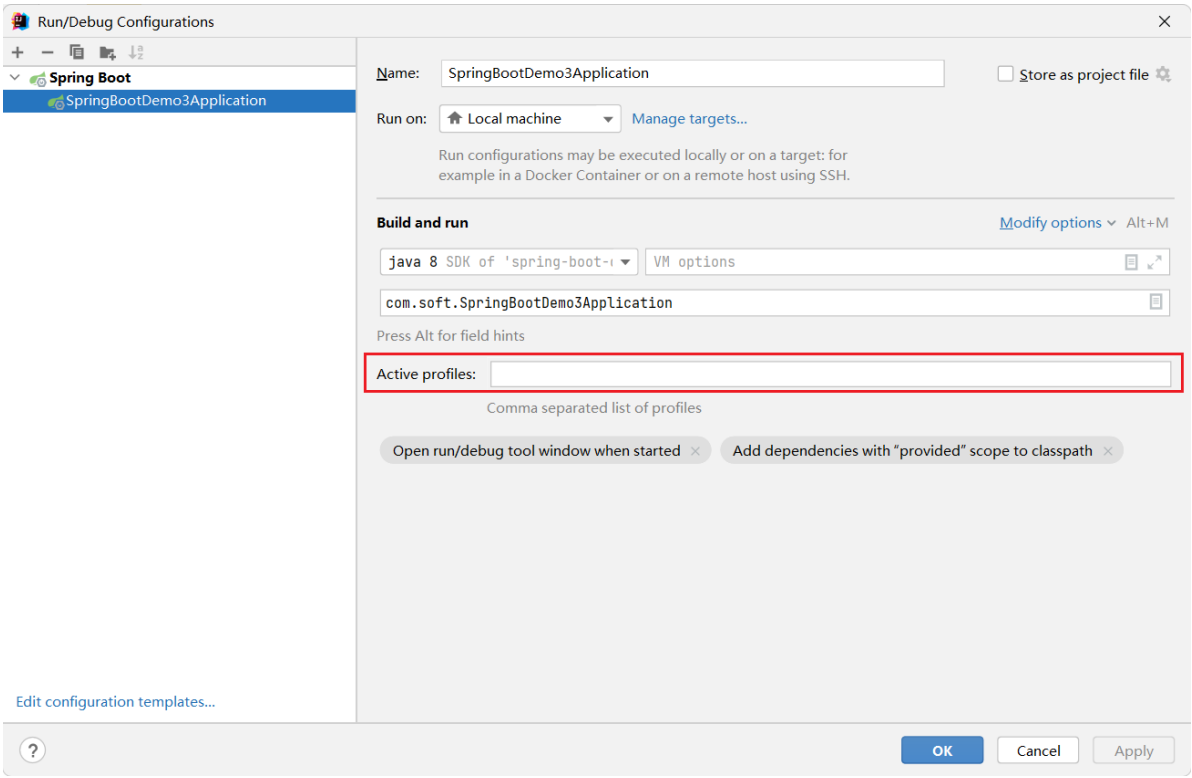
1. 在Program arguments中配置参数：`--spring.profiles.active=xxx` 切换环境（老版本IDEA）



2. 在Program arguments中 VM options下使用命令：`-Dspring.profiles.active=xxx` 切换环境（老版本IDEA）



3. **启动配置中激活指定环境（新版IDEA）**



3. **在核心配置文件中通过激活对应的配置（推荐）**

```yaml
spring:
    profiles:
        active: dev # 通过逗号隔开激活多个配置文件

spring:
    profiles:
        active: pro # 通过逗号隔开激活多个配置文件
```

- **多 profile 文档块模式**

```yaml
server:
    port: 9999
spring:
    profiles:
        active: test # 指定激活配置，通过逗号隔开激活多个配置文件

# 使用横杠分割
---
server:
    port: 8081

spring:
    profiles: dev

# 使用横杠分割
---
server:
    port: 8082
spring:
    profiles: test
```

1.2.4 FAQ 手册

1.2.5 作业实践

1. 简述 yml 文件和 properties 文件的区别。
2. 创建学生对象，包含学号、姓名、年龄等属性，通过配置文件的形式给学生对象赋值。
3. 创建开发环境、测试环境、生产环境配置文件，并通过不同方式切换环境。

1.2.6 面试宝典

1.2.7 拓展资料

1.3 SpringBoot 集成 MyBatis

1.3.1 课程目标

- 掌握 SpringBoot 配置 MyBatis
- 掌握 SpringBoot 集成 MyBatis 实现 CRUD
- 掌握 SpringBoot 数据库连接池

1.3.2 知识架构树

- SpringBoot 配置集成 MyBatis
- SpringBoot 配置数据库连接池
- SpringBoot 配置分页插件

1.3.3 理论知识

1.3.3.1 SpringBoot 集成MyBatis

1）导包

```
<!-- MyBatis启动器 -->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.2.2</version>
</dependency>
<!-- 数据库驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

2）启动类中配置接口扫描及引入公共数据库配置文件

jdbc.properties

```properties
# 数据库链接信息
mysql.driver=com.mysql.cj.jdbc.Driver
mysql.url=jdbc:mysql://localhost:3306/297
mysql.username=root
mysql.password=root

oracle.driver=oracle.jdbc.OracleDriver
oracle.url=jdbc:oracle:thin:@localhost:1521:orcl
oracle.username=user
oracle.password=pass
```

启动类

```java
@SpringBootApplication
// 扫描mapper接口
@MapperScan(basePackages = "com.soft.mapper")
// 加载jdbc.properties文件
@PropertySource(value = {"classpath:jdbc.properties"})
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

3）配置文件中配置数据源和映射文件

```yaml
# 数据源
spring:
  datasource:
    driver-class-name: ${jdbc.driver}
    url: ${jdbc.url}
    username: ${jdbc.username}
    password: ${jdbc.password}

# mybatis配置
mybatis:
  type-aliases-package: com.soft.entity # 实体类起别名
  mapper-locations: com/soft/mapper/*.xml # 映射文件扫描，接口和映射文件在同包下时，可不用配置
  configuration:
    call-setters-on-nulls: true # map映射为空
```

1.3.3.2 SpringBoot 配置数据库连接池

1）导包

```xml
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.2.11</version>
</dependency>
```

2）配置文件

```yaml
# 数据源
spring:
  datasource:
    # 数据库连接池
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: ${jdbc.driver}
    url: ${jdbc.url}
    username: ${jdbc.username}
    password: ${jdbc.password}
    # 初始化大小，最小，最大
    druid:
      initial-size: 5
      min-idle: 5
      max-active: 20
      # 配置获取连接等待超时的时间
      max-wait: 60000
      # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
      time-between-eviction-runs-millis: 60000
      # 配置一个连接在池中最小生存的时间，单位是毫秒
      min-evictable-idle-time-millis: 300000

# mybatis配置
mybatis:
  type-aliases-package: com.soft.entity # 实体类起别名
  mapper-locations: com/soft/mapper/*.xml # 映射文件扫描
  configuration:
    call-setters-on-nulls: true # map映射为空
```

1.3.3.3 SpringBoot 配置PageHelper

1）导包

```xml
<!-- PageHelper分页插件包 -->
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper-spring-boot-starter</artifactId>
    <version>1.4.3</version>
</dependency>
```

2）默认配置即可满足需求

3）使用分页插件

```java
public String queryAll(Student student, Model model){
    // 使用分页插件
    PageHelper.startPage(1, 5);
    List<Student> list = studentService.queryAll(null);
    // 分页信息
    PageInfo pageInfo = new PageInfo<>(list);
    // 封装页面所需分页信息
    model.addAttribute("pageInfo", pageInfo);

    return "list.html";
}
```

1.3.4 FAQ 手册

1.3.5 作业实践

1.3.6 面试宝典

- SpringBoot 配置集成 MyBaits
- 实现CRUD操作。
- 实现分页功能。

1.3.7 拓展资料

1.4 SpringBoot 集成 Log4j2

1.4.1 课程目标

- 掌握 Log4j2 配置

1.4.2 知识架构树

- SpringBoot 集成 Log4j2

1.4.3 理论知识

1.4.3.1 日志概念

> 日志即记录代码执行过程，其包含了代码执行过程中的重要信息。通过日志，可以协助开发人员分析系统，快速定位错误。

1.4.3.2 集成步骤

> 由于 SpringBoot 默认是用 logback 的日志框架，所以需要排除 logback ，不然会出现 jar 依赖冲突的报错；

1）导包

```
<!-- 主启动器依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <!-- 禁用默认的logback -->
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<!-- 引入log4j2 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

2）将`log4j2.xml`文件拷贝到resources目录下即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    status：全局的日志级别，TRACE < DEBUG < INFO < WARN < ERROR < FATAL
        trace：追踪，就是程序推进一下，可以写个trace输出
        debug：调试，一般作为最低级别，trace基本不用。
        info：输出重要的信息，使用较多
        warn：警告，有些信息不是错误信息，但也要给程序员一些提示。
        error：错误信息。用的也很多。
        fatal：致命错误。
-->
<Configuration status="warn">
    <!-- 配置全局属性 -->
    <properties>
        <!-- 文件路径 -->
        <property name="LOG_HOME">logs</property>
        <!-- 文件名字 -->
        <property name="FILE_NAME">app.log</property>
        <!-- 全局布局 -->
        <property name="LAYOUT">%d [%p] [%t] [%c] %m%n</property>
    </properties>

    <!-- Appenders：定义日志输出目的地，内容和格式等 -->
    <Appenders>
        <!--
            输出日志到控制台
            name：控制台名称，方便后期操作
            target：输出日志的类型，错误还是信息
                SYSTEM_OUT
                SYSTEM_ERR
        -->
        <Console name="console" target="SYSTEM_ERR">
            <!--
                信息布局
                pattern：信息布局形式
                    %p：日志级别
                    %t：当前线程
                    %d：日期 YYYY-MM-dd HH/hh mm ss
                    %c：类名
                    %msg、%m：日志信息
                    %n：换行
                    [%p][%t] %d{HH:MM:ss} %c %msg%n
            -->
            <PatternLayout pattern="${LAYOUT}"></PatternLayout>
        </Console>
        <!--
            输出日志到文件
            name：操作文件对象的名字
            fileName：文件名字
            filePattern：路径格式
        -->
        <RollingFile name="rollingFile" fileName="logs/app.log"
filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
            <PatternLayout pattern="${LAYOUT}"></PatternLayout>
```

```
            <!--配置文件使用原则-->
            <Policies>
                <TimeBasedTriggeringPolicy/>
                <SizeBasedTriggeringPolicy size="250 MB"/>
            </Policies>
        </RollingFile>
    </Appenders>
    <Loggers>
        <!--
            自定义日志，如果不指定appender，则使用继承自root的日志类型
            name：用于指定哪些包或者哪些类使用日志
            level：日志级别
            additivity：配置不使用父类的日志
        -->
        <Logger name="com.soft" level="trace" additivity="false">
            <AppenderRef ref="console"/>
        </Logger>
        <Logger name="org.springframework" level="INFO"></Logger>
        <!--根日志配置，要引入目的地-->
        <Root level="error">
            <AppenderRef ref="console"/>
            <AppenderRef ref="rollingFile"/>
        </Root>
    </Loggers>
</Configuration>
```

1.4.4 FAQ 手册

1.4.5 作业实践

- SpringBoot 集成配置 Log4j2

1.4.6 面试宝典

1.4.7 拓展资料

1.5 SpringBoot 集成 Thymeleaf

1.5.1 课程目标

- 掌握模板引擎理论
- 掌握 Thymeleaf 语法

1.5.2 知识架构树

- 模板引擎概述
- Thymeleaf 语法

1.5.3 理论知识

1.5.3.1 模板引擎概述

1）Thymeleaf 简介

> Thymeleaf 是面向 Web 和独立环境的现代服务器端 Java 模板引擎，能够处理HTML，XML，JavaScript，CSS 甚至纯文本。
>
> Thymeleaf 旨在提供一个优雅的、高度可维护的创建模板的方式。 为了实现这一目标，Thymeleaf 建立在自然模板（HTML页面）的概念上，**将其逻辑注入到模板文件中，不会影响模板设计原型**（不影响原来的样式以及内容）。 **这改善了设计的沟通，弥合了设计和开发团队之间的差距。**
>
> Thymeleaf 从设计之初就遵循Web标准——特别是HTML5标准 ，如果需要，Thymeleaf 允许您创建完全符合HTML5验证标准的模板。
>
> 官网地址：https://www.thymeleaf.org/
>
> 模板引擎需要通过后台 Java 代码渲染，所以不能直接请求页面，需要通过后台跳转页面。
>
> **SpringBoot 提供了一个目录 `/templates/`，用于存放自己的模板（html页面），这个目录和 `/WEB-INF/` 目录一样下是受保护的，不能直接请求，需要通过后台的 Java 代码，进行转发访问。**

2）SpringBoot集成模板引擎

- 导入依赖

```
<!-- 模板引擎启动器 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

- 在HTML页面中添加 Thymeleaf 语法支持（头信息、命名空间）

```
<html xmlns:th="http://www.thymeleaf.org">
```

- 默认配置

> 不需要做任何配置，启动器已经帮我们把Thymeleaf的视图器配置完成，而且，还配置了模板文件（html）的位置。
>
> 默认前缀：classpath:/templates/ 默认后缀：.html 所以如果我们返回视图：list，会指向到classpath:/templates/list.html

> Thymeleaf默认会开启页面缓存，提高页面并发能力。但会导致我们修改页面不会立即被展现，因此我们关闭缓存，修改完毕页面，需要使用快捷键`Ctrl + Shift + F9`重新编译工程。

```
# 关闭 Thymeleaf 的缓存
spring:
  thymeleaf:
    cache: false
```

- 代码探究

> Spring Boot 通过 ThymeleafAutoConfiguration 自动配置类对 Thymeleaf 提供了一整套的自动化配置方案，该自动配置类的部分源码如下：

```
@Configuration(proxyBeanMethods = false)
@EnableConfigurationProperties({ThymeleafProperties.class})
@ConditionalOnClass({TemplateMode.class, SpringTemplateEngine.class})
@AutoConfigureAfter({WebMvcAutoConfiguration.class,
WebFluxAutoConfiguration.class})
public class ThymeleafAutoConfiguration {
}
```

> ThymeleafAutoConfiguration 使用 @EnableConfigurationProperties 注解导入了 ThymeleafProperties 类，该类包含了与 Thymeleaf 相关的自动配置属性，其部分源码如下：

```
@ConfigurationProperties(
    prefix = "spring.thymeleaf"
)
public class ThymeleafProperties {
    private static final Charset DEFAULT_ENCODING;
    public static final String DEFAULT_PREFIX = "classpath:/templates/";
    public static final String DEFAULT_SUFFIX = ".html";
    private boolean checkTemplate = true;
    private boolean checkTemplateLocation = true;
    private String prefix = "classpath:/templates/";
    private String suffix = ".html";
    private String mode = "HTML";
    private Charset encoding;
    private boolean cache;

    ...
}
```

ThymeleafProperties 通过 @ConfigurationProperties 注解将配置文件（application.properties/yml） 中前缀为 spring.thymeleaf 的配置和这个类中的属性绑定。

在 ThymeleafProperties 中还提供了以下静态变量：

```
DEFAULT_ENCODING：默认编码格式
DEFAULT_PREFIX：视图解析器的前缀
DEFAULT_SUFFIX：视图解析器的后缀
```

根据以上配置属性可知，Thymeleaf 模板的默认位置在 resources/templates 目录下，默认的后缀是 html，即只要将 HTML 页面放在`classpath:/templates/`下，Thymeleaf 就能自动进行渲染。与 Spring Boot 其他自定义配置一样，我们可以在`application.yml`中修改以 spring.thymeleaf 开始的属性，以实现修改 Spring Boot 对 Thymeleaf 的自动配置的目的。

1.5.3.2 Thymeleaf语法

> Thymeleaf 的使用将其逻辑注到模板件中，不会影响模板设计原型。基于这个目标我们不能随意的改变HTML的布局。所以只能在原有的标签上做文章。标签里面有属性，通过属性进行动态的数据渲染。

1.5.3.2.1 操作HTML属性

> Thymeleaf 封装了所有的HTML属性，以【th:】开头，后跟上属性名称，`th:xxx`，大部分 Thymeleaf 属性中都可以使用表达式来完成需求。

```
th:value -> 给 value 属性赋值，一般都是 <input th:value=""/> 框
th:id -> 给 id 属性赋值 <p th:id=""></p>
th:src -> 给 src 属性赋值 <img th:src="" />
th:href -> 给 href 属性赋值 <a th:href=""></a>

th:text -> 给HTML标签中添加文本，会把字符原样输出
th:utext -> 给HTML标签中添加文本，会把字符中的特殊字符转义

th:if -> 判断，满足条件展示当前元素。
th:each -> 遍历，和th:text或th:value一起使用
th:object -> 引入对象/声明变量，和*{}一起使用
th:attr -> 替换属性
th:field -> 用来绑定后台对象和表单数据

# 页面引入
th:fragment -> 定义引入内容
th:insert -> 引入其他页面内容，将被引用的模板片段插入到自己的标签体中
th:replace -> 引入其他页面内容，将被引用的模板片段替换掉自己
th:incloud -> 引入其他页面内容，类似于 th:insert，而不是插入片段，它只插入此片段的内容。Thymeleaf 3.0 之后不再推荐使用 th:include.
```

属性	属性	属性
`th:abbr`	`th:accept`	`th:accept-charset`
`th:accesskey`	`th:action`	`th:align`
`th:alt`	`th:archive`	`th:audio`
`th:autocomplete`	`th:axis`	`th:background`
`th:bgcolor`	`th:border`	`th:cellpadding`
`th:cellspacing`	`th:challenge`	`th:charset`
`th:cite`	`th:class`	`th:classid`
`th:codebase`	`th:codetype`	`th:cols`
`th:colspan`	`th:compact`	`th:content`
`th:contenteditable`	`th:contextmenu`	`th:data`
`th:datetime`	`th:dir`	`th:draggable`
`th:dropzone`	`th:enctype`	`th:for`
`th:form`	`th:formaction`	`th:formenctype`
`th:formmethod`	`th:formtarget`	`th:fragment`

属性	属性	属性
th:frame	th:frameborder	th:headers
th:height	th:high	th:href
th:hreflang	th:hspace	th:http-equiv
th:icon	th:id	th:inline
th:keytype	th:kind	th:label
th:lang	th:list	th:longdesc
th:low	th:manifest	th:marginheight
th:marginwidth	th:max	th:maxlength
th:media	th:method	th:min
th:name	th:onabort	th:onafterprint
th:onbeforeprint	th:onbeforeunload	th:onblur
th:oncanplay	th:oncanplaythrough	th:onchange
th:onclick	th:oncontextmenu	th:ondblclick
th:ondrag	th:ondragend	th:ondragenter
th:ondragleave	th:ondragover	th:ondragstart
th:ondrop	th:ondurationchange	th:onemptied
th:onended	th:onerror	th:onfocus
th:onformchange	th:onforminput	th:onhashchange
th:oninput	th:oninvalid	th:onkeydown
th:onkeypress	th:onkeyup	th:onload
th:onloadeddata	th:onloadedmetadata	th:onloadstart
th:onmessage	th:onmousedown	th:onmousemove
th:onmouseout	th:onmouseover	th:onmouseup
th:onmousewheel	th:onoffline	th:ononline
th:onpause	th:onplay	th:onplaying
th:onpopstate	th:onprogress	th:onratechange
th:onreadystatechange	th:onredo	th:onreset
th:onresize	th:onscroll	th:onseeked

属性	属性	属性
th:onseeking	th:onselect	th:onshow
th:onstalled	th:onstorage	th:onsubmit
th:onsuspend	th:ontimeupdate	th:onundo
th:onunload	th:onvolumechange	th:onwaiting
th:optimum	th:pattern	th:placeholder
th:poster	th:preload	th:radiogroup
th:rel	th:rev	th:rows
th:rowspan	th:rules	th:sandbox
th:scheme	th:scope	th:scrolling
th:size	th:sizes	th:span
th:spellcheck	th:src	th:srclang
th:standby	th:start	th:step
th:style	th:summary	th:tabindex
th:target	th:title	th:type
th:usemap	th:value	th:valuetype
th:vspace	th:width	th:wrap
th:xmlbase	th:xmllang	th:xmlspace

1.5.3.2.2 标准表达式语法

- 简单表达式

```
变量表达式：${}
    从环境的上下文中获取信息，比如：session、request、ModelAndView、ModelMap、Model，可以理解为JSP中的 EL 表达式 ${}：
    modelAndView.addObject("msg", "hello");
    [[${msg}]]
    <h1 th:text="${msg}"></h1>

选择表达式：*{}
    一般搭配 <th:object> 使用，用于获取对象中属性内容
    modelAndView.addObject("student", student);

    <div th:object="${student}">
        <p th:text="*{name}"></p>
        <p th:text="*{age}"></p>
        <p th:text="*{sex}"></p>
    </div>

```

```
信息表达式：#{}
    常用于获取properties配置中的值，可以用于国际化操作

连接表达式：@{}
    设置连接/路径信息，比如静态资源路径（JS 路径、CSS 路径）；

    <a th:href="@{https://www.baidu.com}">百度一下</a>
    <img th:src="@{/img/13.png}">
    带参数路径：@{xxx/xxx/(k1=v1,k2=v2...)}

片段表达式：~{}
    引入其他页面信息字面量
```

- 字面量

```
普通字符: 'one text', 'Another one!' ...
数字: 0, 34, 3.0, 12.3 ...
布尔: true, false
空对象: null
文字令牌: one, sometext, main, ...
    文字令牌只支持字母、数字、and、括号[]、点.、横杠-、下划线_
    不支持空格逗号等。

<p th:text="one text"></p>
<p th:text="${'one text'}"></p>
<p th:text="${110}"></p>
<p th:text="${true}"></p>
<p th:text="${null}"></p>
<p th:text="${abc}"></p>
```

- 字符操作

```
字符拼接：+
字符替换：|The name is ${name}|

<p th:text="${'a' + 'b'}">
<p th:text="${'我叫' + name + ',今年' + age + '岁'}">

<p th:text="|The name is ${name}|"> 在不使用加号拼接时，需要使用竖线包围
<p th:text="|我叫${name},今年${age}岁|"> 在不使用加号拼接时，需要使用竖线包围
```

- 算数运算符

```
数学运算符：+, -, *, /, %
负数：-

<p th:text="${1 + 2}"></p>
```

- 布尔运算符

```
and（&&）、or（||）、not（!）

<p th:text="${1 == 1}"></p>

<p th:text="${1 == 1 and  2 == 2}"></p>
<p th:text="${1 == 1 &&  2 == 2}"></p>

<p th:text="${1 == 1 or  2 == 2}"></p>
<p th:text="${1 == 1 ||  2 != 2}"></p>
```

- 比较运算符

```
比较运算符：>, <, >=, <= (gt, lt, ge, le)
等值运算符：==, != (eq, ne)

<p th:text="${1 <= 2}"></p>
```

- 条件运算符

```
If-then: (if) ? (then)
If-then-else: (if) ? (then) : (else)
Default: (value) ?: (defaultvalue)
    value为null或者空字符串时显示默认值

<p th:text="${'1' == '1'} ? '男' : '女'"></p>
<p th:text="${'0' == '1'} ? '男' : '女'"></p>
```

- 特殊字符

```
无操作(什么都不做)：_，常用于必须要保证表达式语法完整的时候使用，比如三元表达式。

<p th:text="${18 < 18} ? '未成年' : _"></p>
<p th:text="${20 < 18} ? '未成年' : _"></p>
```

- 内置对象

```
ctx：上下文对象
vars：上下文变量
locale：上下文语言环境
request：Web环境下的HttpServletRequest对象
response：Web环境下的HttpServletResponse对象
session：Web环境下的HttpSession对象
servletContext：Web环境下的HttpContext对象
```

1.5.3.3 常用写法

- **遍历集合以及判定**

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
```

```
    <title>Title</title>
</head>
<body>
<table border="1" align="center">
    <tr>
        <th>下标</th>
        <th>序号</th>
        <th>学号</th>
        <th>姓名</th>
        <th>班级</th>
        <th>性别</th>
        <th>电话</th>
        <th>地址</th>
        <th>专业</th>
        <th>操作</th>
    </tr>
    <!--
        遍历控制层代码放在域对象中的 list 集合。
        th:each 和 Java 中的增强型 for 循环类似
            for(Object obj : list){}
        值得一提的是，thymeleaf 提供了一状态对象，定义方法如下案例，在遍历内容 obj 后面通过逗号隔开
    -->
    <tr th:each="stu,status:${list}">
        <!-- 获取状态对象中当前元素的下标，从 0 开始 -->
        <td th:text="${status.index}"></td>
        <!-- 获取状态对象中当前元素的序号，从 1 开始 -->
        <td th:text="${status.count}"></td>
        <td th:text="${stu.name}"></td>
        <td th:text="${stu.clazz}"></td>
        <td th:text="${#strings.equals('1', stu.sex) ? '男':'女'}"></td>
        <td th:text="${stu.tel}"></td>
        <td th:text="${stu.address}"></td>
        <td th:text="${stu.mno}"></td>
        <td>
            <button type="button" th:if="${stu.delFlg == '0'}">启用</button>
            <button type="button" th:if="${stu.delFlg == '1'}">禁用</button>
            <button type="button" th:onclick="edit([[${stu.no}]])">编辑</button>
        </td>
    </tr>
</table>
<script>
    function edit(no){
        // 请求地址，跳转修改页面；URL 占位符传参
        location.href='/stu/edit/' + no;
    }
</script>
</body>
</html>
```

- **单对象取值**

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>修改页面</h1>
    <form action="/stu/edit" method="post">
        <!-- th:object 接收后台传递对象 -->
        <table th:object="${student}">
            <tr>
                <td>学号</td>
                <!-- *{attr} 获取对象中的值 -->
                <td><input type="text" name="no" th:value="*{no}" readonly/>
</td>
            </tr>
            <tr>
                <td>姓名</td>
                <td><input type="text" name="name" th:value="*{name}"
readonly/></td>
            </tr>
            <tr>
                <td>性别</td>
                <td>
                    <!-- th:field 获取对象中的值 -->
                    <label><input type="radio" name="sex" value="1"
th:field="*{sex}"/>男</label>
                    <label><input type="radio" name="sex" value="0"
th:field="*{sex}"/>女</label>
                </td>
            </tr>
            <tr>
                <td>电话</td>
                <td><input type="text" name="tel" th:value="*{tel}"/></td>
            </tr>
            <tr>
                <td>地址</td>
                <td><input type="text" name="address" th:value="*
{address}"/></td>
            </tr>
            <tr>
                <td>班级</td>
                <td><input type="text" name="clazz" th:value="*{clazz}"/>
</td>
            </tr>
            <tr>
                <td>专业</td>
                <td>
                    <select th:field="*{mno}" name="mno">
                        <option value="">请选择专业</option>
                        <option value="M10001">软件工程</option>
                        <option value="M10002">土木工程</option>
                        <option value="M10003">信息工程</option>
                        <option value="M10004">电竞专业</option>
                        <option value="M10005">美术</option>
                        <option value="M10006">体育</option>
```

```
                    </select>
                </td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="修改"/></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

- **th:onclick 动态参数**

```
<button type="button" th:onclick="|edit(${stu.no})|">编辑</button>
<button type="button" th:onclick="edit([[${stu.no}]])">编辑</button>
```

- **引入公共内容**

应用场景：

1. 引入公共的导航（页面头部）或者版权信息（页面尾部）
2. 引入页面中的部分内容（导入插件）

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div style="background-color:red;" th:fragment="top">
        这里是头信息的导航
    </div>
    <div style="background-color:blue;" th:fragment="bottom">
        域名的备案信息
    </div>
    <div th:fragment="resource">
        <link href=""/>
        <script src=""></script>
    </div>
</body>
</html>
```

```
<!DOCTYPE html>
<!--在HTML标签中引入thymeleaf的语法库-->
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 方式一 -->
    <th:block th:include="~{static::resource}"/>
</head>
<body>
    <!-- 方式二 -->
```

```
    <div th:insert="~{static :: top}"></div>
    <!-- 方式三 -->
    <div th:replace="~{static :: bottom}"></div>
</body>
</html>
```

> 某些情况下，引入公共内容是需要一些参数，可以使用以下结构。

1. 模板名::选择器名或片段名(参数1=参数值1,参数2=参数值2)
2. 模板名::选择器名或片段名(参数值1,参数值2)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div style="background-color:red;" th:fragment="top(arg1, arg2)">
        这里是头信息的导航[[${arg1}]]
        <span th:text="${arg2}"></span>
    </div>
    <div style="background-color:blue;" th:fragment="bottom(arg1, arg2)">
        域名的备案信息
    </div>
</body>
</html>
```

```
<!DOCTYPE html>
<!--在HTML标签中引入thymeleaf的语法库-->
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <!-- 方式二 -->
    <div th:insert="~{top :: top('hello', '你好! ')}"></div>
    <!-- 方式三 -->
    <div th:replace="~{top :: bottom}"></div>
</body>
</html>
```

1.5.3.4 工具类

> 工具类的使用通过 `#类名` 的方式调用。

Execution Info

> execInfo：提供有关在Thymeleaf标准表达式内正在处理的模板的信息。

```
/*
 *    ======================================================
 *    See    javadoc    API    for    class
org.thymeleaf.expression.ExecutionInfo
 *    ======================================================
 */
${#execInfo.templateName}
${#execInfo.templateMode}
${#execInfo.processedTemplateName}
${#execInfo.processedTemplateMode}
${#execInfo.templateNames}
${#execInfo.templateModes}
${#execInfo.templateStack}
```

Messages

> messages：用于在变量表达式中获取外部化消息的工具方法，与使用#{...}语法获得的方式相同。

```
/*
 *    ======================================================
 *    See    javadoc    API    for    class
org.thymeleaf.expression.Messages
 *    ======================================================
 */
${#messages.msg('msgKey')}
${#messages.msg('msgKey',  param1)}
${#messages.msg('msgKey', param1, param2)}
${#messages.msg('msgKey', param1, param2, param3)}
${#messages.msgWithParams('msgKey', new Object[]  {param1, param2, param3,
param4})}
${#messages.arrayMsg(messageKeyArray)}
${#messages.listMsg(messageKeyList)}
${#messages.setMsg(messageKeySet)}
${#messages.msgOrNull('msgKey')}
${#messages.msgOrNull('msgKey', param1)}
${#messages.msgOrNull('msgKey', param1, param2)}
${#messages.msgOrNull('msgKey', param1, param2, param3)}
${#messages.msgOrNullWithParams('msgKey', new Object[] {param1, param2,
param3,param4})}
${#messages.arrayMsgOrNull(messageKeyArray)}
${#messages.listMsgOrNull(messageKeyList)}
${#messages.setMsgOrNull(messageKeySet)}
```

URIs/URLs

> uris：用于在Thymeleaf标准表达式中执行URI/URL操作（尤其是转义/取消转义）的工具对象。

```
/*
 *  ======================================================
 *  See  javadoc  API  for  class  org.thymeleaf.expression.Uris
 *  ======================================================
 */

```

```
${#uris.escapePath(uri)}
${#uris.escapePath(uri,  encoding)}
${#uris.unescapePath(uri)}
${#uris.unescapePath(uri,  encoding)}

/*
 *  Escape/Unescape  as a URI/URL path segment (between '/' symbols)
 */
${#uris.escapePathSegment(uri)}
${#uris.escapePathSegment(uri,  encoding)}
${#uris.unescapePathSegment(uri)}
${#uris.unescapePathSegment(uri,  encoding)}

/*
 *  Escape/Unescape  as  a Fragment Identifier (#frag)
 */
${#uris.escapeFragmentId(uri)}
${#uris.escapeFragmentId(uri,  encoding)}
${#uris.unescapeFragmentId(uri)}
${#uris.unescapeFragmentId(uri,  encoding)}

/*
 *  Escape/Unescape  as  a Query  Parameter (?var=value)
 */
${#uris.escapeQueryParam(uri)}
${#uris.escapeQueryParam(uri, encoding)}
${#uris.unescapeQueryParam(uri)}
${#uris.unescapeQueryParam(uri, encoding)}
```

Conversions

> conversions：允许在模板任意位置执行转换服务的实用程序对象

```
/*
 *    ======================================================
 *    See    javadoc    API    for    class
org.thymeleaf.expression.Conversions
 *    ======================================================
 */
/*
 *    Execute    the    desired    conversion    of    the    'object' value
into    the
 *    specified class.
 */
${#conversions.convert(object, 'java.util.TimeZone')}
${#conversions.convert(object, targetClass)}
```

Dates

> dates：java.util.Date对象的实用程序方法：

```
/*
 *    ======================================================
 *    See    javadoc    API    for    class org.thymeleaf.expression.Dates
 *    ======================================================
```

```
 */
/*
 *    Format date    with the standard locale format Also works with
arrays,lists or sets
 */
${#dates.format(date)}
${#dates.arrayFormat(datesArray)}
${#dates.listFormat(datesList)}
${#dates.setFormat(datesSet)}

/*
 *    Format    date with the ISO8601 format Also works    with arrays,
lists or sets
 */
${#dates.formatISO(date)}
${#dates.arrayFormatISO(datesArray)}
${#dates.listFormatISO(datesList)}
${#dates.setFormatISO(datesSet)}

/*
 *    Format date    with the specified    pattern Also works with
arrays, lists or sets
 */
${#dates.format(date, 'dd/MMM/yyyy HH:mm')}
${#dates.arrayFormat(datesArray, 'dd/MMM/yyyy HH:mm')}
${#dates.listFormat(datesList, 'dd/MMM/yyyy HH:mm')}
${#dates.setFormat(datesSet, 'dd/MMM/yyyy HH:mm')}

/*
 *    Obtain date properties Also    works with arrays, lists or    sets
 */
${#dates.day(date)}                // also arrayDay(...), listDay(...), etc.
${#dates.month(date)}              // also arrayMonth(...), listMonth(...),
etc.
${#dates.monthName(date)}          // also arrayMonthName(...),
listMonthName(...), etc.
${#dates.monthNameShort(date)}     // also
arrayMonthNameShort(..),listMonthNameShort(..),etc.
${#dates.year(date)}               // also arrayYear(...), listYear(...),
etc.
${#dates.dayOfWeek(date)}          // also arrayDayOfWeek(...),
listDayOfWeek(...), etc.
${#dates.dayOfWeekName(date)}      // also
arrayDayOfWeekName(..),listDayOfWeekName(..), etc.
${#dates.dayOfWeekNameShort(date)} // also arrayDayOfWeekNameShort(...),
listDayOfWeekNameShort(...), etc.
${#dates.hour(date)}               // also arrayHour(...), listHour(...),
etc.
${#dates.minute(date)}             // also arrayMinute(...),
listMinute(...), etc.
${#dates.second(date)}             // also arraySecond(...),
listSecond(...), etc.
${#dates.millisecond(date)}        // also arrayMillisecond(...),
listMillisecond(...), etc.

```

```
/*
 *    Create    date (java.util.Date) objects    from its components
 */
${#dates.create(year,month,day)}
${#dates.create(year,month,day,hour,minute)}
${#dates.create(year,month,day,hour,minute,second)}
${#dates.create(year,month,day,hour,minute,second,millisecond)}

/*
 *    Create a date (java.util.Date) object for the    current    date and time
 */
${#dates.createNow()}
${#dates.createNowForTimeZone()}

/*
 *    Create a date (java.util.Date) object for the current date (time set to    00:00)
 */
${#dates.createToday()}
${#dates.createTodayForTimeZone()}
```

Calendars

> calendars：类似于#dates，但对于java.util.Calendar对象

```
/*
 *    =======================================================
 *    See    javadoc    API    for    class    org.thymeleaf.expression.Calendars
 *    =======================================================
 */
/*
 *    Format calendar    with the standard locale format Also works with arrays,lists or sets
 */
${#calendars.format(cal)}
${#calendars.arrayFormat(calArray)}
${#calendars.listFormat(calList)}
${#calendars.setFormat(calSet)}

/*
 *    Format    calendar with the ISO8601 format Also works    with arrays, lists or sets
 */
${#calendars.formatISO(cal)}
${#calendars.arrayFormatISO(calArray)}
${#calendars.listFormatISO(calList)}
${#calendars.setFormatISO(calSet)}

/*
 *    Format calendar    with the specified pattern Also    works with arrays, lists or    sets
 */
${#calendars.format(cal, 'dd/MMM/yyyy HH:mm')}
```

```
${#calendars.arrayFormat(calArray, 'dd/MMM/yyyy HH:mm')}
${#calendars.listFormat(calList, 'dd/MMM/yyyy HH:mm')}
${#calendars.setFormat(calSet, 'dd/MMM/yyyy HH:mm')}

/*
 *    Obtain calendar properties Also    works with arrays,lists    or
sets
 */
${#calendars.day(date)}                // also arrayDay(...), listDay(...),
etc.
${#calendars.month(date)}              // also arrayMonth(...),
listMonth(...), etc.
${#calendars.monthName(date)}          // also arrayMonthName(...),
listMonthName(...), etc.
${#calendars.monthNameShort(date)}     // also arrayMonthNameShort(..),
listMonthNameShort(..),etc.
${#calendars.year(date)}               // also arrayYear(...),
listYear(...), etc.
${#calendars.dayOfWeek(date)}          // also arrayDayOfWeek(...),
listDayOfWeek(...), etc.
${#calendars.dayOfWeekName(date)}      // also arrayDayOfWeekName(...),
listDayOfWeekName(...), etc.
${#calendars.dayOfWeekNameShort(date)} // also arrayDayOfWeekNameShort(...),
listDayOfWeekNameShort(...), etc.
${#calendars.hour(date)}               // also arrayHour(...),
listHour(...), etc.
${#calendars.minute(date)}             // also arrayMinute(...),
listMinute(...), etc.
${#calendars.second(date)}             // also arraySecond(...),
listSecond(...), etc.
${#calendars.millisecond(date)}        // also
arrayMillisecond(...),listMillisecond(...), etc.

/*
 *    Create    calendar (java.util.Calendar) objects from    its
components
 */
${#calendars.create(year,month,day)}
${#calendars.create(year,month,day,hour,minute)}
${#calendars.create(year,month,day,hour,minute,second)}
${#calendars.create(year,month,day,hour,minute,second,millisecond)}
${#calendars.createForTimeZone(year,month,day,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,second,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,second,millisecond
,timeZone)}

/*
 *    Create a calendar (java.util.Calendar) object for the current date and
   time
 */
${#calendars.createNow()}
${#calendars.createNowForTimeZone()}

/*
```

```
 *    Create a calendar(java.util.Calendar) object for the current date (time set to 00:00)
 */
${#calendars.createToday()}
${#calendars.createTodayForTimeZone()}
```

Numbers

> numbers：数字对象的实用程序方法：

```
/*
 *    ======================================================
 *    See    javadoc    API    for    class org.thymeleaf.expression.Numbers
 *    ======================================================
 */
/*
 *    Set    minimum    integer    digits. Also works with    arrays, lists    or    sets
 */
${#numbers.formatInteger(num,3)}
${#numbers.arrayFormatInteger(numArray,3)}
${#numbers.listFormatInteger(numList,3)}
${#numbers.setFormatInteger(numSet,3)}

/*
 *    Set    minimum    integer    digits    and    thousands    separator:
 *    'POINT', 'COMMA', 'WHITESPACE', 'NONE' or 'DEFAULT'(by locale).
 *    Also works with    arrays, lists    or    sets
 */
${#numbers.formatInteger(num,3,'POINT')}
${#numbers.arrayFormatInteger(numArray,3,'POINT')}
${#numbers.listFormatInteger(numList,3,'POINT')}
${#numbers.setFormatInteger(numSet,3,'POINT')}

/*
 *    ========================
 *    Formatting    decimal    numbers
 *    ========================
 */
/*
 *    Set    minimum    integer    digits and (exact) decimal digits.Also works with arrays, lists    or sets
 */
${#numbers.formatDecimal(num,3,2)}
${#numbers.arrayFormatDecimal(numArray,3,2)}
${#numbers.listFormatDecimal(numList,3,2)}
${#numbers.setFormatDecimal(numSet,3,2)}


/*
 *    Set    minimum    integer    digits and    (exact)    decimal digits,and also decimal    separator.
 *    Also works with    arrays, lists    or    sets
 */
```

```
${#numbers.formatDecimal(num,3,2,'COMMA')}
${#numbers.arrayFormatDecimal(numArray,3,2,'COMMA')}
${#numbers.listFormatDecimal(numList,3,2,'COMMA')}
${#numbers.setFormatDecimal(numSet,3,2,'COMMA')}

/*
 *    Set    minimum    integer    digits    and    (exact)    decimal
digits,and also    thousands and
 *    decimal    separator. Also    works    with    arrays, lists    or
sets
 */
${#numbers.formatDecimal(num,3,'POINT',2,'COMMA')}
${#numbers.arrayFormatDecimal(numArray,3,'POINT',2,'COMMA')}
${#numbers.listFormatDecimal(numList,3,'POINT',2,'COMMA')}
${#numbers.setFormatDecimal(numSet,3,'POINT',2,'COMMA')}

/ *
 *    ====================
 *    Formatting    currencies
 *    ====================
 */
${#numbers.formatCurrency(num)}
${#numbers.arrayFormatCurrency(numArray)}
${#numbers.listFormatCurrency(numList)}
${#numbers.setFormatCurrency(numSet)}

/ *
 *    ======================
 *    Formatting    percentages
 *    ======================
 */
${#numbers.formatPercent(num)}
${#numbers.arrayFormatPercent(numArray)}
${#numbers.listFormatPercent(numList)}
${#numbers.setFormatPercent(numSet)}


/ *
 *    Set    minimum    integer    digits    and    (exact)    decimal
 digits.
 */
${#numbers.formatPercent(num, 3, 2)}
${#numbers.arrayFormatPercent(numArray, 3, 2)}
${#numbers.listFormatPercent(numList, 3, 2)}
${#numbers.setFormatPercent(numSet, 3, 2)}

/*
 *    ===============
 *    Utility    methods
 *    ===============
 */
/*
 *    Create    a sequence (array) of integer numbers going from x to y
 */
${#numbers.sequence(from,to)}
```

```
${#numbers.sequence(from,to,step)}
```

Strings

> strings String工具类

```
/*
 *    =======================================================
 *    See    javadoc    API    for    class
org.thymeleaf.expression.Strings
 *    =======================================================
 */
/*
 *    Null-safe toString()
 */
${#strings.toString(obj)}    //    also array*, list* and set*

/*
 *    Check whether a String is empty (or null). Performs    a trim()
operation before check
 *    Also works with    arrays, lists or sets
 */
${#strings.isEmpty(name)}
${#strings.arrayIsEmpty(nameArr)}
${#strings.listIsEmpty(nameList)}
${#strings.setIsEmpty(nameSet)}

/*
 *    Perform    an 'isEmpty()' check on    a string and return    it if
false, defaulting    to
 *    another    specified string if    true. Also works with arrays, lists
   or sets
 */
${#strings.defaultString(text,default)}
${#strings.arrayDefaultString(textArr,default)}
${#strings.listDefaultString(textList,default)}
${#strings.setDefaultString(textSet,default)}

/*
 *    Check whether a    fragment is    contained in a String Also works
with arrays,lists or sets
 */
${#strings.contains(name,'ez')}           //    also array*, list* and
set*
${#strings.containsIgnoreCase(name,'ez')} //  also array*, list* and set*

/*
 *    Check whether a    String starts or ends with a fragment
 *    Also works with    arrays, lists or sets
 */
${#strings.startsWith(name,'Don')}           //    also array*, list* and
set*
${#strings.endsWith(name,endingFragment)}    //    also array*, list* and
set*

```

```
/*
 *    Substring-related operations Also works    with arrays, lists or sets
 */
${#strings.indexOf(name,frag)}            //    also array*, list* and
set*
${#strings.substring(name,3,5)}           //    also array*, list* and
set*
${#strings.substringAfter(name,prefix)}   //    also array*, list* and
set*
${#strings.substringBefore(name,suffix)}  //    also array*, list* and
set*
${#strings.replace(name,'las','ler')}     //    also array*, list* and
set*

/*
 *    Append and prepend Also    works with arrays, lists or    sets
 */
${#strings.prepend(str,prefix)}    //    also array*, list* and    set*
${#strings.append(str,suffix)}     //    also array*, list* and    set*

/*
 *    Change    case Also works    with arrays, lists or sets
 */
${#strings.toUpperCase(name)}    //    also array*, list* and    set*
${#strings.toLowerCase(name)}    //    also array*, list* and    set*

/*
 *    Split and join
 */
${#strings.arrayJoin(namesArray,',')}
${#strings.listJoin(namesList,',')}
${#strings.setJoin(namesSet,',')}
${#strings.arraySplit(namesStr,',')}    //    returns    String[]
${#strings.listSplit(namesStr,',')}     //    returns    List<String>
${#strings.setSplit(namesStr,',')}      //    returns    Set<String>

/*
 *    Trim Also works with arrays, lists or sets
 */
${#strings.trim(str)}    //    also array*, list* and    set*

/*
 *    Compute length Also  works with arrays, lists or sets
 */
${#strings.length(str)}    //    also array*, list* and    set*

/*
 *    Abbreviate text    making it have a maximum size of n.    If text
is bigger, it
 *    will be clipped and finished in "..." Also works with arrays, lists
or sets
 */
${#strings.abbreviate(str,10)}    //    also array*, list* and    set*

/*
```

```
 *    Convert    the    first character    to upper-case (and vice-versa)
 */
${#strings.capitalize(str)}        //    also array*, list* and    set*
${#strings.unCapitalize(str)}      //    also array*, list* and    set*

/*
 *    Convert    the    first character    of every word to upper-case
 */
${#strings.capitalizeWords(str)}               //    also array*, list* and
   set*
${#strings.capitalizeWords(str,delimiters)}    //    also array*, list* and
   set*

/*
 *    Escape    the    string
 */
${#strings.escapeXml(str)}             //    also array*, list* and    set*
${#strings.escapeJava(str)}            //    also array*, list* and    set*
${#strings.escapeJavaScript(str)}      //    also array*, list* and    set*
${#strings.unescapeJava(str)}          //    also array*, list* and    set*
${#strings.unescapeJavaScript(str)}    //    also array*, list* and    set*

/*
 *    Null-safe comparison and concatenation
 */
${#strings.equals(first, second)}
${#strings.equalsIgnoreCase(first, second)}
${#strings.concat(values...)}
${#strings.concatReplaceNulls(nullValue, values...)}

/*
 *    Random
 */
${#strings.randomAlphanumeric(count)}
```

Objects

```
/*
 *    =========================================================
 *    See javadoc    API for    class org.thymeleaf.expression.Objects
 *    =========================================================
 */
/*
 *    Return obj if it is not null, and default otherwise Also works with
arrays, lists    or sets
 */
${#objects.nullSafe(obj,default)}
${#objects.arrayNullSafe(objArray,default)}
${#objects.listNullSafe(objList,default)}
${#objects.setNullSafe(objSet,default)}
```

Booleans

```
/*
```

```
 *    =======================================================
 *    See    javadoc    API    for class org.thymeleaf.expression.Bools
 *    =======================================================
 */
/*
 *    Evaluate a condition in the same way that it would be evaluated in a th:if tag
 *    (see conditional evaluation chapter afterwards).Also works with arrays, lists or sets
 */
${#bools.isTrue(obj)}
${#bools.arrayIsTrue(objArray)}
${#bools.listIsTrue(objList)}
${#bools.setIsTrue(objSet)}

/*
 *    Evaluate with negation Also works with arrays, lists or sets
 */
${#bools.isFalse(cond)}
${#bools.arrayIsFalse(condArray)}
${#bools.listIsFalse(condList)}
${#bools.setIsFalse(condSet)}

/*
 *    Evaluate and apply AND operator Receive an array, a list or a set as parameter
 */
${#bools.arrayAnd(condArray)}
${#bools.listAnd(condList)}
${#bools.setAnd(condSet)}

/*
 *    Evaluate and apply OR operator Receive an array, a list or a set as parameter
 */
${#bools.arrayOr(condArray)}
${#bools.listOr(condList)}
${#bools.setOr(condSet)}
```

Arrays

```
/*
 *    =======================================================
 *    See    javadoc API for class org.thymeleaf.expression.Arrays
 *    =======================================================
 */
/*
 *    Converts to array, trying to infer array component class.
 *    Note that if resulting array is    empty, or if the elements
 *    of the target object are not all of the same class,this method will return Object[].
 */
${#arrays.toArray(object)}

/*
```

```
 *    Convert to arrays of the specified component class.
 */
${#arrays.toStringArray(object)}
${#arrays.toIntegerArray(object)}
${#arrays.toLongArray(object)}
${#arrays.toDoubleArray(object)}
${#arrays.toFloatArray(object)}
${#arrays.toBooleanArray(object)}

/*
 *    Compute    length
 */
${#arrays.length(array)}

/*
 *    Check    whether    array    is    empty
 */
${#arrays.isEmpty(array)}


/*
 *    Check if element or elements are contained in array
 */
${#arrays.contains(array, element)}
${#arrays.containsAll(array, elements)}
```

Lists

```
/*
 *    =======================================================
 *    See    javadoc    API for class org.thymeleaf.expression.Lists
 *    =======================================================
 */
/*
 *    Converts to list
 */
${#lists.toList(object)}

/*
 *    Compute    size
 */
${#lists.size(list)}

/*
 *    Check whether list is empty
 */
${#lists.isEmpty(list)}

/*
 *    Check if element or elements are contained in list
 */
${#lists.contains(list, element)}
${#lists.containsAll(list, elements)}

/*
```

```
 *    Sort a copy of the given list.The members of the list must implement
 *    comparable or you must define a comparator.
 */
${#lists.sort(list)}
${#lists.sort(list, comparator)}
```

Sets

```
/*
 *    ========================================================
 *    See    javadoc    API    for    class    org.thymeleaf.expression.Sets
 *    ========================================================
 */
/*
 *    Converts to set
 */
${#sets.toSet(object)}

/*
 *    Compute    size
 */
${#sets.size(set)}

/*
 *    Check whether set is empty
 */
${#sets.isEmpty(set)}

/*
 *     Check if element or elements are contained in set
 */
${#sets.contains(set, element)}
${#sets.containsAll(set, elements)}
```

Maps

```
/*
 *    ========================================================
 *    See    javadoc    API    for    class    org.thymeleaf.expression.Maps
 *    ========================================================
 */
/*
 *    Compute    size
 */
${#maps.size(map)}

/*
 *    Check whether map is empty
 */
${#maps.isEmpty(map)}

/*
 *    Check if key/s or value/s are contained in maps
 */
${#maps.containsKey(map, key)}
```

```
${#maps.containsAllKeys(map, keys)}
${#maps.containsValue(map, value)}
${#maps.containsAllValues(map, value)}
```

聚合函数

```
/*
 *    =======================================================
 *    See javadoc API for class org.thymeleaf.expression.Aggregates
 *    =======================================================
 */
/*
 *    Compute sum. Returns null if array or collection is empty
 */
${#aggregates.sum(array)}
${#aggregates.sum(collection)}

/*
 *    Compute average. Returns null if array or collection is empty
 */
${#aggregates.avg(array)}
${#aggregates.avg(collection)}
```

IDs

> ids：处理可能重复的id属性的实用方法（例如，作为迭代的结果）。

```
/*
 *    =======================================================
 *    See javadoc API for class org.thymeleaf.expression.Ids
 *    =======================================================
 */
/*
 *    Normally used in th:id attributes, for appending a counter to the id attribute value
 *    so that it remains unique even when involved in an iteration process.
 */
${#ids.seq('someId')}

/*
 *    Normally used in th:for attributes in <label> tags, so that these labels can refer to Ids generated by means if the #ids.seq(...) function.
 *    Depending on whether the <label> goes before or after the element with the #ids.seq(...)
 *    function, the "next" (label goes before "seq") or the "prev" function (label goes after "seq") function should be called.
 */
${#ids.next('someId')}
${#ids.prev('someId')}
```

1.5.3.5 工具类案例

遍历指定次数

```
<ul>
    <!--
        遍历指定次数，这里需要用到 Tymyleaf 提供的工具类对象：${#numbers.sequence(from,to)}
    -->
    <li th:each="index:${#numbers.sequence(from,to)}" th:text="${index}"></li>
</ul>
```

日期格式化

```
<span th:text="${#dates.format(日期, 'yyyy-MM-dd hh:mm:ss')}"></span>
```

字符转换

```
<!-- 65 转换为 ASCII 值 -->
<span th:text="${#conversions.convert(65, 'java.lang.Character')}"></span>
```

1.5.4 FAQ 手册

1.5.5 作业实践

- 创建管理员表和学生信息表，完成如下功能。
  - 创建登录页面，完成管理员登录功能。
  - 创建展示页面，完成学生信息展示。
  - 创建添加页面，完成学生信息添加功能。
  - 创建修改页面，完成学生信息修改功能。
  - 创建删除按钮，完成学生信息删除功能。

1.5.6 面试宝典

1.5.7 拓展资料

1.6 SpringBoot 高级配置

1.6.1 课程目标

- 掌握 SpringBoot 配置静态资源
- 掌握 SpringBoot 配置定时任务
- 掌握 SpringBoot 配置事务
- 掌握 SpringBoot 配置拦截器
- 了解 SpringBoot 配置 JSP
- 掌握 SpringBoot 配置热部署

1.6.2 知识架构树

1.6.3 理论知识

1.6.3.1 SpringBoot 配置静态资源

1）"`/**`"

> 静态资源（HTML、CSS、JavaScript、图片、视频、音频等）映射规则，以下的目录都可以理解为根目录（"/"），按照以下目录的顺序挨个匹配，匹配到之后停止，后面就不在匹配。

```
SpringBoot默认提供的存放静态资源的路径：
    "classpath:/META-INF/resources/" 一般是引入三方依赖的时候静态资源的目录
    "classpath:/resources/"
    "classpath:/static/"
    "classpath:/public/"
    "/"
```

2）"webjars/"**

> 以 Maven 的形式导入前端资源
>
> 去classpath:/META-INF/resources/webjars/目录下找
>
> https://www.webjars.org/

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>jquery</artifactId>
    <version>3.4.1</version>
</dependency>
```

```
http://localhost:8080/webjars/jquery/3.4.1/jquery.js
```

3）欢迎页

> 可以将`index.html`页面放到以下任意一个路径即可。如果以下路径中有多个index.html找到一个就停止。

```
"classpath:/META-INF/resources/"
"classpath:/resources/"
"classpath:/static/"
"classpath:/public/"
```

4）更换启动图标

> 在classpath目录下创建`banner.txt`即可
>
> http://patorjk.com/software/taag http://www.network-science.de/ascii/

5）更换浏览器图标

> 找一个后缀名为【.ico】的文件，将文件重命名`favicon.ico`，然后放到资源文件夹（resource/static/public）下，重启服务器即可。

1.6.3.2 SpringBoot 配置定时任务

> 希望某些需求在指定的时间点去执行。备份数据、发送邮件等。

时间表达式：

```
语法结构：
    秒    分    时    日    月    星期   年(可以为空)

    第一位：表示秒，取值范围： 0-59
    第二位：表示分，取值范围： 0-59
    第三位：表示小时，24小时制，取值 0-23
    第四位：日期天/日，取值范围：1-31
    第五位：日期月份，取值范围：1-12
    第六位：星期，取值范围：1-7，表示星期一，星期二...
        注意：1表示星期天，2表示星期一。
    第七位：年份，可以留空

表达式：
    *：每；如果写在秒上，表示每秒；写在分钟表示每分钟；
    /：步长；例如，1/5，如果在秒上，表示，从第一秒开始，每5秒执行一次；
    ?：只能用在每月第几天和星期两个域。表示不指定值，当 2 个子表达式其中之一被指定了值以后，为了避免冲突，需要将另一个子表达式的值设为“?”
    -：表示范围，例如，在分域使用5-20，表示从5分到20分钟每分钟触发一次
    ,：枚举
    L：last，表示最后，只能出现在星期和每月第几天域，如果在星期域使用1L,意味着在最后的一个星期日触发。
    W：表示有效工作日(周一到周五),只能出现在每月第几日域，系统将在离指定日期的最近的有效工作日触发事件。注意一点，W的最近寻找不会跨过月份
    LW：这两个字符可以连用，表示在某个月最后一个工作日，即最后一个星期五
    #：用于确定每个月第几个星期几，只能出现在每月第几天域。例如在1#3，表示某月的第三个星期日
--------------------------------------------------------------------------------
eg：
* * * * * * -> 每秒执行
0 0 2 * * ? -> 每天凌晨两点执行

生成表达式：http://qqe2.com/cron
```

1）编写一个定时任务类

```
@Component // 把这个类交给spirng管理
public class MyQuartz {

    @Scheduled(cron="1,5,10 * * * * ?")// 定时任务的表达式
    public void doIt(){
        System.out.println(new Date());
    }
}
```

2）启动类上开启定时任务

```
@EnableScheduling // 开启定时任务
```

1.6.3.3 SpringBoot 配置事务

- 配置

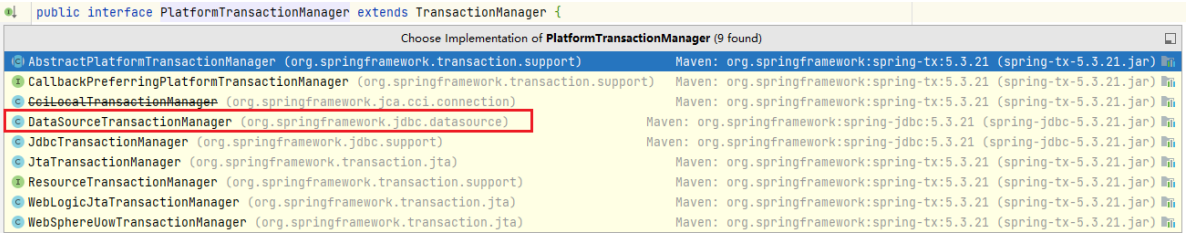
SpringBoot 在启动时已经加载了事务管理器，所以只需要在需要添加事务的方法/类上添加 `@Transactional` 即可生效，无需额外配置。

- `TransactionAutoConfiguration` 配置类解析

> SpringBoot 启动时加载 `/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` 或者 `spring.factories` 文件中定义的所有自动配置类。其中包含了 `org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration` 事务自动配置类。

```
@AutoConfiguration(after = { JtaAutoConfiguration.class, HibernateJpaAutoConfiguration.class,
        DataSourceTransactionManagerAutoConfiguration.class, Neo4jDataAutoConfiguration.class })
@ConditionalOnClass(PlatformTransactionManager.class)
@EnableConfigurationProperties(TransactionProperties.class)
public class TransactionAutoConfiguration {
```

> 配置类中使用注解 `@ConditionalOnClass` 验证了 `PlatformTransactionManager.class` 接口的实例。接口的实现类如下：



> 接口的实现类中包含了 `DataSourceTransactionManager` 类，这个类在 SpringBoot 启动时通过 `org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration` 配置类进行了导入，如下图。所以容器中就包含了 `PlatformTransactionManager.class` 类型的实例，所以 `TransactionAutoConfiguration` 配置类正常加载。

```
@AutoConfiguration
@ConditionalOnClass({ JdbcTemplate.class, TransactionManager.class })
@AutoConfigureOrder(Ordered.LOWEST_PRECEDENCE)
@EnableConfigurationProperties(DataSourceProperties.class)
public class DataSourceTransactionManagerAutoConfiguration {

    @Configuration(proxyBeanMethods = false)
    @ConditionalOnSingleCandidate(DataSource.class)
    static class JdbcTransactionManagerConfiguration {

        @Bean
        @ConditionalOnMissingBean(TransactionManager.class)
        DataSourceTransactionManager transactionManager(Environment environment, DataSource dataSource,
                ObjectProvider<TransactionManagerCustomizers> transactionManagerCustomizers) {
            DataSourceTransactionManager transactionManager = createTransactionManager(environment, dataSource);
            transactionManagerCustomizers.ifAvailable((customizers) -> customizers.customize(transactionManager));
            return transactionManager;
        }

        private DataSourceTransactionManager createTransactionManager(Environment environment, DataSource dataSource) {
            return environment.getProperty( key: "spring.dao.exceptiontranslation.enabled", Boolean.class, Boolean.TRUE)
                    ? new JdbcTransactionManager(dataSource) : new DataSourceTransactionManager(dataSource);
        }

    }

}
```

> `TransactionAutoConfiguration` 配置类中编写了一个内部类`public static class EnableTransactionManagementConfiguration`，其中包含了`@EnableTransactionManagement`注解，所以这也就是为什么启动类中不添加`@EnableTransactionManagement`注解事务也能生效的原因。

1.6.3.4 SpringBoot 配置拦截器

1）创建拦截器类，实现HandlerInterceptorAdapter，重写方法（根据自己业务需求）

```
@Component
public class MyInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("拦截器执行");
    }
}
```

2）编写一个配置类，实现WebMvcConfigurer，配置自定义拦截器，添加到组件中

```
@Configuration // 告诉spring我是一个配置类
public class MvcConfig implements WebMvcConfigurer {

    @Resource
    MyInterceptor myInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
```

```
        // 将自定义的拦截器添加到spring中
        // addPathPatterns(): 拦截请求
        // excludePathPatterns(): 黑名单
        registry.addInterceptor(myInterceptor).addPathPatterns("/**/*");
    }
}

// 在老版本中要继承WebMvcConfigurerAdapter，重写addInterceptors()方法
```

1.6.3.5 SpringBoot 配置支持 JSP

> 随着技术发展，JSP的局限性越发突出，为了迎合前后端分离的发展，SpringBoot默认不支持JSP。但是一些需求下还可能会使用到JSP。

1）添加 Tomcat 的场景启动器

```
<!-- Tomcat启动器 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

2）在启动类中覆盖原有的内置的 Tomcat，需要继承 SpringBootServletInitializer，重写configure方法

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

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
builder) {
        return builder.sources(Application.class);
    }
}
```

3）添加本地的 Tomcat 运行项目。

1.6.3.6 SpringBoot 热部署

1）导入开发者工具包

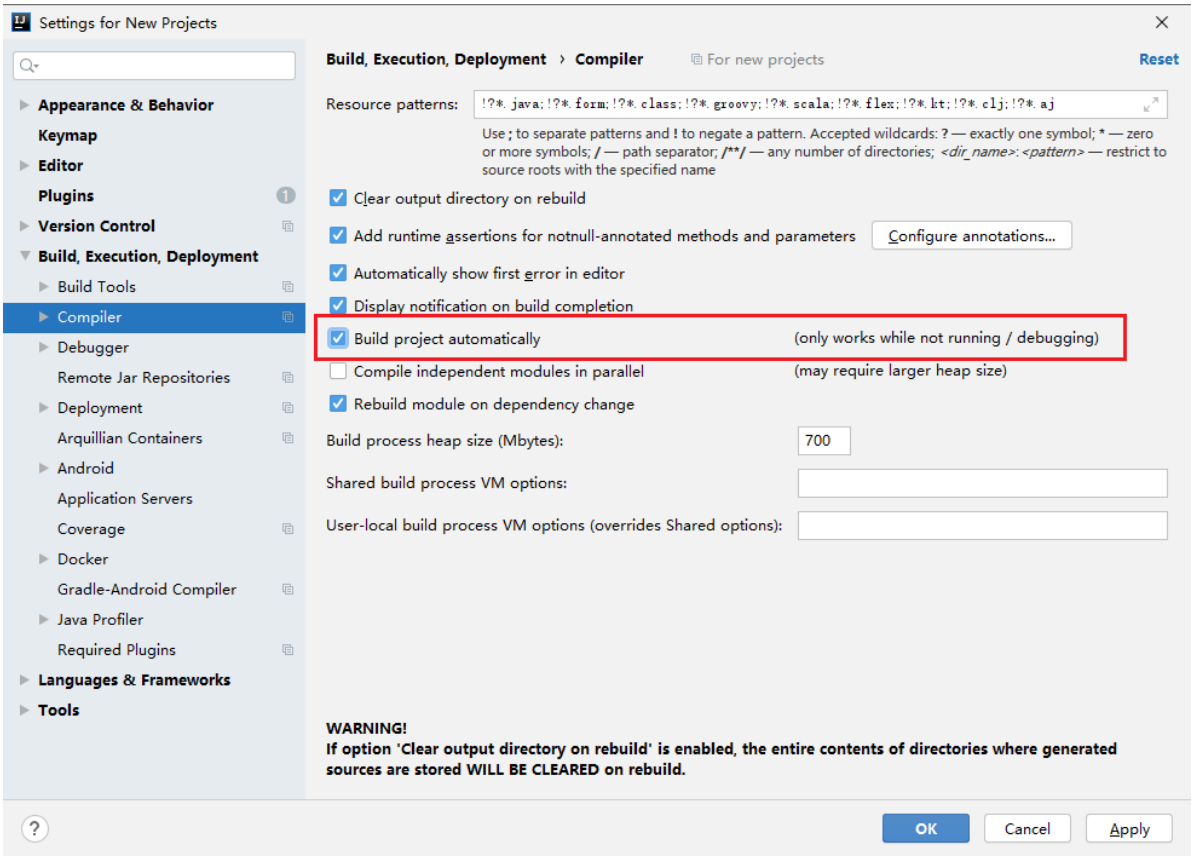
```
<!--devtools热部署-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
    <scope>true</scope>
</dependency>
```

2）修改配置文件

```yaml
spring:
  devtools:
    restart:
      enabled: true  #设置开启热部署
      additional-paths: src/main/java #重启目录
      exclude: WEB-INF/**
  freemarker:
    cache: false    #页面不加载缓存，修改即时生效
```

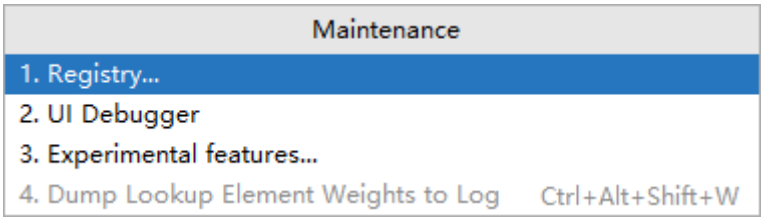
3）修改idea配置

3.1） 配置自动编译

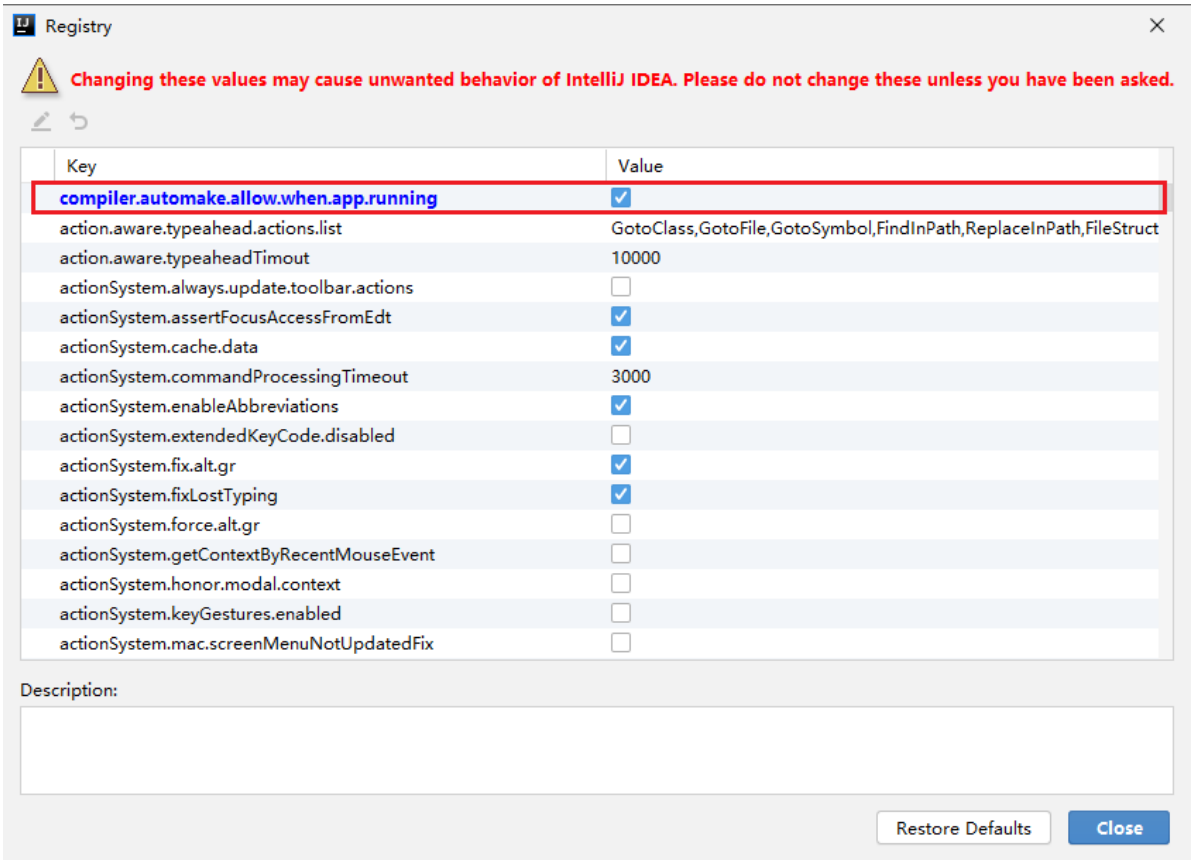


3.2） 配置自动编译

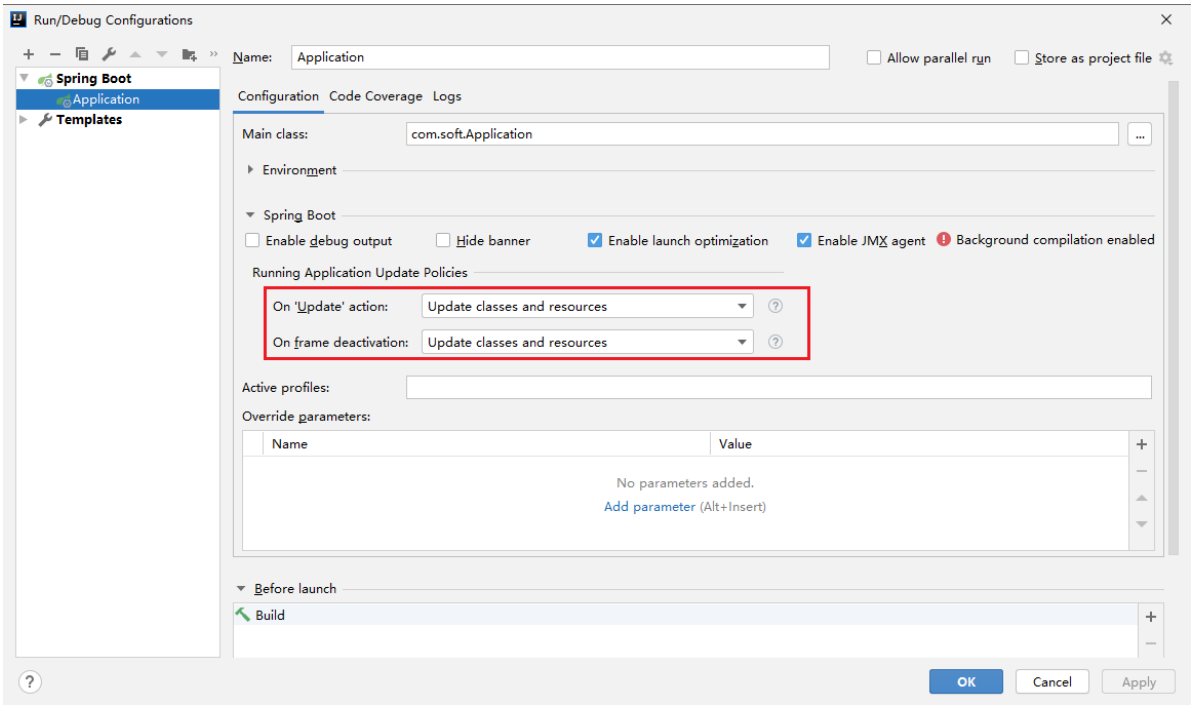
使用快捷键 `ctrl + shift + alt + /` ，选择 `Registry...`



在选项中找到 `compiler.automake.allow.when.app.running` 选项并打勾，应用退出即可。



4）启动配置



1.6.4 FAQ 手册

1.6.5 作业实践

1. 在工程中静态资源目录添加图片，创建模板，加载图片。
2. 使用定时任务，实现按小时报时功能。
3. SpringBoot 事务完成转账业务。

1.6.6 面试宝典

1.6.7 拓展资料

1.7 SpringBoot 自定义启动器

1.7.1 课程目标

- 掌握并配置 SpringBoot 启动器

1.7.2 知识架构树

- 启动器概念
- 创建自定义启动器

1.7.3 理论知识

1.7.3.1 Starter简介

启动器简介

> SpringBoot 中的 Starter 是一种非常重要的机制，能够抛弃以前繁杂的配置，将其统一集成进 Starter，应用者只需要在 Maven 中引入 Starter 依赖，SpringBoot 就能自动扫描到要加载的信息并启动相应的**默认配置**。
>
> Starter 让我们摆脱了各种依赖库的处理，需要配置各种信息的困扰。SpringBoot 会自动通过 **classpath** 路径下的类发现需要的Bean，并注册进 IOC 容器。SpringBoot 提供了针对日常企业应用研发各种场景的 spring-boot-starter 依赖模块。所有这些依赖模块都遵循着**约定成俗**的默认配置，并允许我们调整这些配置，即遵循**"约定大于配置"**的理念。
>
> 在我们的日常开发工作中，经常会有一些独立于业务之外的配置模块，我们经常将其放到一个特定的包下，然后如果另一个工程需要复用这块功能的时候，需要将代码硬拷贝到另一个工程，重新集成一遍，麻烦至极。如果我们将这些可独立于业务代码之外的功配置模块封装成一个个 Starter，复用的时候只需要将其在 pom 中引用依赖即可，SpringBoot 为我们完成自动装配。

命名要求

> SpringBoot 官方提供的 starter 以 `spring-boot-starter-xxx` 的方式命名的。
>
> 官方建议自定义的 starter 使用 `xxx-spring-boot-starter` 命名规则。以区分 SpringBoot 生态提供的 starter。

1.7.3.2 创建自定义启动器

1.7.3.2.1 创建自定义Starter

1）新建 Maven 工程，按照官方命名要求命名，例如：`demo-spring-boot-starter`

2）导入自动配置依赖

> 导入`spring-boot-autoconfigure`依赖之后，当前项目中就拥有了基本的 Spring 环境，可以做对象的基础管理。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <!-- 当前启动器的坐标地址，其他项目可以通过该坐标引入该启动器 -->
    <groupId>com.riu</groupId>
    <artifactId>custom-spring-boot-starter</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- 导入自动配置依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
            <version>2.7.1</version>
        </dependency>
    </dependencies>
</project>
```

3）编写功能代码

> 这里使用工具类为例，工具类中定义了全局属性和公共方法。

```
package com.riu.util;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.context.properties.ConfigurationProperties;

/**
 * 工具类
 * @author riu
 */
// @Component 自定义的工具类可能和主工程的包路径不一致，扫描不到
// 开启自动配置，使用 @ConfigurationProperties 时必须添加该注解
@EnableAutoConfiguration
// 配置文件赋值前缀配置
@ConfigurationProperties(prefix = "coustom")
public class CustomUtil {

    // 定义全局属性并初始化
    private String name = "default";

    /**
     * 实现打印一句话需求
     */
    public void sayHi(){
        System.out.println(name);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
}

```

4）编写配置类

> 因为 SpringBoot 不推荐使用 xml 配置文件形式向容器注入对象，所以这里需要使用配置类向 Spring 容器中注入对象。

```
// 该注解标识当前类是一个配置类，配置类被加载时，@Bean 修饰的方法的返回值会自动注入到 Spring 容器中
@Configuration
public class CustomUtilAutoConfiguration  {

    // 将方法的返回值注入到 Spring 容器中
    @Bean
    public CustomUtil customUtil(){
        CustomUtil customUtil = new CustomUtil();
        customUtil.setName("defaultInConfiguration");

        return customUtil;
    }
}
```

5）创建

- 在resources 目录下创建 `/META-INF/` 目录，并创建 `spring.factories` 文件

```
# SpringBoot 启动时会读取所有依赖下 /META-INF/spring.factories 文件，并根据 org.springframework.boot.autoconfigure.EnableAutoConfiguration 关键字获取对应配置类的全限定名称，依此来自动注入对象
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.riu.autoconfigure.CustomAutoConfiguration
```

- 在resources 目录下创建 `/META-INF/spring` 目录，并创建 `org.springframework.boot.autoconfigure.AutoConfiguration.imports` 文件

```
com.riu.autoconfigure.CustomUtilAutoConfiguration
```

1.7.3.2.2 测试自定义启动器

1）在 SpringBoot 项目中添加自定义启动器依赖

```
<dependency>
    <groupId>com.riu</groupId>
    <artifactId>custom-spring-boot-starter</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

2）注入自定义启动器暴露的对外对象

```java
@Resource // 注入自定义启动器中的工具类
CustomUtil customUtil;

/**
 * 编写方法调用工具类方法
 */
public String customUtil(){
    customUtil.sayHi();
}
```

3）通过配置文件给属性赋值

```yaml
custom:
  name: Tom
```

- **自定义启动器启动注解**

> 通过以上配置即可完成自定义 Starter 的使用。在 SpringBoot 启动时自动注入对象，可直接使用。但，并不是所有的 Starter 都需要在 SpringBoot 启动后能直接使用，例如：定时任务、事务等启动器，在需要时开启。

1）创建标记类

> 标记类是决定自定义启动时的自动配置是否生效的关键，类似比赛的发令枪，当运动员听到发令枪时开始比赛。

```java
public class CustomEnableSelector{
}
```

2） 创建开关注解

> 开关注解是写在 SpringBoot 启动类上，用于开启自定义启动器的自动配置，类似比赛的裁判。开关注解添加之后，导入标记类，也就是裁判手里的发令枪。

```java
/**
 * 注解的作用就是一个开关，当添加了这个注解，自定义启动器可以生效。不添加就不生效
 */
@Target({ElementType.TYPE}) // 注解添加的位置。ElementType.TYPE 在类上
@Retention(RetentionPolicy.RUNTIME) // 注解编译的时期。
@Import({CustomEnableSelector.class}) // 在添加注解之后导入 CustomEnableSelector 类的实例
public @interface EnableCustom {
}

```

3）改造自动注入配置类

> 条件注解，Spring 容器中有 `CustomEnableSelector` 实例时(标记类) 才会启用 Starter 自动配置。`@ConditionalOnBean(CustomEnableSelector.class)` 注解用于验证容器中是否包含标记类，类似于运动员是否听到了发令枪的枪声。

```java
// 该注解标识当前类是一个配置类，配置类被加载时，@Bean 修饰的方法的返回值会自动注入到 Spring 容器中
@Configuration
```

```
// 条件注解，Spring 容器中有 DiyConfigurationSelector 实例时(标记类) 才会启用 Starter
@ConditionalOnBean(CustomConfigurationSelector.class)
public class CustomUtilAutoConfiguration  {

    // 将方法的返回值注入到 Spring 容器中
    @Bean
    public CustomUtil customUtil(){
        CustomUtil customUtil = new CustomUtil();
        customUtil.setName("defaultInConfiguration");

        return customUtil;
    }
}
```

4）SpingBoot 启动类添加开关注解

```
@SpringBootApplication
@EnableCustom // 自定义启动器开关注解
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

1.7.4 FAQ 手册

1.7.5 作业实践

1.7.6 面试宝典

1.7.7 拓展资料