文件下载

1、JSP 页面中添加超链接，进行下载。

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-13
  Time: 14:01
  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>
    <a href="/file/download?fileName=1.png">1.png</a><br/>
    <a href="/file/download?fileName=2.png">2.png</a><br/>
    <a href="/file/download?fileName=3.png">3.png</a>
</body>
</html>
```

2、业务方法

```
@GetMapping("/download")
public void download(String fileName,
                     HttpServletRequest request,
                     HttpServletResponse response){
    if(fileName!=null){
        String path =
request.getSession().getServletContext().getRealPath("file");
        File file = new File(path,fileName);
        OutputStream outputStream = null;
        if(file.exists()){
            //设置下载文件
            response.setContentType("application/force-download");
            //设置文件名
            response.setHeader("Content-
Disposition","attachment;filename="+fileName);
            try {
                outputStream = response.getOutputStream();
                outputStream.write(FileUtils.readFileToByteArray(file));
                outputStream.flush();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }finally {
                if(outputStream!=null){
                    try {
                        outputStream.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
}
```

Spring MVC 数据校验

数据校验是每个项目中必不可少的模块，Spring MVC 提供了两种数据校验的组件：

- 基于 Validator 接口进行校验
- 使用 Annotation JSR-303 标准校验

使用基于 Validator 接口进行校验会复杂一些，具体的数据校验的规则需要开发者手动设置。而使用 Annotation JSR-303 标准会相对简单一些，开发者不需要编写校验规则，直接通过注解的形式给每一条数据添加验证规则，具体操作是直接在实体类的属性上添加对应的校验注解即可。

基于 Validator 接口

1、创建实体类

```
package com.southwind.entity;

import lombok.Data;

@Data
public class Student {
    private String name;
    private String password;
}
```

2、自定义数据校验器 StudentValidation，实现 Validator 接口，重写接口的抽象方法，加入校验规则。

```
package com.southwind.validation;

import com.southwind.entity.Student;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

public class StudentValidation implements Validator {
```

```
    @Override
    public boolean supports(Class<?> aClass) {
        return Student.class.equals(aClass);
    }

    @Override
    public void validate(Object o, Errors errors) {
        ValidationUtils.rejectIfEmpty(errors,"name",null,"姓名不能为空");
        ValidationUtils.rejectIfEmpty(errors,"password",null,"密码不能为空");
    }
}
```

3、控制层业务方法

```
package com.southwind.controller;

import com.southwind.entity.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/validate")
public class ValidateHandler {

    /**
     * 给JSP表单绑定模型对象
     * @param model
     * @return
     */
    @GetMapping("/login")
    public String login(Model model){
        model.addAttribute(new Student());
        return "login";
    }

    /**
     * 数据校验
     * @param student
     * @param bindingResult
     * @return
     */
    @PostMapping("/login")
    public String login(@Validated Student student, BindingResult
bindingResult){
```

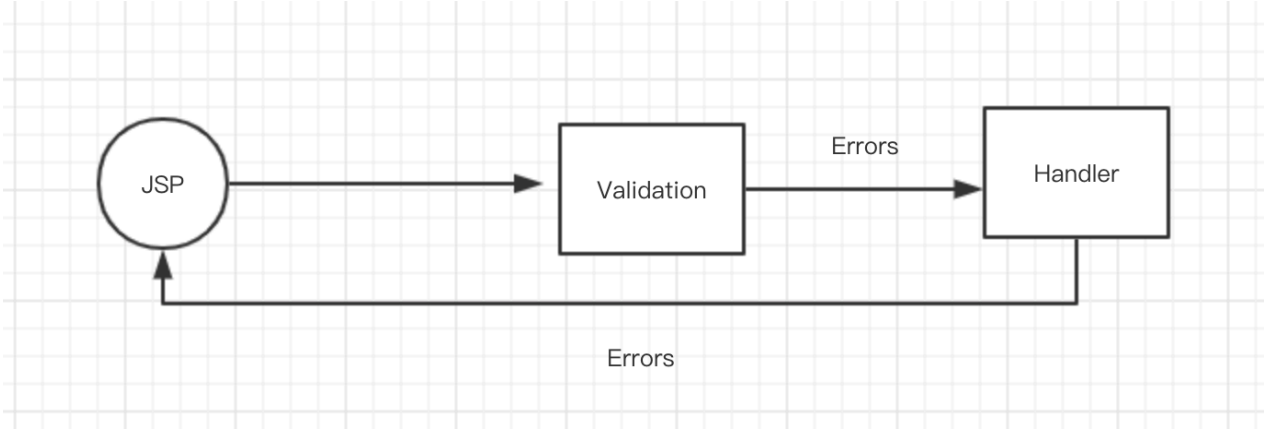
```
        if(bindingResult.hasErrors()){
            return "login";
        }
        return "success";
    }
}
```

4、springmvc.xml 配置 validator

```
<mvc:annotation-driven validator="studentValidator"></mvc:annotation-driven>
<bean id="studentValidator"
class="com.southwind.validation.StudentValidation"></bean>
```

5、JSP

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-13
  Time: 14:23
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>学生登陆</h1>
    <form:form modelAttribute="student" action="/validate/login"
method="post">
        学生姓名:<form:input path="name"></form:input><form:errors path="name">
</form:errors><br/>
        学生密码:<form:input path="password"></form:input><form:errors
path="password"></form:errors><br/>
        <input type="submit" value="提交"/>
    </form:form>
</body>
</html>
```



Annotation JSR-303 标准

Hibernater Validator，通过注解完成校验规则的绑定。

@Null 只能为 null

@NotNull 不能为 null

@Size 设置数据长度

@NotEmpty 不能为空

String str = null;

String str = "";

1、pom.xml

```
<!-- JSR-303 -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.3.6.Final</version>
</dependency>
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.1.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.logging</groupId>
  <artifactId>jboss-logging</artifactId>
  <version>3.4.1.Final</version>
</dependency>

<!-- JDK9以上 -->
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
```

```
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.3.0</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.3.0</version>
</dependency>
<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>
```

2、创建实体类，通过注解的方式给属性指定校验规则。

```
package com.southwind.entity;

import lombok.Data;
import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;

import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

@Data
public class Account {
    @NotEmpty(message = "用户名不能为空")
    private String username;
    @Size(min = 6,max = 20,message = "密码长度为6-20位")
    private String password;
    @Email(regexp = "^[a-zA-Z0-9_.-]+@[a-zA-Z0-9-]+(\\\\\\\\.[a-zA-Z0-9-
]+)*\\\\\\\\.[a-zA-Z0-9]{2,6}$",message = "请输入正确的邮箱")
    private String email;
    @Pattern(regexp = "^((13[0-9])|(14[5|7])|(15([0-3]|[5-9]))|(18[0,5-
9]))\\\\\\\\\\\\\\\\\\d{8}$",message = "请输入正确的电话格式")
    private String phone;
}
```

3、业务方法

```
@GetMapping("/register")
public String register(Model model){
    model.addAttribute(new Account());
    return "register";
}

@PostMapping("/register")
public String register(@Valid Account account,BindingResult bindingResult){
    if(bindingResult.hasErrors()){
        return "regiser";
    }
    return "success";
}
```

4、springmvc.xml

```
<mvc:annotation-driven></mvc:annotation-driven>
```

5、register.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-13
  Time: 16:15
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>用户注册</h1>
    <form:form modelAttribute="account" action="/validate/register"
method="post">
        用户名：<form:input path="username"></form:input><form:errors
path="username"></form:errors><br/>
        密码：<form:input path="password"></form:input><form:errors
path="password"></form:errors><br/>
        邮箱：<form:input path="email"></form:input><form:errors path="email">
</form:errors><br/>
        电话：<form:input path="phone"></form:input><form:errors path="phone">
</form:errors><br/>
        <input type="submit" value="提交"/>
    </form:form>
</body>
```

```
</html>
```

Spring MVC表单标签库

1、Student 实体类

```
package com.southwind.entity2;

import lombok.Data;

@Data
public class Student {
    private Integer id;
    private String name;
    private Integer age;
    private String gender;
}
```

2、Handler

```
package com.southwind.controller;

import com.southwind.entity2.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/student")
public class StudentHandler {

    @RequestMapping("/get")
    public String get(Model model){
        Student student = new Student();
        student.setId(1);
        student.setName("张三");
        student.setAge(22);
        student.setGender("男");
        model.addAttribute("student",student);
        return "student";
    }
}
```

3、JSP

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

```
  Date: 2020-02-13
  Time: 17:38
  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>
    <h1>修改学生信息</h1>
    <form action="" method="post">
        学生编号:<input type="text" name="id" value="${student.id}" readonly/>
<br/>
        学生姓名:<input type="text" name="name" value="${student.name}"/><br/>
        学生年龄:<input type="text" name="age" value="${student.age}"/><br/>
        学生性别:<input type="text" name="gender" value="${student.gender}"/>
<br/>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

使用 Spring MVC 表单标签可以直接将业务数据绑定到 JSP 表单中，非常简单。

表单标签库的使用

1、JSP 页面导入 Spring MVC 表单标签库。

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

2、将 form 表单与业务数据进行绑定，通过 modelAttribute 属性完成绑定，将 modelAttribute 的值设置为控制器向 model 对象存值时的 name 即可。

```
<form:form modelAttribute="student" action="/student/update" method="post">
    学生编号:<form:input path="id"></form:input><br/>
    学生姓名:<form:input path="name"></form:input><br/>
    学生年龄:<form:input path="age"></form:input><br/>
    学生性别:<form:input path="gender"></form:input><br/>
    <input type="submit" value="提交"/>
</form:form>
```

常用标签

1、form 标签

```
<form:form modelAttribute="student" method="post"></form:form>
```

渲染的是 HTML 中的 `<form></form>`，通过 modelAttribute 属性绑定具体的业务数据。

2、input 标签

```
<form:input path="name"></form:input>
```

渲染的是 HTML 中的 `<input type="text"/>`，from 标签绑定的是业务数据，input 标签绑定的是业务数据中的属性值，通过 path 与业务数据的属性名对应，并支持级联。

```
package com.southwind.entity2;

import lombok.Data;

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

```
package com.southwind.entity2;

import lombok.Data;

@Data
public class Student {
    private Integer id;
    private String name;
    private Integer age;
    private String gender;
    private Address address;
}
```

```
@RequestMapping("/get")
public String get(Model model){
    Student student = new Student();
    student.setId(1);
    student.setName("张三");
    student.setAge(22);
    student.setGender("男");
    Address address = new Address();
    address.setId(1);
```

```
    address.setName("科技路");
    student.setAddress(address);
    model.addAttribute("student",student);
    return "student2";
}
```

```
<form:form modelAttribute="student" action="/student/update" method="post">
    学生编号:<form:input path="id"></form:input><br/>
    学生姓名:<form:input path="name"></form:input><br/>
    学生年龄:<form:input path="age"></form:input><br/>
    学生性别:<form:input path="gender"></form:input><br/>
    学生地址:<form:input path="address.name"></form:input><br/>
    <input type="submit" value="提交"/>
</form:form>
```

3、password 标签

```
<form:password path="password"></form:password>
```

渲染的是 HTML 中的 `<input type="password"/>`，通过 path 与业务数据的属性名对应，password 标签的值不会在页面显示。

4、checkbox 标签

```
<form:checkbox path="hobby" value="读书"></form:checkbox>
```

渲染的是 HTML 中的 `<input type="checkbox"/>`，通过 path 与业务数据的属性名对应，可以绑定 boolean、数组和集合。

如果绑定 boolean 类型的变量，该变量值为 true，则表示选中，false 表示不选中。

```
student.setFlag(true);

checkbox:<form:checkbox path="flag" value="1"></form:checkbox>
```

如果绑定数组和集合类型，集合中的元素等于 checkbox 的 value 值，则该项选中，否则不选中。

```
student.setHobby(Arrays.asList("读书","看电影","旅行"));

<form:checkbox path="hobby" value="读书"></form:checkbox>读书<br/>
<form:checkbox path="hobby" value="看电影"></form:checkbox>看电影<br/>
<form:checkbox path="hobby" value="打游戏"></form:checkbox>打游戏<br/>
<form:checkbox path="hobby" value="听音乐"></form:checkbox>听音乐<br/>
<form:checkbox path="hobby" value="旅行"></form:checkbox>旅行<br/>
```

5、checkboxs 标签

```
<form:checkboxs items="${student.hobby}" path="selectHobby"></form:checkboxs>
```

渲染的是 HTML 中的一组 `<input type="checkbox"/>`，这里需要结合 items 和 path 两个属性来使用，items 绑定被遍历的集合或数组，path 绑定选中的集合或数组，items 是全部选型，path 为默认选中的选型。

```
student.setHobby(Arrays.asList("读书","看电影","打游戏","听音乐","旅行"));
student.setSelectHobby(Arrays.asList("读书","看电影"));

<form:checkboxes path="selectHobby" items="${student.hobby}">
</form:checkboxes>
```

需要注意的是 path 可以直接绑定业务数据的属性，items 则需要通过 EL 表达式从域对象中取值，不能直接写属性名。

6、radiobutton 标签

```
<form:radiobutton path="radioId" value="0"></form:radiobutton>
```

渲染的是 HTML 中的一个 `<input type="radio"/>`，绑定的数据与标签的 value 值相等为选中状态，否则为不选中状态。

```
student.setRadioId(1);

<form:radiobutton path="radioId" value="0"></form:radiobutton>男
<form:radiobutton path="radioId" value="1"></form:radiobutton>女
```

7、radiobuttons 标签

```
<form:radiobuttons items="${student.grade}" path="selectGrade">
</form:radiobuttons>
```

渲染的是 HTML 中的一组 `<input type="radio"/>`，这里需要结合 items 和 path 两个属性来使用，items 绑定被遍历的集合或数组，path 绑定被选中的值，items 是全部选型，path 为默认选中的选型。

```
Map<Integer,String> gradeMap = new HashMap<>();
gradeMap.put(1,"一年级");
gradeMap.put(2,"二年级");
gradeMap.put(3,"三年级");
gradeMap.put(4,"四年级");
gradeMap.put(5,"五年级");
gradeMap.put(6,"六年级");
student.setGradeMap(gradeMap);
student.setSelectGrade(3);

<form:radiobuttons path="selectGrade" items="${student.gradeMap}">
</form:radiobuttons>
```

8、select 标签

```
<form:select items="${student.citys}" path="selectCity"/>
```

渲染的是 HTML 中的一个 `<select/>`，这里需要结合 items 和 path 两个属性来使用，items 绑定被遍历的集合或数组，path 绑定被选中的值，用法与 radiobuttons 标签一致。

```
Map<Integer,String> cityMap = new HashMap<>();
cityMap.put(1,"北京");
cityMap.put(2,"上海");
cityMap.put(3,"广州");
cityMap.put(4,"深圳");
student.setCityMap(cityMap);
student.setSelectCity(2);

<form:select path="selectCity" items="${student.cityMap}"></form:select>
```

9、form:select 标签结合 form:options 使用

form:select 只定义 path 属性，在 form:select 标签内部添加一个子标签 form:options，设置 items 属性。

```
<form:select path="selectCity">
    <form:options items="${student.cityMap}"></form:options>
</form:select>
```

10、form:select 标签结合 form:option 使用

form:select 定义 path 属性，给每一个 form:option 设置 value 属性，path 与哪个 value 相等，该项默认选中。

```
<form:select path="selectCity">
    <form:option value="1">西安</form:option>
    <form:option value="2">杭州</form:option>
    <form:option value="3">成都</form:option>
</form:select>
```

Spring MVC国际化

国际化是指同一个应用程序在不同语言设置的浏览器中，自动显示不同的语言，Spring MVC 对国际化操作做了很好的集成，只需要简单配置即可实现国际化。

1、springmvc.xml 配置。

```
<!-- 国际化资源文件 -->
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <!-- 多语言配置文件放在根路径，以 language 开头 -->
    <property name="basename" value="classpath:language"></property>
    <property name="useCodeAsDefaultMessage" value="true"></property>
</bean>

<!-- 拦截器 -->
<mvc:interceptors>
    <bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
        <property name="paramName" value="lang"></property>
    </bean>
</mvc:interceptors>

<!-- 配置 SessionLocaleResolver，动态获取 Locale 对象，存入 Session -->
<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver"></bean>
```

2、创建国际化资源文件 language_en_US.properties，language_zh_CN.properties，分别存储英文和中文资源。

```
language.cn = \u4E2D\u6587
language.en = English
info = login
username = username
password = password
repassword = repassword
tel = tel
email = email
submit = submit
reset = reset
```

```
language.cn = \u4E2D\u6587
language.en = English
info = \u767B\u9646
username = \u7528\u6237\u540D
password = \u5BC6\u7801
repassword = \u786E\u8BA4\u5BC6\u7801
tel = \u7535\u8BDD
email = \u7535\u5B50\u90AE\u7BB1
submit = \u63D0\u4EA4
reset = \u91CD\u7F6E
```

3、Handler

```
package com.southwind.controller;

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

@Controller
@RequestMapping("/inter")
public class InterHandler {

    @GetMapping("/index")
    public String index(){
        return "inter";
    }
}
```

4、JSP

```
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-14
  Time: 15:32
```

```
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <a href="index?lang=en_US">English</a>
    <a href="index?lang=zh_CN">中文</a>
    <h1><spring:message code="info"></spring:message></h1>
    <form>
        <spring:message code="username"/>:<input type="text"/><br/>
        <spring:message code="password"/>:<input type="password"/><br/>
        <spring:message code="repassword"/>:<input type="password"/><br/>
        <spring:message code="tel"/>:<input type="text"/><br/>
        <spring:message code="email"/>:<input type="text"/><br/>
        <input type="submit" value="<spring:message code="submit"/> "/>
        <input type="reset" value="<spring:message code="reset"/> "/>
    </form>
</body>
</html>
```