一、接收文件格式数据

1.1 在springmvc配置文件中配置文件解析器

```
<!--文件解析器-->
<!--此id必须为multipartResolver-->
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!--设置一次上传文件的总大小-->
    <property name="maxUploadSize" value="5242880"/>
</bean>
```

1.2 请求视图

```
<%--
    上传文件三要素:
    当前form表单必须为post
    当前form表单类型必须为multipart/form-data
    必须提供一个文件上传项<input type="file" name="fileParam"><br/>
--%>
<form action="${pageContext.request.contextPath}/day02/fileUpload"
      method="post" enctype="multipart/form-data">
    上传一个文件: <input type="file" name="fileParam"><br/>
    再传一个文件: <input type="file" name="fileParam"><br/>
    <input type="submit" value="上传文件">
</form>
```

1.3 编写处理器

```
//注意处理器参数必须和页面input中name属性一致
@RequestMapping("/fileUpload")
public String fileUpload(MultipartFile[] fileParam) throws IOException {
    //指定上传文件的路径
    File targetFile = new File("D:\\file");
    for (MultipartFile multipartFile : fileParam) {
        //指定上传文件名
        String filename = UUID.randomUUID()+multipartFile.getOriginalFilename();
        //把fileParam文件上传到targetFile文件下名字叫filename
        multipartFile.transferTo(new File(targetFile, filename));
    }
    return "success";
}
```

二、接收json数据用@RequestBody将其转成javaBean对象

2.1 导入jquery

```
<script type="text/javascript" src="${pageContext.request.contextPath}/js/jquery.min.js">
</script>
```

2.2 编写ajax请求

```
<button id="jsonBtn">发送json格式的utf-8数据</button>
<script type="text/javascript">
    $(function () {
        $("#jsonBtn").click(function () {
            $.ajax({
                type: "POST",
                url: "${pageContext.request.contextPath}/day02/getRequestBody",
                contentType: "application/json;charset=UTF-8",
                data: '{"id":"1","name":"小明"}',
                success: function(data){
                    <%--回调函数暂时用不到--%>
                }
            })
        })
    })
</script>
```

2.3 编写处理器

```
//打印结果：User{id=1, name='小明'}
@RequestMapping("/getRequestBody")
public void getRequestBody(@RequestBody User user){
    System.out.println(user);
}
```

三、异步请求用@ResponseBody返回数据

3.1 修改上面案例处理器返回一个字符串

```
/**
 * 页面回调函数中alert出回调的data为：操作成功
 * produces = "text/html;charset=utf-8"指定返回值的数据类型和编码格式
 */
@RequestMapping(value = "/getRequestBody", produces = "text/html;charset=utf-8")
@ResponseBody
public String getRequestBody(@RequestBody User user){
    System.out.println(user);
    return "操作成功";
}
```

3.2 继续修改案例处理器返回一个对象

```
@RequestMapping(value = "/getRequestBody")
@ResponseBody
public User getRequestBody(@RequestBody User user){
    System.out.println(user);
    return user;
}
```

3.3 使用传统servlet方式返回数据

```
//了解即可
@RequestMapping("/ajaxGetValueOld")
public void ajaxGetValueOld(HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=UTF-8");
    response.getWriter().write("传智播客");
}
```

四、restful风格开发介绍

4.1 什么是restful?

> restful简称REST，全称是Representational State Transfer。
>
> REST是一种架构风格，其强调HTTP应当以资源为中心。
>
> 它制定了HTTP请求四个动作，分别表示对资源的CRUD操作:
>
> GET(获取)、POST(新建)、PUT(更新)、DELETE(删除)

	原来风格URL写法	REST风格的URL写法
获取所有	/findUsers	GET /users
根据ID获取	/findUserById?id=1	GET /user/1
新增	/addUser	POST /user
根据ID删除	/deleteUserById?id=1	DELETE /user/1
根据ID修改	/updateUserById?id=1	PUT /user/1

4.2 restful风格处理器编写

```
//PostMapping相当于@RequestMapping(method = RequestMethod.POST)
@PostMapping("/restful")
@ResponseBody
public String restfulPost(){
    System.out.println("post------");
    return "success";
}

@GetMapping("/restful")
@ResponseBody
public String restfulGet(){
    System.out.println("get------");
    return "success";
}

@PutMapping("/restful")
@ResponseBody
public String restfulPut(){
    System.out.println("put------");
    return "success";
}

@DeleteMapping("/restful")
@ResponseBody
public String restfulDelete(){
    System.out.println("delete------");
    return "success";
}
```

4.3 使用@PathVariable接收REST风格请求地址中占位符的值

4.3.1 请求视图

```
<button id="restBtn">rest</button>
```

```
<script type="text/javascript">
    $(function () {
        $("#restBtn").click(function () {
            $.ajax({
                type: "POST",
                url: "${pageContext.request.contextPath}/day02/restful/1",
                contentType: "application/json;charset=UTF-8",
                data: '{"id":"1","name":"小明"}',
                success: function(data){
                    alert(data)
                }
            })
        })
    })
</script>
```

4.3.2 处理器

```
//注意：如果是JSON格式数据依然需要@RequestBody来接收
@PostMapping("/restful/{id}")
@ResponseBody
public String restfulPost(@PathVariable String id, @RequestBody User user){
    System.out.println("post------"+id);
    System.out.println(user);
    return "success";
}
```

4.4 @RestController注解说明

restful风格多用于前后端绝对分离的项目开发中，这时同步请求将无法使用，我们所有的处理器都将成为返回数据的异步请求。

此刻，可以将所有处理器方法上的@ResponseBody注解提取到类上去。

然后，进一步可以使用@RestController来替代@Controller和@ResponseBody两个注解。

写法如下：

```
@RestController
@RequestMapping("/day02")
public class Day02Controller {

}
```

五、springmvc中的请求转发

5.1 请求视图

```
<a href="${pageContext.request.contextPath}/day02/forwardMvcView">请求转发经过视图解析器</a><br/>
<a href="${pageContext.request.contextPath}/day02/forwardMvc">请求转发不经过视图解析器</a><br/>
<a href="${pageContext.request.contextPath}/day02/forwardOld">请求转发传统的方式</a><br/>
```

5.2 处理器

```
//请求转发经过视图解析器
@RequestMapping("/forwardMvcView")
public String forwardMvcView(){
    return "success";
}

//forward:关键字后面的路径表示不再经过视图解析器
@RequestMapping("/forwardMvc")
public String forwardMvc(){
    return "forward:/WEB-INF/pages/success.jsp";
}

//请求转发传统的方式
@RequestMapping("/forwardOld")
public void forwardOld(HttpServletRequest request, HttpServletResponse response) throws
Exception {
    request.getRequestDispatcher("/WEB-INF/pages/success.jsp").forward(request, response);
}
```

六、springmvc中的重定向

6.1 请求视图

```
<a href="${pageContext.request.contextPath}/day02/redirectMvc">mvc方式的重定向</a><br/>
<a href="${pageContext.request.contextPath}/day02/redirectOld">传统方式的重定向</a><br/>
```

6.2 处理器

```
//注意：WEB-INF下的所有页面都必须经过请求转发，而不能直接访问。
@RequestMapping("/redirectMvc")
public String redirectMvc(){
    //return "redirect:forwardMvcView";
    return "redirect:/day02/forwardMvcView";
}

@RequestMapping("/redirectOld")
public void redirectOld(HttpServletRequest request, HttpServletResponse response) throws
Exception {
    response.sendRedirect(request.getContextPath()+"/day02/forwardMvcView");
}
```

七、springmvc向request域设置值

7.1 使用Model对象

```
@RequestMapping("/getRequestValue")
public String getRequestValue(Model model){
    //向request域中放置一个值"小明", key为value
    model.addAttribute("value", "小明");
    return "success";
}
```

7.2 使用ModelAndView

```
@RequestMapping("/getRequestValueMv")
public ModelAndView getRequestValueMv(ModelAndView mv){
    //向request域中放置一个值"小明", key为value
    mv.addObject("value", "小明");
    mv.setViewName("success");
    return mv;
}

//或者

@RequestMapping("/getRequestValueMv")
public ModelAndView getRequestValueMv(){
    ModelAndView mv = new ModelAndView();
    //向request域中放置一个值"小明", key为value
    mv.addObject("value", "小明");
    mv.setViewName("success");
    return mv;
}
```

7.3 传统方式

```
@RequestMapping("/getRequestValueOld")
public String getRequestValueOld(HttpServletRequest request){
    request.setAttribute("value", "小明");
    return "success";
}
```

八、springmvc操作session域

8.1 向session域中设置值

```
@Controller
@RequestMapping("/day02")
//只要向request域中放置的key为value，该值就会被同步到session域中
@SessionAttributes("value")
public class Day02Controller {

}
```

8.2 使用ModelMap从session域中获取值

```
@RequestMapping("/getValue")
public String getValue(ModelMap modelMap){
    System.out.println(modelMap.get("value"));
    return "success";
}
```

8.3 清空session域

```
@RequestMapping("/delValue")
public String delValue(SessionStatus sessionStatus){
    //清空session
    sessionStatus.setComplete();
    return "success";
}
```

九、异常处理机制

在Java中，对于异常的处理一般有两种方式:

- 一种是当前方法捕获处理（try-catch），这种处理方式会造成业务代码和异常处理代码的耦合。
- 另一种是自己不处理，而是抛给调用者处理（throws），调用者再抛给它的调用者，也就是一直向上抛。
  在这种方法的基础上，衍生出了SpringMVC的异常处理机制。

由于我们的代码最后都是有Spring框架来调用的，也就是说异常最终会抛到框架，然后由框架进行统一处理。

SpringMVC的异常处理思路:所有异常都向上抛，再后指定一个统一的异常处理器进行处理。

自定义异常处理器:

自定义一个类实现HandlerExceptionResolver，然后将类注册到IOC中就可以了。

```
@Component
public class CommonExceptionResolver implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest httpServletRequest,
                                         HttpServletResponse httpServletResponse,
                                         Object o,Exception e) {
        ModelAndView modelAndView = new ModelAndView();
        //指定错误消息
        modelAndView.addObject("message",e.getMessage());
        //指定错误视图页面
        modelAndView.setViewName("error");
        return modelAndView;
    }
}
```