http://blog.csdn.net/tonytfjing/article/details/39207551

1.DispatcherServlet

SpringMVC具有统一的入口DispatcherServlet，所有的请求都通过DispatcherServlet。

DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据某某规则分发到目标Controller来处理。 所以我们现在web.xml中加入以下配置：

```
<!-- 初始化 DispatcherServlet时，该框架在web应用程序
      WEB-INF目录中寻找一个名为[servlet-名称]-servlet.xml的文件，
         并在那里定义相关的Beans，重写在全局中定义的任何Beans -->
   <servlet>
     <servlet-name>springMybatis</servlet-name>
     <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
     <servlet-name>springMybatis</servlet-name>
     <!-- 所有的的请求，都会被DispatcherServlet处理 -->
     <url-pattern>/</url-pattern>
   </servlet-mapping>
```

2.静态资源不拦截

如果只配置拦截类似于*.do格式的url，则对静态资源的访问是没有问题的，但是如果配置拦截了所有的请求（如我们上面配置的“/”），就会造成js文件、css文件、图片文件等静态资源无法访问。

一般实现拦截器主要是为了权限管理，主要是拦截一些url请求，所以不对静态资源进行拦截。要过滤掉静态资源一般有两种方式，

第一种是采用<mvc:default-servlet-handler />，（一般Web应用服务器默认的Servlet名称是"default"，所以这里我们激活Tomcat的defaultServlet来处理静态文件，在web.xml里配置如下代码即可：）

```
<!--  该servlet为tomcat,jetty等容器提供,将静态资源映射从/改为/static/目录，
如原来访问  http://localhost/foo.css  ,现在
http://localhost/static/foo.css  -->
<!-- 不拦截静态文件 -->
<servlet-mapping>
    <servlet-name>default</servlet-name>
```

```
    <url-pattern>/js/*</url-pattern>
    <url-pattern>/css/*</url-pattern>
    <url-pattern>/images/*</url-pattern>
    <url-pattern>/fonts/*</url-pattern>
</servlet-mapping>
```

Tomcat, Jetty, JBoss, and GlassFish 默认 Servlet的名字 -- "default"
Resin 默认 Servlet的名字 -- "resin-file"
WebLogic 默认 Servlet的名字 -- "FileServlet"
WebSphere 默认 Servlet的名字 -- "SimpleFileServlet"

如果你所有的Web应用服务器的默认Servlet名称不是"default"，则需要通过default-servlet-name属性显示指定：

```
<mvc:default-servlet-handler default-servlet-name="所使用的Web服务器默认使用的Servlet名称" />
```

第二种是采用<mvc:resources />，在springmvc的配置文件中加入以下代码：

```
<mvc:resources mapping="/js/**" location="/static_resources/javascript/"/>
<mvc:resources mapping="/styles/**" location="/static_resources/css/"/>
<mvc:resources mapping="/images/**" location="/static_resources/images/"/>
```

3.自定义拦截器

SpringMVC的拦截器HandlerInterceptorAdapter对应提供了三个preHandle，postHandle，afterCompletion方法。preHandle在业务处理器处理请求之前被调用，

postHandle在业务处理器处理请求执行完成后,生成视图之前执行，afterCompletion在DispatcherServlet完全处理完请求后被调用,可用于清理资源等 。所以要想实现自己的权限管理逻辑，需要继承HandlerInterceptorAdapter并重写其三个方法。

首先在springmvc.xml中加入自己定义的拦截器我的实现逻辑CommonInterceptor，

```
<!--配置拦截器, 多个拦截器,顺序执行 -->
<mvc:interceptors>
    <mvc:interceptor>
        <!-- 匹配的是url路径， 如果不配置或/**,将拦截所有的Controller -->
        <mvc:mapping path="/" />
        <mvc:mapping path="/user/**" />
        <mvc:mapping path="/test/**" />
        <bean class="com.alibaba.interceptor.CommonInterceptor"></bean>
```

```
9.      </mvc:interceptor>
10.     <!-- 当设置多个拦截器时，先按顺序调用preHandle方法，
11.          然后逆序调用每个拦截器的postHandle和afterCompletion方法 -->
12. </mvc:interceptors>
```

我的拦截逻辑是“在未登录前，任何访问url都跳转到login页面；登录成功后跳转至先前的url”，具体代码如下：

```
1. package com.alibaba.interceptor;
2. import javax.servlet.http.HttpServletRequest;
3. import javax.servlet.http.HttpServletResponse;
4. import org.slf4j.Logger;
5. import org.slf4j.LoggerFactory;
6. import org.springframework.web.servlet.ModelAndView;
7. import
org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
8. import com.alibaba.util.RequestUtil;
9. /**
10.  * @author tfj
11.  * 2014-8-1
12.  */
13. public class CommonInterceptor extends HandlerInterceptorAdapter{
14.     private final Logger log =
LoggerFactory.getLogger(CommonInterceptor.class);
15.     public static final String LAST_PAGE = "com.alibaba.lastPage";
16.     /*
17.      * 利用正则映射到需要拦截的路径
18.      */
19.     private String mappingURL;
20.
21.     public void setMappingURL(String mappingURL) {
22.                 this.mappingURL = mappingURL;
23.     }
24.  */
25.     /**
26.      * 在业务处理器处理请求之前被调用
27.      * 如果返回false
28.      *     从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器
链
29.      * 如果返回true
```

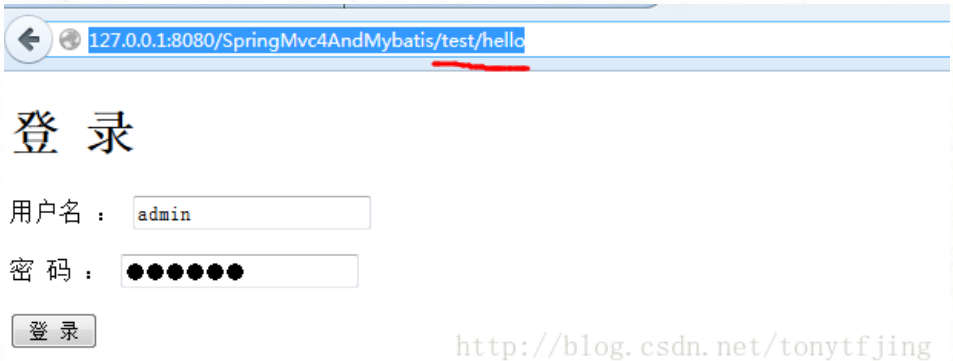
```
30.      *    执行下一个拦截器,直到所有的拦截器都执行完毕
31.      *    再执行被拦截的Controller
32.      *    然后进入拦截器链,
33.      *    从最后一个拦截器往回执行所有的postHandle()
34.      *    接着再从最后一个拦截器往回执行所有的afterCompletion()
35.      */
36.     @Override
37.     public boolean preHandle(HttpServletRequest request,
38.             HttpServletResponse response, Object handler) throws
Exception {
39.         if ("GET".equalsIgnoreCase(request.getMethod())) {
40.             RequestUtil.saveRequest();
41.         }
42.         log.info("==============执行顺序: 1、
preHandle================");
43.         String requestUri = request.getRequestURI();
44.         String contextPath = request.getContextPath();
45.         String url = requestUri.substring(contextPath.length());
46.
47.         log.info("requestUri:"+requestUri);
48.         log.info("contextPath:"+contextPath);
49.         log.info("url:"+url);
50.
51.         String username =
(String)request.getSession().getAttribute("user");
52.         if(username == null){
53.             log.info("Interceptor: 跳转到login页面! ");
54.             request.getRequestDispatcher("/WEB-INF/jsp/login.jsp")
55.                     .forward(request, response);
56.             return false;
57.         }else
58.             return true;
59.     }
60.
61.     /**
62.      * 在业务处理器处理请求执行完成后,生成视图之前执行的动作
63.      * 可在modelAndView中加入数据，比如当前时间
64.      */
65.     @Override
```

```
66.     public void postHandle(HttpServletRequest request,
67.             HttpServletResponse response, Object handler,
68.             ModelAndView modelAndView) throws Exception {
69.         log.info("==============执行顺序: 2、
postHandle================");
70.         if(modelAndView != null){  //加入当前时间
71.             modelAndView.addObject("var", "测试postHandle");
72.         }
73.     }
74.
75.     /**
76.      * 在DispatcherServlet完全处理完请求后被调用,可用于清理资源等
77.      *
78.      * 当有拦截器抛出异常时,会从当前拦截器往回执行所有的拦截器的
afterCompletion()
79.      */
80.     @Override
81.     public void afterCompletion(HttpServletRequest request,
82.             HttpServletResponse response, Object handler, Exception ex)
83.             throws Exception {
84.         log.info("==============执行顺序: 3、
afterCompletion================");
85.     }
86. }
```

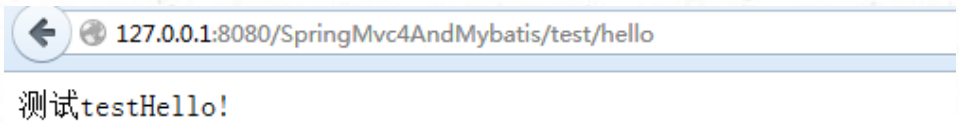
注：上述代码里我写了一个RequestUtil,主要实现获取当前Request、Session对象，保存和加密页面，取出等功能。

至此，拦截器已经实现了，效果如图：

我直接访问/test/hello，会被拦截



登录成功后会跳转至/test/hello对应的页面



SpringMVC拦截器两种方法：

配置文件加<mvc:interceptors>

1. implements HandlerInterceptor
2. extends HandlerInterceptorAdapter

   preHandle (controller之前执行)

   postHandle(controller执行完毕, ModelAndView返回之前执行)

   afterCompletion(最后执行)