实际应用

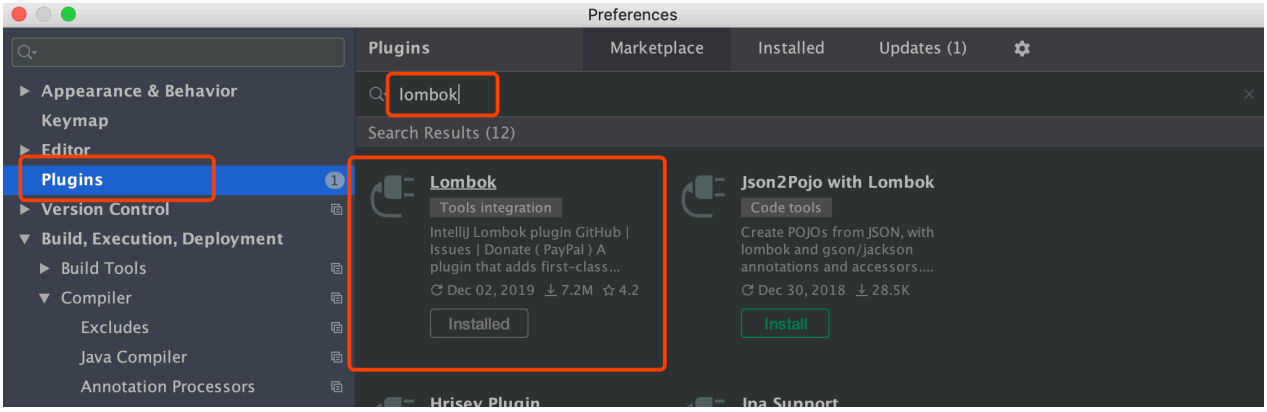
JSTL

1、pom.xml

```
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.10</version>
</dependency>
```

Lombok 的功能是简化实体类代码的编写工作，常用的方法 getter、setter、toString 等方法都可以由 Lombok 自动生成，开发者不需要自己手动编写，Lombok 的使用需要安装插件。



2、创建实体类

```
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class User {
    private Integer id;
    private String name;
}
```

3、Handler 中创建业务方法，返回 User 对象

```java
package com.southwind.controller;

import com.southwind.entity.User;
import com.southwind.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import java.util.Arrays;
import java.util.List;

@Controller
@RequestMapping("/user")
public class UserHandler {

    @Autowired
    private UserService userService;

    @GetMapping("/findAll")
    public ModelAndView findAll(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("index");
        modelAndView.addObject("list",userService.findAll());
        return modelAndView;
    }

    @GetMapping("findById/{id}")
    public ModelAndView findById(@PathVariable("id") Integer id){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("update");
        modelAndView.addObject("user",userService.findById(id));
        return modelAndView;
    }

    @PostMapping("/save")
    public String save(User user){
        userService.save(user);
        return "redirect:/user/findAll";
    }

    @GetMapping("/deleteById/{id}")
    public String deleteById(@PathVariable("id") Integer id){
        userService.deleteById(id);
        return "redirect:/user/findAll";
    }
```

```
    @PostMapping("/update")
    public String update(User user){
        userService.update(user);
        return "redirect:/user/findAll";
    }

}
```

4、Service

```
package com.southwind.service;

import com.southwind.entity.User;

import java.util.Collection;

public interface UserService {
    public Collection<User> findAll();
    public User findById(Integer id);
    public void save(User user);
    public void deleteById(Integer id);
    public void update(User user);
}
```

```
package com.southwind.service.impl;

import com.southwind.entity.User;
import com.southwind.repository.UserRepository;
import com.southwind.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Collection;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public Collection<User> findAll() {
        return userRepository.findAll();
    }

    @Override
    public User findById(Integer id) {
```

```
        return userRepository.findById(id);
    }

    @Override
    public void save(User user) {
        userRepository.save(user);
    }

    @Override
    public void deleteById(Integer id) {
        userRepository.deleteById(id);
    }

    @Override
    public void update(User user) {
        userRepository.update(user);
    }
}
```

5、Repository

```
package com.southwind.repository;

import com.southwind.entity.User;

import java.util.Collection;

public interface UserRepository {
    public Collection<User> findAll();
    public User findById(Integer id);
    public void save(User user);
    public void deleteById(Integer id);
    public void update(User user);
}
```

```
package com.southwind.repository.impl;

import com.southwind.entity.User;
import com.southwind.repository.UserRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

@Repository
public class UserRepositoryImpl implements UserRepository {
```

```
    private static Map<Integer,User> map;

    static {
        map = new HashMap<>();
        map.put(1,new User(1,"张三"));
        map.put(2,new User(2,"李四"));
        map.put(3,new User(3,"王五"));
    }

    @Override
    public Collection<User> findAll() {
        return map.values();
    }

    @Override
    public User findById(Integer id) {
        return map.get(id);
    }

    @Override
    public void save(User user) {
        map.put(user.getId(),user);
    }

    @Override
    public void deleteById(Integer id) {
        map.remove(id);
    }

    @Override
    public void update(User user) {
        map.put(user.getId(),user);
    }
}
```

6、JSP

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-23
  Time: 18:03
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
```

```
    <title>Title</title>
</head>
<body>
    <h1>Index</h1>
    <table>
        <tr>
            <th>编号</th>
            <th>姓名</th>
            <th>操作</th>
        </tr>
        <c:forEach items="${list}" var="user">
            <tr>
                <td>${user.id}</td>
                <td>${user.name}</td>
                <td>
                    <a href="/user/deleteById/${user.id}">删除</a>
                    <a href="/user/findById/${user.id}">修改</a>
                </td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-24
  Time: 13:04
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/user/save" method="post">
        <input type="text" name="id"/><br/>
        <input type="text" name="name"/><br/>
        <input type="submit"/>
    </form>
</body>
</html>
```

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
```

```
  Date: 2020-02-24
  Time: 13:04
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/user/update" method="post">
        <input type="text" name="id" value="${user.id}" readonly/><br/>
        <input type="text" name="name" value="${user.name}"/><br/>
        <input type="submit"/>
    </form>
</body>
</html>
```

Spring Boot 整合 Thymeleaf

Thymeleaf 是目前较为流行的视图层技术，Spring Boot 官方推荐使用 Thymeleaf。

什么是 Thymeleaf

Thymeleaf 是一个支持原生 THML 文件的 Java 模版，可以实现前后端分离的交互方式，即视图与业务数据分开响应，它可以直接将服务端返回的数据生成 HTML 文件，同时也可以处理 XML、JavaScript、CSS 等格式。

Thymeleaf 最大的特点是既可以直接在浏览器打开（静态方式），也可以结合服务端将业务数据填充到 HTML 之后动态生成的页面（动态方法），Spring Boot 支持 Thymeleaf，使用起来非常方便。

1、创建 Maven 工程，不需要创建 Web 工程，pom.xml

```
<?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>

    <groupId>com.southwind</groupId>
    <artifactId>springbootthymeleaf</artifactId>
    <version>1.0-SNAPSHOT</version>

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

```
        <version>2.2.4.RELEASE</version>
    </parent>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
    </dependencies>


</project>
```

2、application.yml

```
spring:
  thymeleaf:
    prefix: classpath:/templates/   #模版路径
    suffix: .html                   #模版后缀
    servlet:
      content-type: text/html       #设置 Content-type
    encoding: UTF-8                 #编码方式
    mode: HTML5                     #校验 H5 格式
    cache: false                    #关闭缓存，在开发过程中可以立即看到页面修改的结果
```

3、创建 Handler

```
package com.southwind.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/hello")
public class HelloHandler {

    @GetMapping("/index")
    public ModelAndView index(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("index");
        modelAndView.addObject("name","张三");
```

```
        return modelAndView;
    }

}
```

4、启动类

```
package com.southwind;

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

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

5、HTML

```
<!DOCTYPE html>
<html lang="en">
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>Index</h1>
    <p th:text="${name}">Hello World</p>
</body>
</html>
```

如果需要加载后台返回的业务数据，则需要在 HTML 页面中使用 Thymeleaf 模版标签来完成。

1、需要引入模版标签。

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

2、通过特定的标签完成操作。

```
 <p th:text="${name}">Hello World</p>
```

Thymeleaf 模版标签不同于 JSTL，Thymeleaf 模版标签是直接嵌入到 HTML 原生标签内部。

Thymeleaf 常用标签

- th:text

th:text 用于文本的显示，将业务数据的值填充到 HTML 标签中。

- th:if

th:if 用于条件判断，对业务数据的值进行判断，如果条件成立，则显示内容，否则不显示，具体的使用如下所示。

```
@GetMapping("/if")
public ModelAndView ifTest(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("score",90);
    return modelAndView;
}
```

```
<p th:if="${score>=90}">优秀</p>
<p th:if="${score<90}">良好</p>
```

- th:unless

th:unless 也用作条件判断，逻辑与 th:if 恰好相反，如果条件不成立则显示，否则不显示。

```
@GetMapping("/unless")
public ModelAndView unlessTest(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("score",90);
    return modelAndView;
}
```

```
<p th:unless="${score>=90}">优秀</p>
<p th:unless="${score<90}">良好</p>
```

- th:switch th:case

th:switch th:case 两个结合起来使用，用作多条件等值判断，逻辑与 Java 中的 switch-case 一致，当 switch 中的业务数据等于某个 case 时，就显示该 case 对应的内容。

```
@GetMapping("/switch")
public ModelAndView switchTest(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("studentId",1);
    return modelAndView;
}
```

```
<div th:switch="${studentId}">
    <p th:case="1">张三</p>
    <p th:case="2">李四</p>
    <p th:case="3">王五</p>
</div>
```

- th:action

用来指定请求的 URL，相当于 form 表单中的 action 属性

```
<form th:action="@{/hello/login}" method="post">
    <input type="submit"/>
</form>
```

```
@GetMapping("/redirect/{url}")
public String redirect(@PathVariable("url") String url, Model model){
    model.addAttribute("url","/hello/login");
    return url;
}
```

```
<form th:action="${url}" method="post">
    <input type="submit"/>
</form>
```

如果 action 的值直接写在 HTML 中，则需要使用 @{}，如果是从后台传来的数据，则使用${}。

- th:each

用来遍历集合

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

```
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class User {
    private Integer id;
    private String name;
}
```

```
@GetMapping("/each")
public ModelAndView each(){
    List<User> list = Arrays.asList(
            new User(1,"张三"),
            new User(2,"李四"),
            new User(3,"王五"));
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("list",list);
    return modelAndView;
}
```

```
<table>
  <tr>
    <th>编号</th>
    <th>姓名</th>
  </tr>
  <tr th:each="user:${list}">
    <td th:text="${user.id}"></td>
    <td th:text="${user.name}"></td>
  </tr>
</table>
```

- th:value

用来给标签赋值。

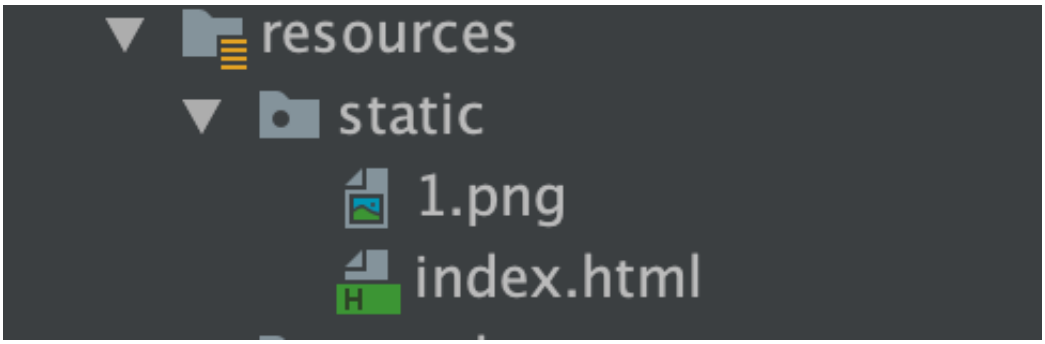
```
@GetMapping("/value")
public ModelAndView value(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("value","Spring Boot");
    return modelAndView;
}
```

```
<input type="text" th:value="${value}"/>
```

- th:src

用来引入静态资源，相当于 HTML 原生标签 img、script 的 src 属性。

图片，css，js，静态加载的 html 都需要放置在 resources/static 文件中




```
@GetMapping("/src")
public ModelAndView src(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("src","/1.png");
    return modelAndView;
}
```

```
<img th:src="${src}"/>
```

如果 src 的值直接写在 HTML 中

```
<img th:src="@{/1.png}"/>
```

- th:href

用作设置超链接的 href

```
@GetMapping("/href")
public ModelAndView href(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("test");
    modelAndView.addObject("href","https://www.baidu.com");
    return modelAndView;
}
```

```
<a th:href="${href}">百度</a>
```