

* 学习目标

- * 能够掌握SpringBoot端口号和上下文路径配置
- * 能够掌握SpringBoot工程热部署
- * 能够掌握获取配置内容和日志的配置
- * 能够掌握SpringBoot添加Druid连接池
- * 能够在SpringBoot使用JSP
- * 能够掌握SpringBoot文件上传
- * 能够掌握在SpringBoot中使用过滤器
- * 能够使用SpringBoot集成JUnit
- * 能够掌握YAML文件的配置
- * 能够掌握Thymeleaf的概述和HelloWorld的开发
- * 能够掌握Thymeleaf的语法

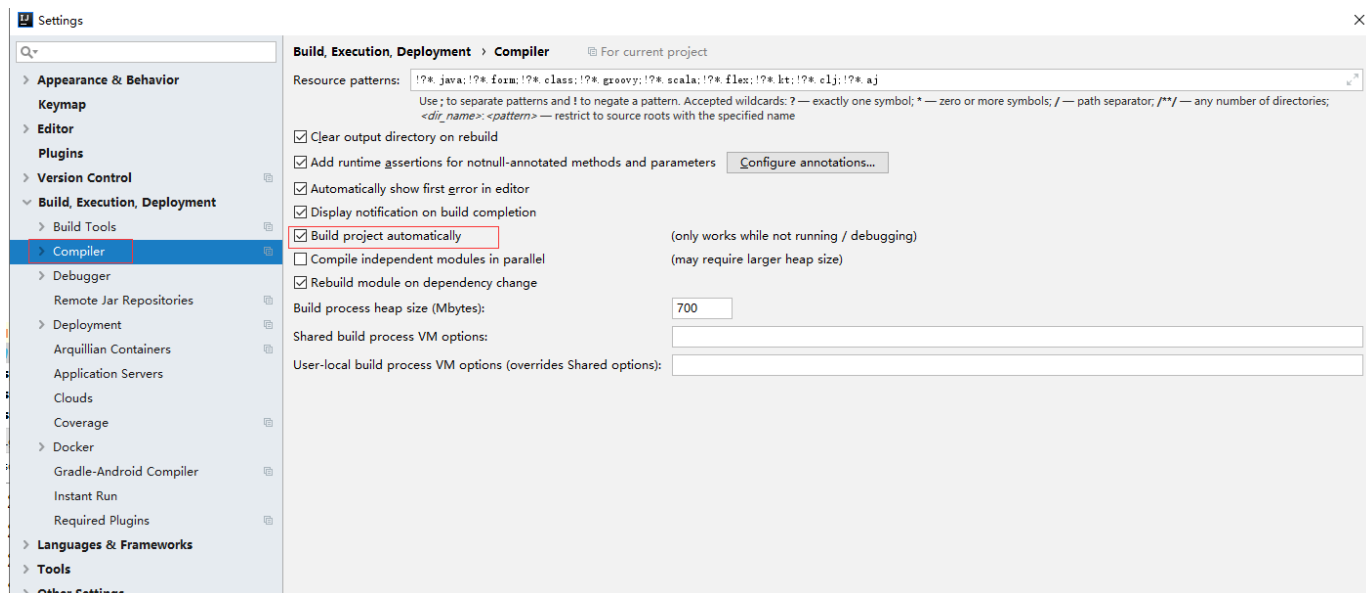
* 回顾

- * 能够掌握SpringBoot端口号和上下文路径配置

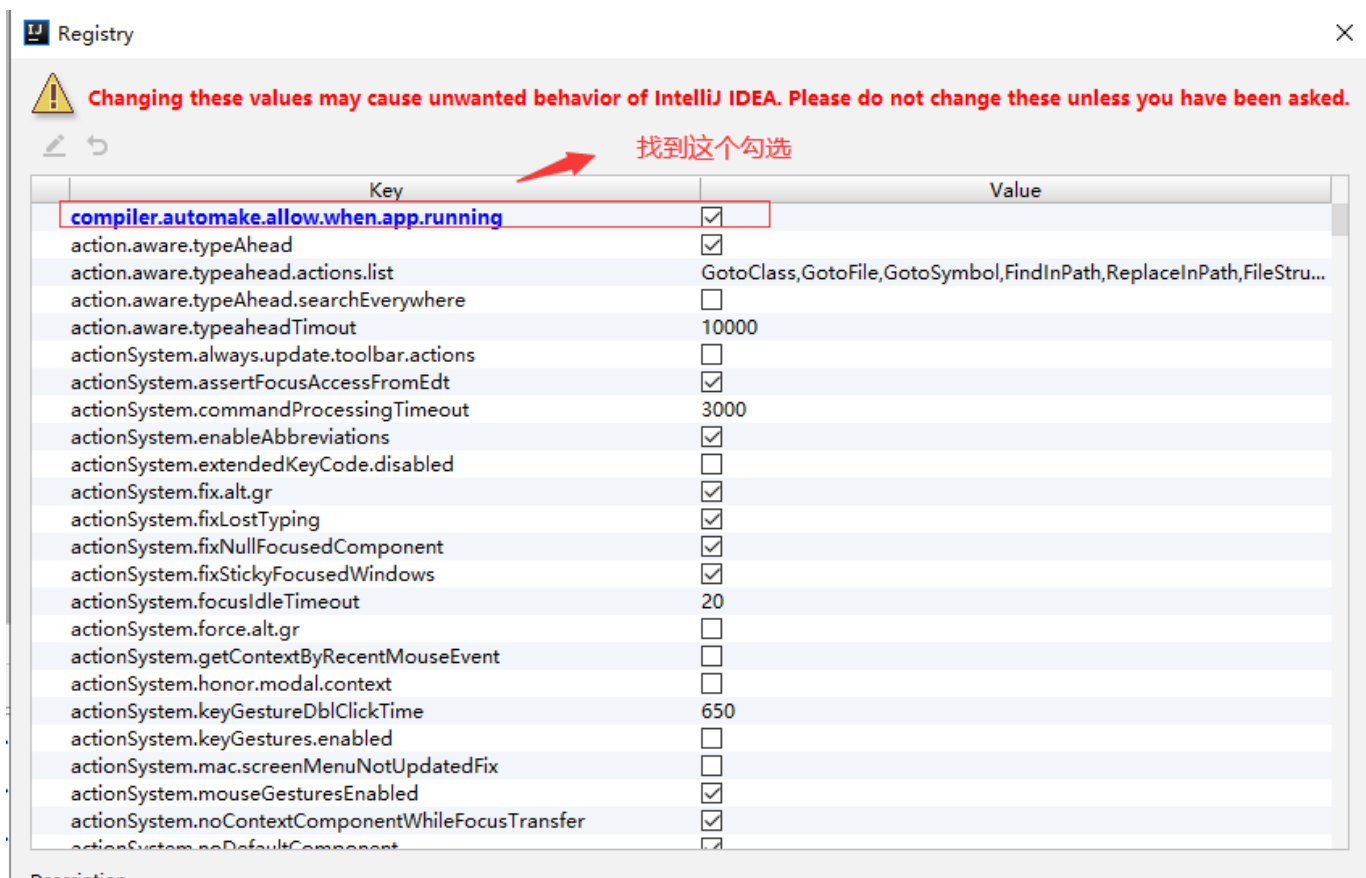
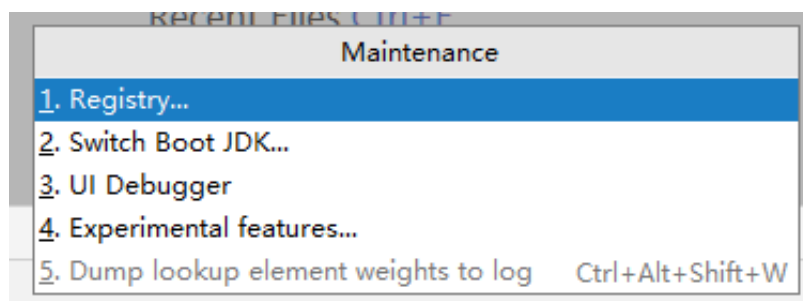
```
1 server.port=8082
2 server.servlet.context-path=/lg
```

- * 能够掌握SpringBoot工程热部署

```
1 * 添加依赖
2 <!-- 热部署配置 -->
3 <dependency>
4     <groupId>org.springframework.boot</groupId>
5     <artifactId>spring-boot-devtools</artifactId>
6 </dependency>
```



* ctrl+shift+alt+ /



* 能够掌握获取配置内容和日志的配置

```
1 * 案例一：（获取配置内容）
2 * 配置
3 * 在application.properties
4 lg.content=test springboot
5 * 获取
6 @Value("${lg.content}")
7 private String content;
8
9 * 案例二：（配置日志）
10 * 配置
11 logging.file.path=D:/wook666/log
12 logging.level.com.lg.dao=debug
13 #logging.pattern.console=%d{yyyy-MM-dd} [%thread] %-5level %logger{50} -%msg%n
14 #logging.level.root=info # 默认选项
15 * 使用
16 * 使用Lombok的@Slf4j注解
17 * 核心代码
18 @Slf4j
19 public class HelloController
20     log.info("xiaohei123");
```

* 能够掌握SpringBoot添加Druid连接池

```
1 * 添加数据库连接池druid
2 * 依赖
3 <dependency>
4     <groupId>com.alibaba</groupId>
5     <artifactId>druid-spring-boot-starter</artifactId>
6     <version>1.1.10</version>
7 </dependency>
8 * 配置
9 spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
10 spring.datasource.driver=com.mysql.jdbc.Driver
11 spring.datasource.url=jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
12 spring.datasource.username=root
13 spring.datasource.password=root
```

* 能够在SpringBoot使用JSP

```
1 * 添加依赖
2 <!-- 添加servlet依赖模块 -->
3 <dependency>
4     <groupId>javax.servlet</groupId>
5     <artifactId>javax.servlet-api</artifactId>
6     <scope>provided</scope>
7 </dependency>
8 <!-- 添加jstl标签库依赖模块 -->
9 <dependency>
10     <groupId>javax.servlet</groupId>
11     <artifactId>jstl</artifactId>
12 </dependency>
13 <!-- 使用jsp引擎，springboot内置tomcat没有此依赖 -->
14 <dependency>
15     <groupId>org.apache.tomcat.embed</groupId>
16     <artifactId>tomcat-embed-jasper</artifactId>
17     <scope>provided</scope>
18 </dependency>
19
20 * 配置
21 # 支持jsp
22 spring.mvc.view.prefix=/WEB-INF/jsp/
23 spring.mvc.view.suffix=.jsp
24 #spring.thymeleaf.cache=false
25 #spring.thymeleaf.check-template=false
26
27 * 构建webapp目录
28 * 配置成webapp根目录
29 * 打包
30 * pom.xml 配置
31 <build>
32     <plugins>
33         <plugin>
34             <groupId>org.springframework.boot</groupId>
35             <artifactId>spring-boot-maven-plugin</artifactId>
36         </plugin>
```

```

37     </plugins>
38     <resources>
39         <resource>
40             <!-- 打包时将jsp文件拷贝到META-INF目录下-->
41             <directory>src/main/webapp</directory>
42             <targetPath>META-INF/resources</targetPath>
43             <includes>
44                 <include>**/*</include>
45             </includes>
46         </resource>
47         <resource>
48             <directory>src/main/resources</directory>
49             <includes>
50                 <include>**/*</include>
51             </includes>
52             <filtering>false</filtering>
53         </resource>
54         <resource>
55             <directory>src/main/java</directory>
56             <excludes>
57                 <exclude>**/*.java</exclude>
58             </excludes>
59         </resource>
60     </resources>
61 </build>

```

* 能够掌握SpringBoot文件上传

```

1 # 文件上传配置
2 # 单个文件的最大值
3 spring.servlet.multipart.max-file-size=50MB
4 # 上传文件总的最大值
5 #spring.servlet.multipart.max-request-size=100MB

```

* 能够掌握在SpringBoot中使用过滤器

```

1 @Configuration

```

```

2 public class WebConfig {
3     @Bean
4     public FilterRegistrationBean encodingFilter(){
5         FilterRegistrationBean registrationBean=new FilterRegistrationBean();
6         CharacterEncodingFilter encodingFilter=new CharacterEncodingFilter();
7         encodingFilter.setEncoding("UTF-8");
8         registrationBean.addUrlPatterns("/*");
9         registrationBean.setFilter(encodingFilter);
10        return registrationBean;
11    }
12
13    @Bean
14    public FilterRegistrationBean hiddenFilter(){
15        FilterRegistrationBean registrationBean=new FilterRegistrationBean();
16        HiddenHttpMethodFilter hiddenHttpMethodFilter=new HiddenHttpMethodFilter();
17        registrationBean.setFilter(hiddenHttpMethodFilter);
18        registrationBean.addUrlPatterns("/*");
19        return registrationBean;
20    }
21 }
22
23 * 在SpringBoot可以添加配置替换上面的过滤器
24 * HiddenHttpMethodFilter
25 spring.mvc.hiddenmethod.filter.enabled=true
26 * 全局编码配置
27 spring.http.encoding.charset=UTF-8
28 spring.http.encoding.force=true
29 spring.http.encoding.enabled=true
30 # Character encoding to use to decode the URI.
31 server.tomcat.uri-encoding=UTF-8

```

* 能够使用SpringBoot集成JUnit

```

1 * 添加依赖
2 * IOC 测试
3 @RunWith(SpringRunner.class)
4 @SpringBootTest(classes = LgApplication.class)
5 public class AppTest {
6     @Autowired

```

```

7     private CompanyService companyService;
8     @Test
9     public void test1() {
10         Company company = companyService.getCompany(1001);
11         System.out.println(company);
12     }
13 }
14 * 温馨提醒:
15 * SpringRunner 继承 SpringJUnit4ClassRunner, 然后啥事也不干
16 * 估计是觉得SpringJUnit4ClassRunner名字太长了, 换个名字
17 * Mock 测试
18 * 方式一:
19 @RunWith(SpringRunner.class)
20 @SpringBootTest(classes = LgApplication.class)
21 public class AppTest2 {
22     @Autowired
23     private WebApplicationContext applicationContext;
24     private MockMvc mockMvc;
25     @Before
26     public void before() {
27         this.mockMvc = MockMvcBuilders.webAppContextSetup(applicationContext).b
28     }
29     @Test
30     public void test18() throws Exception {
31         String result = mockMvc.perform(delete("/emp/sexCount")).
32             andExpect(status().isOk()).
33             andDo(print()).andReturn().getResponse().getContentAsString();
34         System.out.println(result);
35     }
36 }
37
38 * 方式二:
39 @RunWith(SpringRunner.class)
40 @SpringBootTest(classes = LgApplication.class)
41 @AutoConfigureMockMvc
42 public class AppTest2 {
43     @Autowired
44     private MockMvc mockMvc;
45     @Test
46     public void test18() throws Exception {

```

```

47     String result = mockMvc.perform(delete("/emp/sexCount")).
48         andExpect(status().isOk()).
49         andDo(print()).andReturn().getResponse().getContentAsString();
50     System.out.println(result);
51 }
52 }
53 * 配置（不配置会报错）
54 spring.datasource.druid.web-stat-filter.enabled=false

```

* 能够掌握YAML文件的配置



```

1 * 案例一：把properties替换成yaml文件
2 # banner
3 spring:
4   banner:
5     image:
6       location: classpath:banner.jpg
7   datasource:
8     type: com.alibaba.druid.pool.DruidDataSource
9     driver: com.mysql.jdbc.Driver
10    url: jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
11    username: root
12    password: root
13    druid:
14      web-stat-filter:

```



```
15         enabled: false
16     mvc:
17         view:
18             prefix: /WEB-INF/jsp/
19             suffix: .jsp
20             hiddenmethod:
21                 filter:
22                     enable: true
23     thymeleaf:
24         cache: false
25         check-template: false
26     servlet:
27         multipart:
28             max-request-size: 50MB
29             max-file-size: 100MB
30     http:
31         encoding:
32             charset: UTF-8
33             force: true
34             enabled: true
35
36 # server
37 server:
38     port: 8088
39     servlet:
40         context-path: /lg
41
42 # mybatis
43 mybatis:
44     mapper-locations: com/lg/dao/*.xml */
45     type-aliases-package: com.lg.bean
46 # log
47 logging:
48     file:
49         path: D:/wook666/log
50     level:
51         com.lg.dao: debug
52 # 自定义
53 lg:
54     content: xiaohei
```

```
55
56 * 案例二：自定数据和获取
57 * 添加依赖
58 <dependency>
59   <groupId>org.springframework.boot</groupId>
60   <artifactId>spring-boot-configuration-processor</artifactId>
61   <optional>true</optional>
62 </dependency>
63 # 自定义
64 lg:
65   content: xiaohei
66   addresses:
67     - provinceName: 广东
68       cityName: 广州12
69     - provinceName: 广东
70       cityName: 佛山12
71 @ConfigurationProperties(prefix = "lg")
72 public class HelloController {
73   @Value("${lg.content}")
74   private String content;
75   private List<Address> addresses;
76   public void setAddresses(List<Address> addresses) {
77     this.addresses = addresses;
78   }
79 }
80
81 * @Value解析对象数组的时候回报错，使用ConfigurationProperties解析
```

* 能够掌握Thymeleaf的概述和HelloWorld的开发



* 能够掌握Thymeleaf的语法

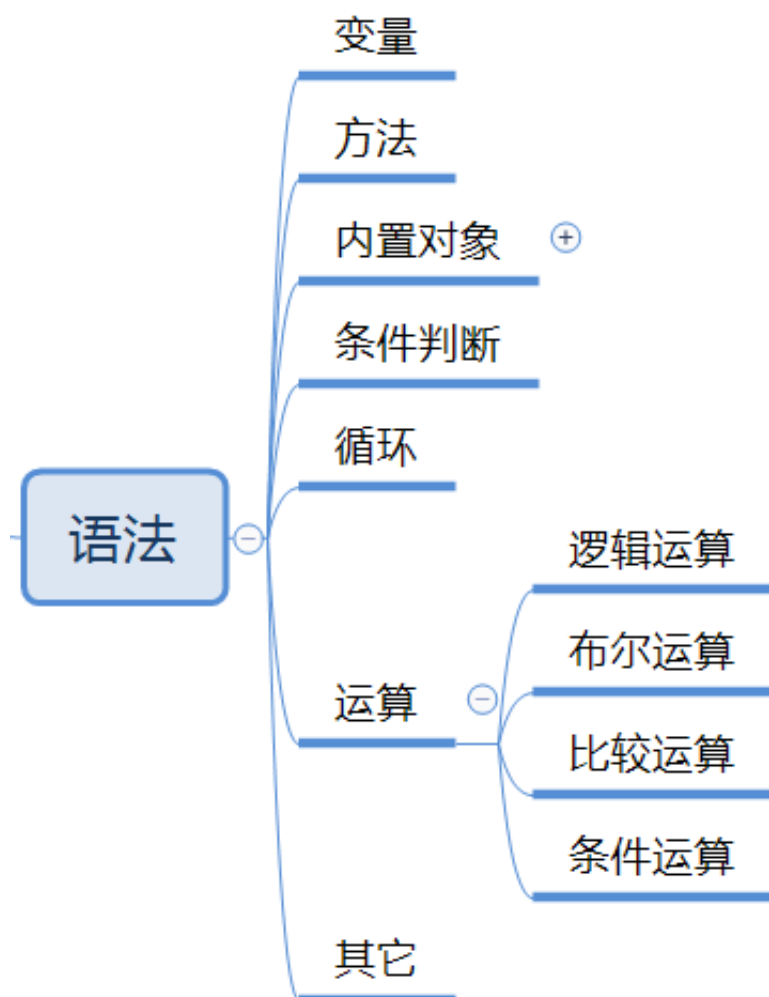
```
1 * 添加依赖
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-thymeleaf</artifactId>
5 </dependency>
6 * 在resources目录创建templates目录
7 * 新建test1.html
8 <!DOCTYPE html>
9 <html lang="en" xmlns:th="http://www.thymeleaf.org">
10 <head>
11   <meta charset="UTF-8">
12   <title>hello thymeleaf</title>
13 </head>
14 <body>
15   <h1>你好, 亮哥教育</h1>
16   <h1 th:text="${msg}">你好</h1>
17 </body>
18 </html>
19
20 * Controller
21 @Controller
22 @RequestMapping("/thymeleaf")
23 public class ThymeleafController {
24   @RequestMapping("/test1")
25   public String test1(Model model){
```

```

26     model.addAttribute("msg","hello thymeleaf");
27     return "test1";
28 }
29 }
30
31 * 测试
32 * http://localhost:8088/lg/thymeleaf/test1
33 * 温馨提醒:
34 * Thymeleaf中所有的表达式都需要写在`指令`中, 指令是HTML5中的自定义属性,
35   在Thymeleaf中所有指令都是以`th:`开头, 直接用浏览器打, 会自动忽略这些指令, 这样就不
36 * 注意: 加入了thymeleaf之后, 之前jsp访问就报错

```

* 能够掌握Thymeleaf的语法



```

1 * 案例一: (变量)
2 <!DOCTYPE html>

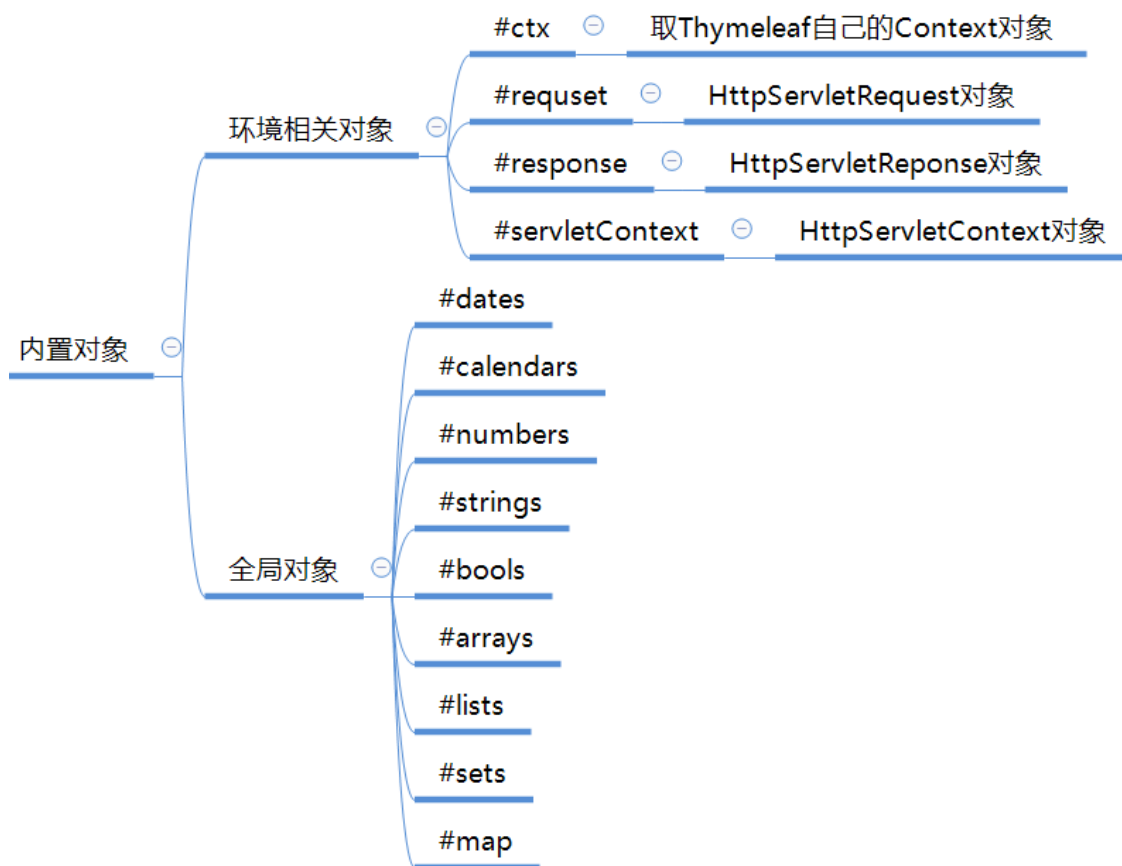
```

```

3 <html lang="en" xmlns:th="http://www.thymeleaf.org">
4 <head>
5     <meta charset="UTF-8">
6     <title>hello thymeleaf</title>
7 </head>
8 <body>
9     <h1>你好，亮哥教育</h1>
10    <p> name:<span th:text="${user.username}"></span><p>
11    <p> sex:<span th:text="${user.sex}"></span> </p>
12    <h1 th:object="${user}">
13        <p> name:<span th:text="*{username}"></span><p>
14        <p> sex:<span th:text="*{sex}"></span> </p>
15    </h1>
16 </body>
17 </html>
18 @RequestMapping("/test2")
19 public String test2(Model model){
20     User user=new User();
21     user.setUsername("xiaohei");
22     user.setSex("男");
23     model.addAttribute("user",user);
24     return "test2";
25 }
26
27 * 案例二：（方法）
28 <h1 th:object="${user}">
29     <p> name:<span th:text="*{username.startsWith('xiaohei')}"></span><p>
30     <p> sex:<span th:text="*{sex}"></span> </p>
31 </h1>

```

* 内置对象



```

1 * 案例三：（内置对象）
2 model.addAttribute("date", new Date());
3 <p>日期: <span th:text="${#dates.format(date, 'yyyy/MM/dd')}"></span></p>
4 * 案例四：（运算）
5 <p>运算: <span th:text="${12*2}"></span></p>
6 <p>运算: <span th:text="${12>2}"></span></p>
7 <p>运算: <span th:text="${user.sex}? '1':'2'"></span></p>
8 * 案例五：（条件判断）
9 <p><span th:if="${user.sex}=='男'">您是男的</span></p>
10 <div th:switch="${role}">
11   <p><span th:case="'admin'">管理员</span></p>
12   <p><span th:case="'user'">用户</span></p>
13 </div>
14 model.addAttribute("role", "admin");
15 * 案例六：（循环）
16 <div>
17   <ul>
18     <li th:each="user:${users}">
19       <span th:text="${user.username}"></span>-
  
```

```
20     <span th:text="${user.sex}"></span>
21   </li>
22 </ul>
23 </div>
```

* JS模板

模板引擎不仅可以渲染html，也可以对JS中的进行预处理。而且为了在纯静态环境下可以运行，其Thymeleaf代码可以被注释起来

其它

JS模板

语法结构

`const user = /*[[Thymeleaf表达式]]*/ "静态环境下的默认值"`

```
1 * 案例
2 <script th:inline="javascript">
3     const sex = /*[[${user.sex}]]*/ '女';
4     console.log(sex)
5 </script>
```