* 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

---

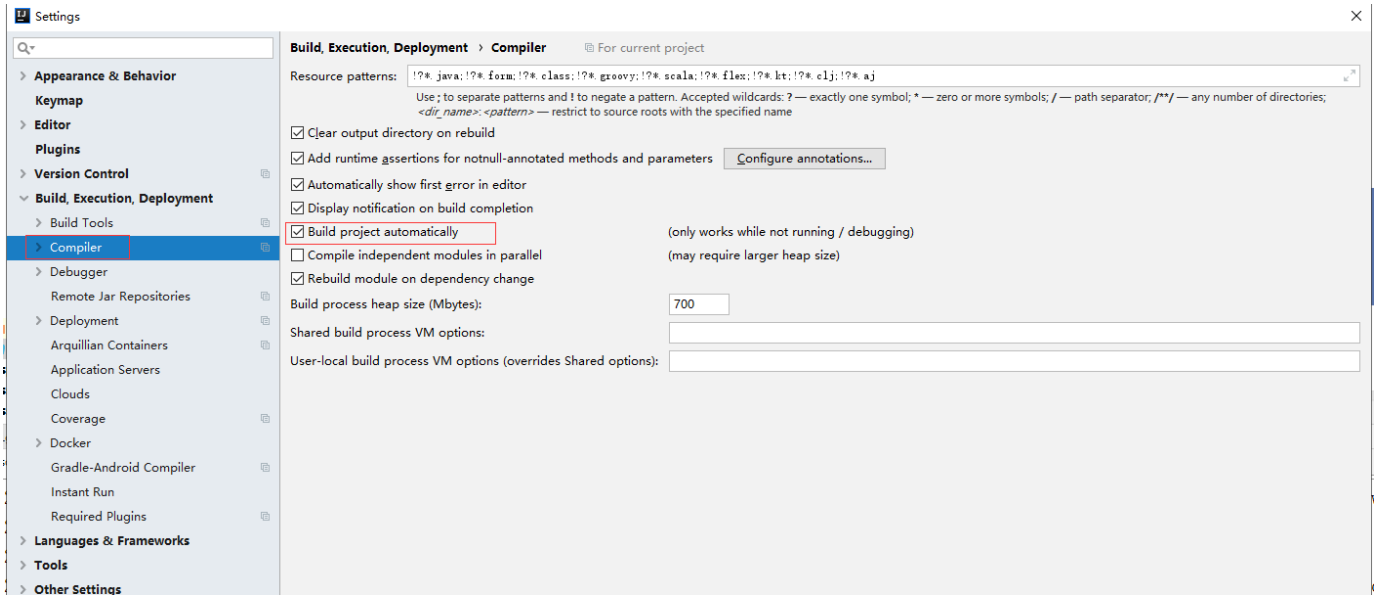
 * 回顾

* 能够掌握SpringBoot端口号和上下文路径配置

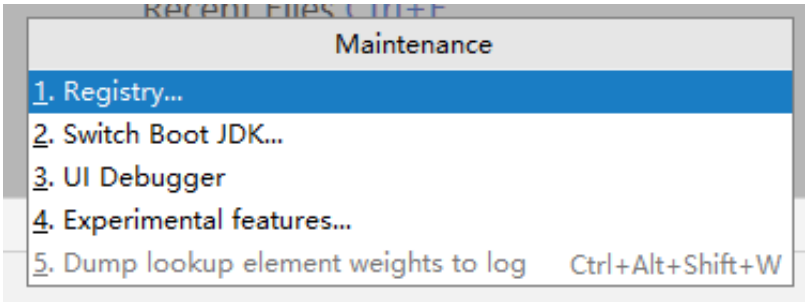
```
server.port=8082
server.servlet.context-path=/lg
```

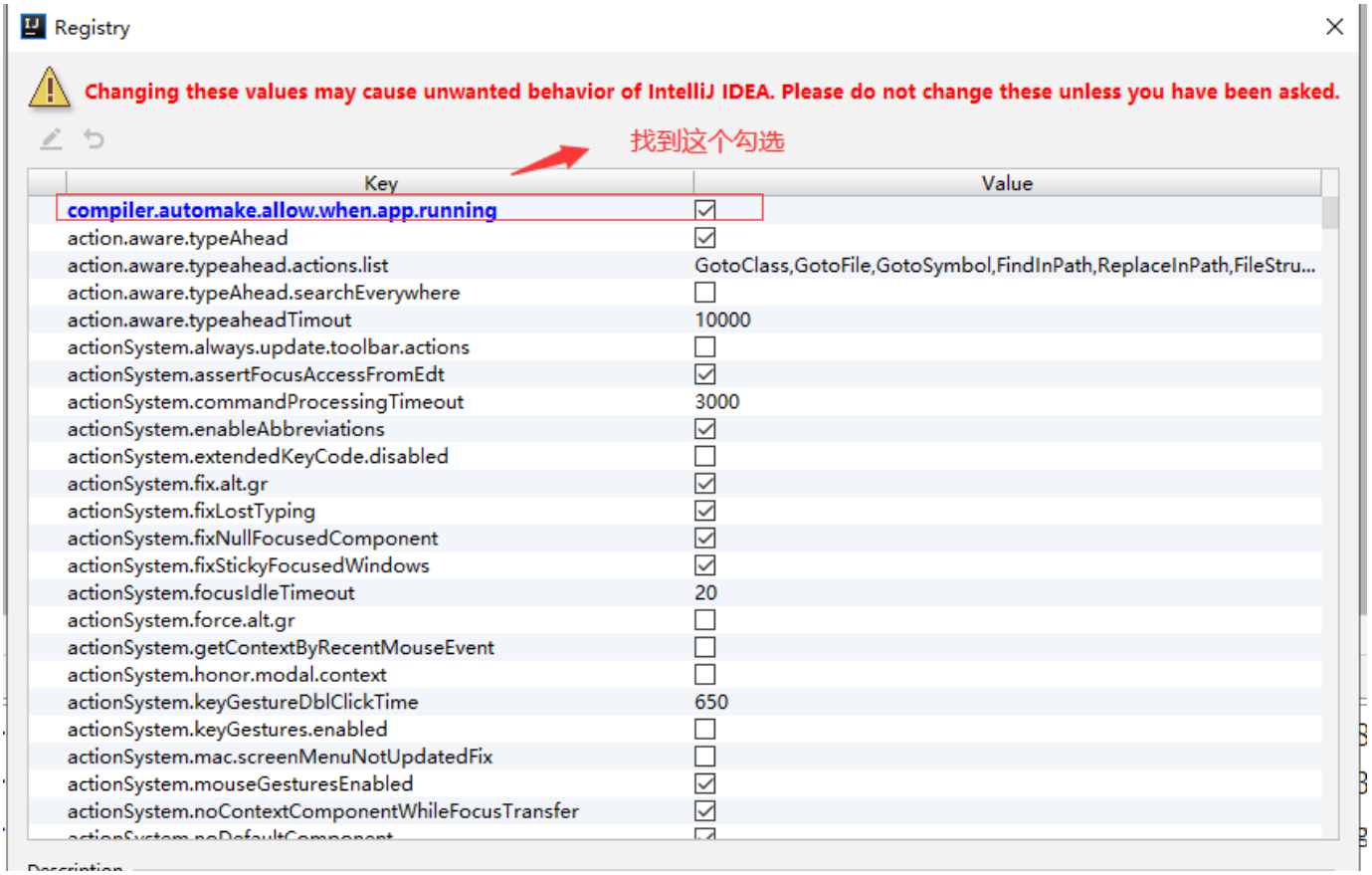
* 能够掌握SpringBoot工程热部署

```
* 添加依赖
<!--热部署配置-->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
</dependency>
```



* ctrl+shift+alt+/





* 能够掌握获取配置内容和日志的配置

```
* 案例一：（获取配置内容）
 * 配置
  * 在application.properties
 lg.content=test springboot
 * 获取
  @Value("${lg.content}")
  private String content;

* 案例二：（配置日志）
 * 配置
logging.file.path=D:/wook666/log
logging.level.com.lg.dao=debug
#logging.pattern.console=%d{yyyy-MM-dd} [%thread] %-5level %logger{50} -%msg%n
#logging.level.root=info # 默认选项
 * 使用
  * 使用Lombok的@Slf4j注解
  * 核心代码
    @Slf4j
    public class HelloController
    log.info("xiaohei123");
```

* 能够掌握SpringBoot添加Druid连接池

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

* 能够在SpringBoot使用JSP

```
* 添加依赖
<!-- 添加servlet依赖模块 -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <scope>provided</scope>
</dependency>
  <!-- 添加jstl标签库依赖模块 -->
  <dependency>
     <groupId>javax.servlet</groupId>
     <artifactId>jstl</artifactId>
  </dependency>
  <!-- 使用jsp引擎，springboot内置tomcat没有此依赖 -->
  <dependency>
     <groupId>org.apache.tomcat.embed</groupId>
     <artifactId>tomcat-embed-jasper</artifactId>
     <scope>provided</scope>
  </dependency>

* 配置
# 支持jsp
spring.mvc.view.prefix=/WEB-INF/jsps/
spring.mvc.view.suffix=.jsp
#spring.thymeleaf.cache=false
#spring.thymeleaf.check-template=false

* 构建webapp目录
  * 配置成webapp根目录
* 打包
 * pom.xml 配置
<build>
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
           </plugin>
```

```
        </plugins>
        <resources>
            <resource>
                <!-- 打包时将jsp文件拷贝到META-INF目录下-->
                <directory>src/main/webapp</directory>
                <targetPath>META-INF/resources</targetPath>
                <includes>
                    <include>**/**</include>
                </includes>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/**</include>
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/java</directory>
                <excludes>
                    <exclude>**/*.java</exclude>
                </excludes>
            </resource>
        </resources>
</build>
```

* 能够掌握SpringBoot文件上传

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

* 能够掌握在SpringBoot中使用过滤器

```
@Configuration
```

```
public class WebConfig {
    @Bean
    public FilterRegistrationBean encodingFilter(){
        FilterRegistrationBean registrationBean=new FilterRegistrationBean();
        CharacterEncodingFilter encodingFilter=new CharacterEncodingFilter();
        encodingFilter.setEncoding("UTF-8");
        registrationBean.addUrlPatterns("/*");
        registrationBean.setFilter(encodingFilter);
        return registrationBean;
    }

    @Bean
    public FilterRegistrationBean hiddenFilter(){
        FilterRegistrationBean registrationBean=new FilterRegistrationBean();
        HiddenHttpMethodFilter hiddenHttpMethodFilter=new HiddenHttpMethodFilte
        registrationBean.setFilter(hiddenHttpMethodFilter);
        registrationBean.addUrlPatterns("/*");
        return registrationBean;
    }
}

* 在SpringBoot可以添加配置替换上面的过滤器
 * HiddenHttpMethodFilter
spring.mvc.hiddenmethod.filter.enabled=true
 * 全局编码配置
spring.http.encoding.charset=UTF-8
spring.http.encoding.force=true
spring.http.encoding.enabled=true
# Character encoding to use to decode the URI.
server.tomcat.uri-encoding=UTF-8
```

* 能够使用SpringBoot集成Junit

```

* 添加依赖
 * IOC 测试
@RunWith(SpringRunner.class)
@SpringBootTest(classes = LgApplication.class)
public class AppTest {
    @Autowired
```

```
    private CompanyService companyService;
    @Test
    public void test1() {
        Company company = companyService.getCompany(1001);
        System.out.println(company);
    }
}
* 温馨提醒:
  * SpringRunner 继承 SpringJUnit4ClassRunner，然后啥事也不干
    * 估计是觉得SpringJUnit4ClassRunner名字太长了，换个名字
 * Mock 测试
* 方式一:
@RunWith(SpringRunner.class)
@SpringBootTest(classes = LgApplication.class)
public class AppTest2 {
    @Autowired
    private WebApplicationContext applicationContext;
    private MockMvc mockMvc;
    @Before
    public void before() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(applicationContext).b
    }
    @Test
    public void test18() throws Exception {
        String result = mockMvc.perform(delete("/emp/sexCount")).
                andExpect(status().isOk()).
                andDo(print()).andReturn().getResponse().getContentAsString();
        System.out.println(result);
    }
}

* 方式二:
@RunWith(SpringRunner.class)
@SpringBootTest(classes = LgApplication.class)
@AutoConfigureMockMvc
public class AppTest2 {
    @Autowired
    private MockMvc mockMvc;
    @Test
    public void test18() throws Exception {
```

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

* 能够掌握YAML文件的配置


```
* 案例一：把properties替换成yaml文件
# banner
spring:
  banner:
    image:
      location: classpath:banner.jpg
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/lg01?characterEncoding=utf-8
    username: root
    password: root
    druid:
      web-stat-filter:
```

```yaml
        enabled: false
  mvc:
    view:
      prefix: /WEB-INF/jsps/
      suffix: .jsp
      hiddenmethod:
        filter:
          enable: true
  thymeleaf:
    cache: false
    check-template: false
  servlet:
    multipart:
      max-request-size: 50MB
      max-file-size: 100MB
  http:
    encoding:
      charset: UTF-8
      force: true
      enabled: true

# server
server:
  port: 8088
  servlet:
    context-path: /lg

# mybatis
mybatis:
  mapper-locations: com/lg/dao/*.xml    */
  type-aliases-package: com.lg.bean
# log
logging:
  file:
    path: D:/wook666/log
  level:
    com.lg.dao: debug
# 自定义
lg:
  content: xiaohei
```

```

* 案例二：自定数据和获取
*  添加依赖
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
# 自定义
lg:
  content: xiaohei
  addresses:
    - provinceName: 广东
      cityName: 广州12
    - provinceName: 广东
      cityName: 佛山12
@ConfigurationProperties(prefix = "lg")
public class HelloController {
    @Value("${lg.content}")
    private String content;
    private List<Address> addresses;
    public void setAddresses(List<Address> addresses) {
        this.addresses = addresses;
    }
}

* @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

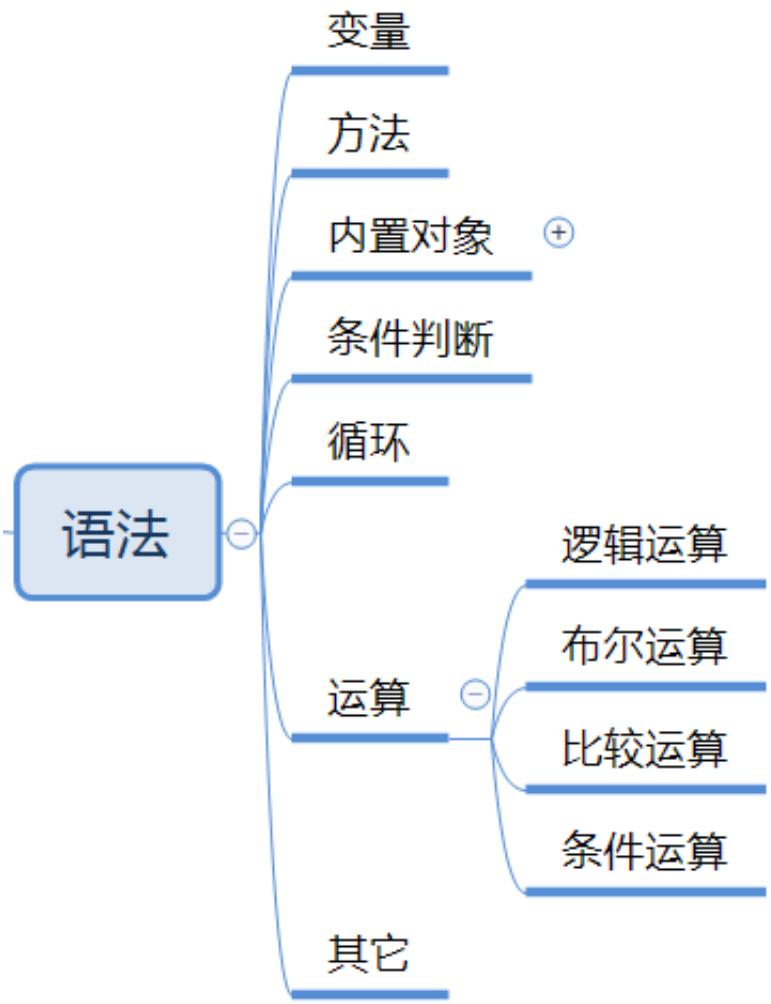
* 能够掌握Thymeleaf的语法

```
* 添加依赖
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
* 在resources目录创建templates目录
 * 新建test1.hmtl
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>hello thymeleaf</title>
</head>
<body>
    <h1>你好，亮哥教育</h1>
    <h1 th:text="${msg}">你好</h1>
</body>
</html>

* Controller
@Controller
@RequestMapping("/thymeleaf")
public class ThymeleafController {
    @RequestMapping("/test1")
    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

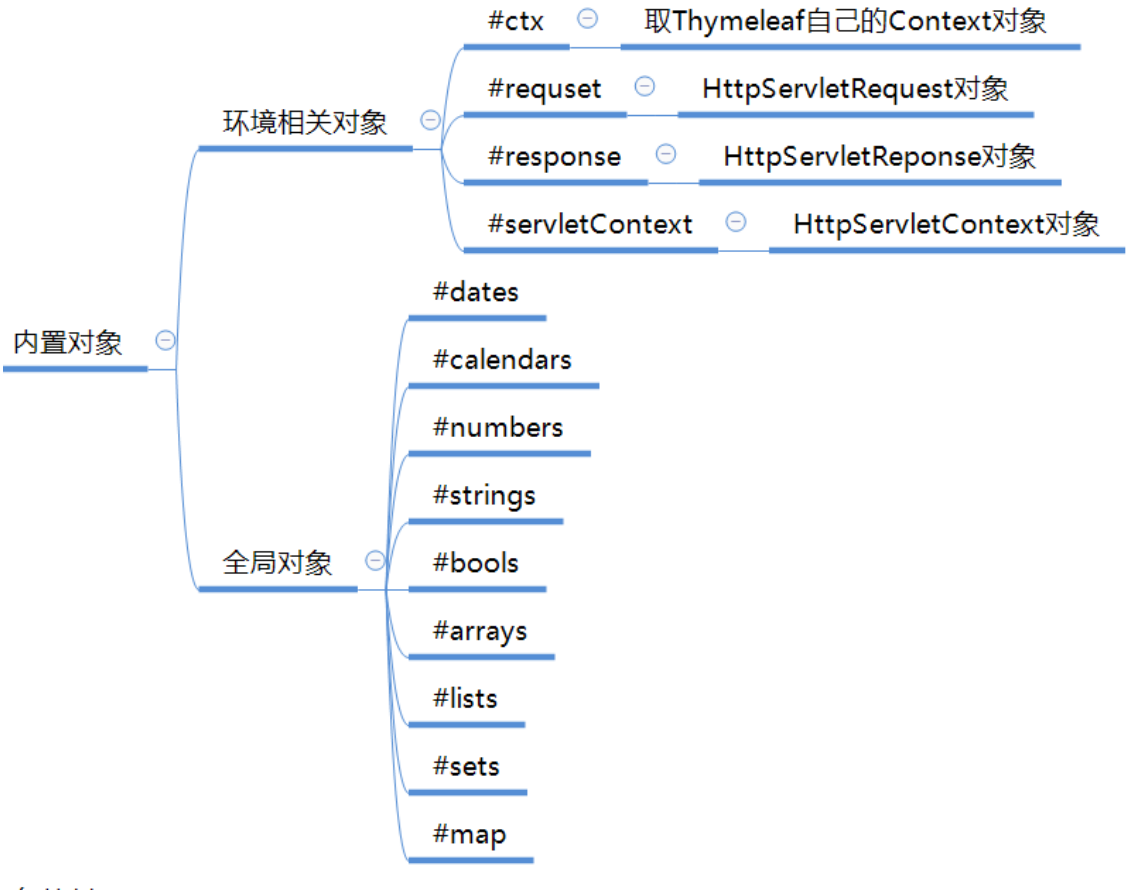



```
1  * 案例一：（变量）
2  <!DOCTYPE html>
```

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>hello thymeleaf</title>
</head>
<body>
    <h1>你好，亮哥教育</h1>
    <p>  name:<span th:text="${user.username}"></span><p>
    <p>  sex:<span th:text="${user.sex}"></span> </p>
    <h1 th:object="${user}">
        <p>  name:<span  th:text="*{username}"></span><p>
        <p>  sex:<span  th:text="*{sex}"></span> </p>
    </h1>
</body>
</html>
@RequestMapping("/test2")
public String test2(Model model){
  User user=new User();
  user.setUsername("xiaohei");
  user.setSex("男");
  model.addAttribute("user",user);
  return "test2";
}

* 案例二：（方法）
 <h1 th:object="${user}">
        <p>  name:<span  th:text="*{username.startsWith('xiaohei')}"></span><p>
        <p>  sex:<span  th:text="*{sex}"></span> </p>
 </h1>
```

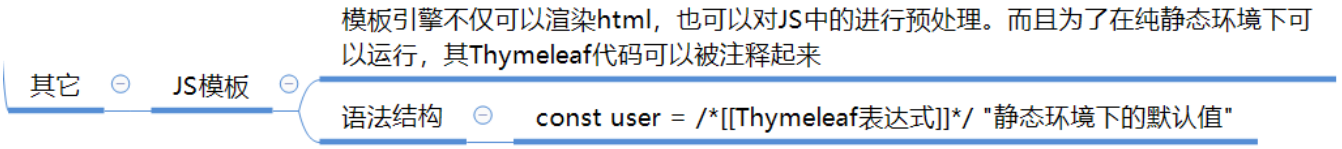
* 内置对象




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

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

* JS模板




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