```
/*
copyright @Qixuan Sun
*/
```

Struts1, 2, Spring区别

- SpringMVC 单例，非线程安全 Struts1 单例，非线程安全 Struts2 线程安全，对每个请求产生一个实例
- SpringMVC的入口是Servlet，Struts2是filter Spring前端总控制器是DispatcherServlet Structs2前端总控制器是FilterDispatcher Structs1前端总控制器是ActionServlet
- Structs是在接受参数时，用属性来接受参数，说明参数是让多个方法共享的 SpringMVC用方法来接受参数
- SpringMVC是基于方法的设计，Struts是基于类
- Spring框架全面支持mvc设计模式。SpringMVC很灵活。 Struts 1的后端控制器侵入性太强，要求必须继承Action或者Action类的某个派生类。从请求(HttpServletRequest)中获取参数也使用了类型（ActionForm），所以使用起来需要记住很多类，需要封装。 Strtus2 的后端控制器尽管比较灵活，但是也需要些繁琐的配置文件。后端控制器到路径的映射也是很麻烦。无论Struts哪个版本，对视图技术（view）的支持都不够强。

Spring Controller返回值

1. ModelAndView
2. Model
3. ModelMap
4. Map: 返回json数据
5. View: 直接返回视图
6. String: 返回String，那么字符串描述的视图的逻辑名字，通过视图解析器解析为物理视图地址。例如 return "index"; return "forward:index"; return "redirect: index";
7. void: 返回void，那么在后端控制器处理函数中，可以直接使用重定向技术，如果没有重定向，那么
8. @ResponseBody Object

HTTP协议(应用层)

七种请求

Http七种请求方法：

1. GET
2. POST
3. PUT
4. DELETE
5. HEAD
6. TRACE

7. OPTIONS

GET/POST

GET-POST区别: (预测考简答)

1. 安全性: GET提交的表单参数显示在URL上, POST不会
2. GET是从服务器上获取数据, POST是向服务器提交数据
3. GET请求会被浏览器主动缓存, POST不会
4. GET请求只能URL编码, POST支持多种编码
5. GET请求的URL有长度限制, POST提交的表单数据没有限制

HTTP状态码

Http响应类型:

1. 404 not found (服务器回复请求内容不存在)
2. 200 正常响应
3. 500 服务器运行过程中遇到内部故障

HTTP缓存

补充Http缓存机制: (考题)

1. 为什么使用Http缓存? 通常情况下通过网络获取内容速度慢成本高，有些响应需要在客户端和服务器之间进行多次往返通信，这就拖延了浏览器可以使用和处理内容的时间，同时也增加了访问者的数据成本。通过缓存，使用资源副本，大大减少获取资源时间，能够减少网络带宽消耗、减少延迟与网络阻塞，同时降低服务器压力，提高服务器性能。
2. 怎么缓存? Http的Message中的header里带有需要访问内容的最后访问时间, 服务器通过客户端的最后访问时间判断内容是否被更新, 若未被更新, 则回复客户端让其使用缓存

Others

include/forward

```
include:
    response.getWriter().println("text");
    RequestDispatcher rd = request.getRequestDispatcher("/b");
    rd.include(request,response);
    response.getWriter().println("over");
    输出结果为 text hello(/b输出的) over
forward:
    把请求转发给其他组件，但是结束后不返回原来组件操作
    rd.forward(request,response);
    之后不应该再有response.getWriter().println();
    但转发请求到那个组件不能用close函数，谁开的谁关
    如果使用forward，不允许在原来转发的函数中输出内容,所以输出的是hello而不是text hello
    会有两句报错:
        response.getWriter().println("text");
```

```
        response.getWriter().println("over");
```

JSP-EL表达式

11个隐含变量除了pageContext 是PageContext类型 其余都是map类型

Filter定义

```
  public void doFilter(ServletRequest req, ServletResponse res, FilterChain
chain) throws ServletException, IOException{
    // ...Servlet预处理...
    chain.doFilter(req, res);// 给后续过滤器响应，如果没有这句话，则不进行后续过滤
    // ...Servlet后处理...
  }
}
```

Tag-doStartTag/doEndTag

```
返回值
int doStartTag();
  static final int SKIP_BODY
  Skip body evaluation. Valid return value for doStartTag and doAfterBody.
  static final int EVAL_BODY_INCLUDE
  Evaluate body into existing out stream. Valid return value for doStartTag.

int doEndTag();
  static final int SKIP_PAGE
  Skip the rest of the page. Valid return value for doEndTag.
  static final int EVAL_PAGE
  Continue evaluating the page. Valid return value for doEndTag().
```

Struts1-ActionDispatcher

```
protected ActionDispatcher dispatcher = new ActionDispatcher(this,
ActionDispatcher.MAPPING_FLAVOR);
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response) throws Exception
{
    return dispatcher.execute(mapping, form, request, response);
}
```

Struts1-Action/DispatchAction

在login.jsp的from中

```
<html:form action="/login.do" method="post" focus="userName">
```

的 "login.do”改成"login.do?method=loginCheck”

对应的action方法中，将execute方法名改成loginCheck。

修改之后，怎么也不走loginCheck这个方法，很是纳闷。后来才发现，原因出在这：

LoginAction extends org.apache.struts.action.Action

LoginAction 继承的类不对，应该继承 org.apache.struts.actions.Dispatchaction.下面就来说说他们的区别：

<action>的parameter属性是给DispatchAction使用的，你的类要继承DispatchAction类，而不是普通的Action，Action只会执行execute方法，DispatchAction会根据parameter的值执行特定的方法，注意parameter的值不要设置为execute，也不要覆盖DispatchAction中的execute（），因为DispatchAction继承于Action，它的execute会首先执行，在execute（）方法中取出parameter的值，通过java反射调用指定的方法。

struct-config.xml 中 action的parameter属性设置成 "method”，这样jsp页面中的"login.do?method=loginCheck”就起作用了

Annotation

```
Annotation Types
    HandlesTypes
    HttpConstraint
    HttpMethodConstraint
    MultipartConfig
    ServletSecurity
    WebFilter
    WebInitParam
    WebListener
    WebServlet

WebServlet
java.lang.String  name
WebInitParam[]  initParams
java.lang.String[]  urlPatterns
E.g. WebServlet(name="TestServlet", urlPatterns={"/test"},initParams=
{&#064;WebInitParam(name="test", value="true")})

WebFilter
java.lang.String  filterName
WebInitParam[]  initParams
java.lang.String[]  servletNames
java.lang.String[]  urlPatterns
E.g. @WebFilter("/path/*")

WebInitParam
```

```
java.lang.String  name
java.lang.String  value

WebListner
java.lang.String  value
```