1. 拦截器Interceptor 的作用

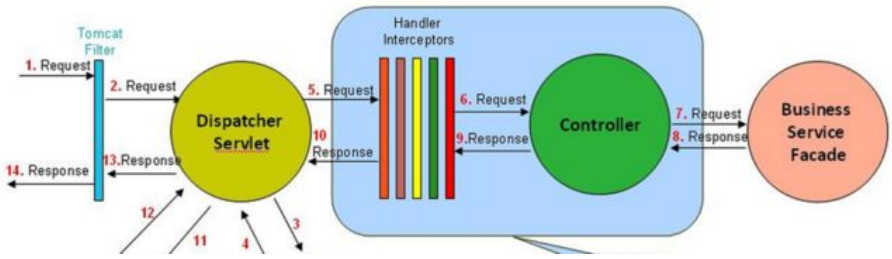
Spring MVC 的处理器拦截器类似于 Servlet 开发中的过滤器 Filter，用于对处理器进行预处理和后处理。用户可以自己定义一些拦截器来实现特定的功能。谈到拦截 器，还要向大家提一个词------拦截器链（Interceptor Chain）。拦截器链就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。

说到这里，可能大家脑海中有了一个疑问，这不是我们之前学的过滤器吗？是的它和过滤器是有几分相似，但是也有区别，接下来我们就来说说他们的区别：

过滤器是 servlet 规范中的一部分，任何 java web 工程都可以使用。 拦截器是 SpringMVC 框架自己的，只有使用了 SpringMVC 框架的工程才能用。

过滤器在 url- pattern 中配置了/*之后，可以对所有要访问的资源拦截。
拦截器它是只会拦截访问的控制器方法，如果访问的是 jsp，html,css,image 或者 js 是不会进行拦截的。它也是 AOP 思想的具体应用。

我们要想自定义拦截器， 要求必须实现：`HandlerInterceptor`接口。



2. 自定义拦截器的步骤

2.1 第一步：编写一个普通类实现 HandlerInterceptor 接口

```
package com.bruceliu.web.interceptor;
1  /**
2   * @author bruceliu
3   * @create 2019-07-20 23:26
4   * @description
5   */
6  public class HandlerInterceptorDemo1 implements
HandlerInterceptor {
7
8      @Override
9      public boolean preHandle(HttpServletRequest
request,HttpServletResponse
```

```
10              response, Object handler)
11              throws Exception {
12          System.out.println("preHandle 拦截器拦截了");
13          return true;
14      }
15
16      @Override
17      public void postHandle(HttpServletRequest request,
HttpServletResponse response,
18                             Object handler,
19                             ModelAndView modelAndView)
throws Exception {
20          System.out.println("postHandle 方法执行了");
21      }
22
23      @Override
24      public void afterCompletion(HttpServletRequest
request, HttpServletResponse
25              response, Object handler, Exception ex)
26              throws Exception {
27          System.out.println("afterCompletion 方法执行
了");
28      }
29  }
```

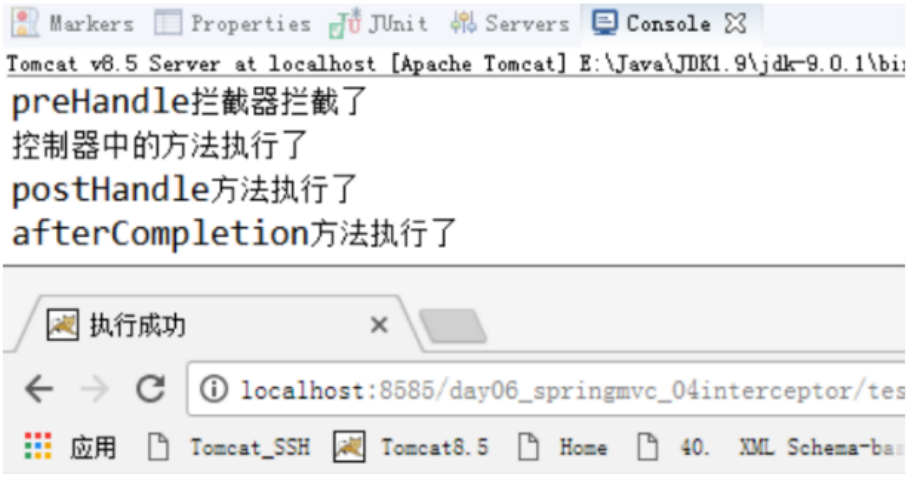
2.2 第二步：配置拦截器

```
1  <!-- 配置拦截器     -->
2  <mvc:interceptors>
3      <mvc:interceptor>
4      <mvc:mapping path="/**"/>
5          <bean id="handlerInterceptorDemo1"

class="com.bruceliu.web.interceptor.HandlerInterceptorDemo
1"></bean>
7      </mvc:interceptor>
8  </mvc:interceptors>
```

~~2.3 控制器略~~

2.3 测试运行结果



3. 拦截器的细节

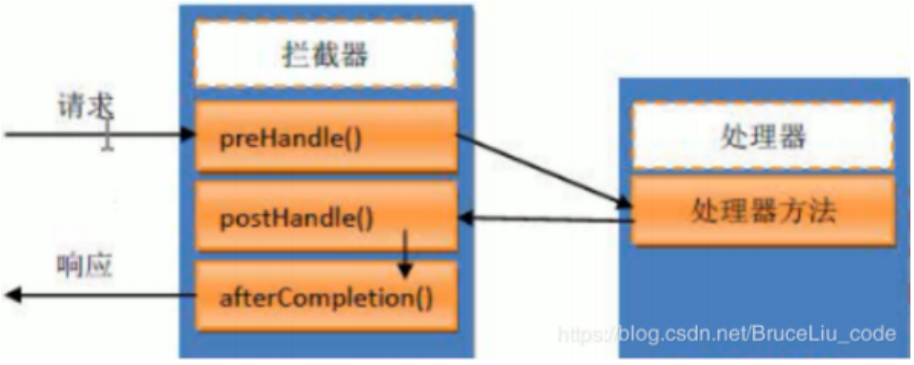
3.1 拦截器的放行

放行的含义是指，如果有下一个拦截器就执行下一个，如果该拦截器处于拦截器链的最后一个，则执行控制器中的方法。

```
@Override
public boolean preHandle(HttpServletRequest request, HttpServlet
        throws Exception {
    System.out.println("preHandle拦截器拦截了");
    return false;
}
```

只有当此方法返回true的时候，程序才能继续执行

3.2 拦截器中方法的说明




```
1  public interface HandlerInterceptor {
2      /**
3       * 如何调用：
4       * 按拦截器定义顺序调用
5       * 何时调用：
6       * 只要配置了都会调用
7       * 有什么用：
8       * 如果程序员决定该拦截器对请求进行拦截处理后还要调用其他的拦
截器，或者是业务处理器去
```

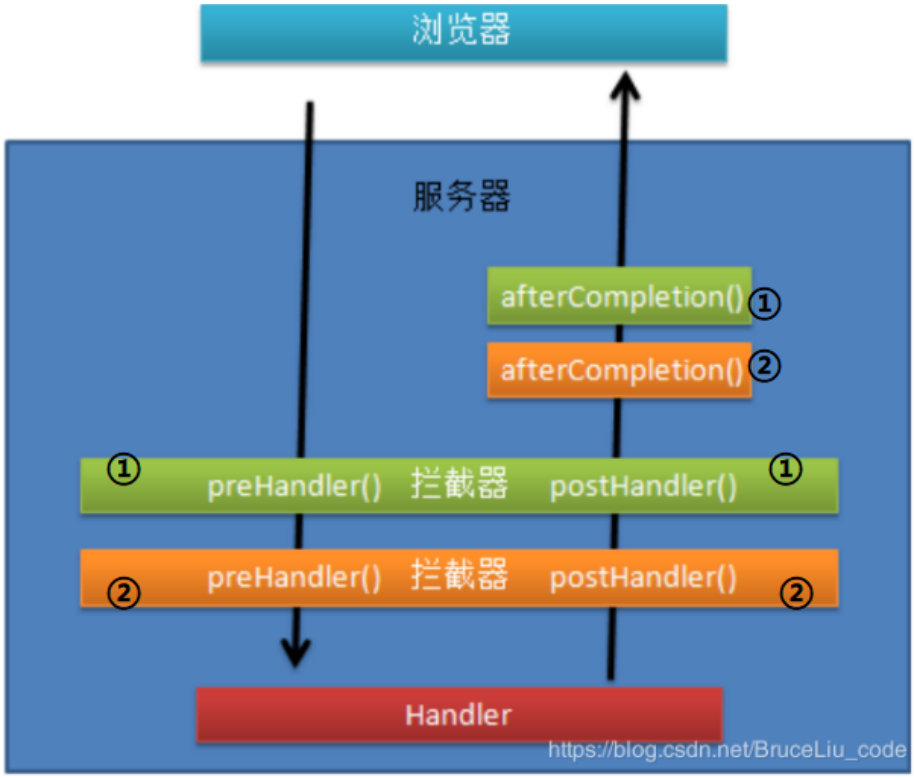
```
9       * 进行处理，则返回 true。如果程序员决定不需要再调用其他的组
件（拦截器、业务处理器）去处理请求，则返回false。*/
10      default boolean preHandle(HttpServletRequest
request,
HttpServletResponse
11      response, Object handler)
12      throws Exception {
13          return true;
14      }
15
16      /**
17      * 如何调用：
18      * 按拦截器定义逆序调用
19      * 何时调用：
20      * 在拦截器链内所有拦截器返成功调用
21      * 有什么用：
22      * 在业务处理器处理完请求后，但是     DispatcherServlet
向客户端返回响应前
被调用，
23      * 在该方法中对用户请求 request 进行处理。
24      */
25      default void postHandle(HttpServletRequest
request, HttpServletResponse
26          response, Object handler, @Nullable
ModelAndView modelAndView) throws Exception
28      {}
29
30
31      /**
32      * 如何调用：
33      * 按拦截器定义逆序调用
34      * 何时调用：
35      * 只有     preHandle 返回     true 才调用
36      * 有什么用：
        * 在DispatcherServlet完全处理完请求后被调用，
37      * 可以在该方法中进行一些资源清理的操作。
38      */
39      default void afterCompletion(HttpServletRequest
request, HttpServletResponse
40      response, Object handler,@Nullable Exception ex)
throws Exception {
42      }
43  }
```

3.3 拦截器的作用路径

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <!-- 用于指定对拦截的    url -->
        <mvc:exclude-mapping path=""/>
        <!-- 用于指定排除的    url-->
        <bean id="handlerInterceptorDemo1"
class="com.bruceliu.web.interceptor.HandlerInterceptorDemo
1">
        </bean>
    </mvc:interceptor>
</mvc:interceptors>
```

3.4 多个拦截器的执行顺序

多个拦截器是按照配置的顺序决定的。



4. 拦截器的简单案例（验证用户是否登录）

4.1 实现思路

1、有一个登录页面，需要写一个 controller 访问页面

2、登录页面有一提交表单的动作。需要在 controller 中处理。

2.1、判断用户名密码是否正确

2.2、如果正确 向 session 中写入用户信息

2.3、返回登录成功。

3、拦截用户请求，判断用户是否登录

3.1、如果用户已经登录。重定向 到main.jsp页面

3.2、如果用户未登录，跳转到登录页面

4.2 控制器代码

```
1  /**
2   * @author bruceliu
3   * @create 2019-07-20 23:39
4   * @description
5   */
   @Controlelr
6  public class Demo1 {
7
8      //跳转到登陆页面
9    @RequestMapping("/login")
10      public String login(Model model) throws Exception
{
11          return "login";
12      }
13
14      //处理登陆提交请求
15      //userid：用户账号，pwd：密码
16      @RequestMapping("/loginsubmit")
17      public String loginsubmit(HttpSession session,
String userid, String pwd) throws
18              Exception {
19          //向 session 记录用户身份信息
            if(userid=="正确" && pwd == "123" )
20          {
                session.setAttribute("activeUser",
userid);
                return "redirect:/main.jsp";
            }
            else return "login";
21
22      }
23
24      //退出
25      @RequestMapping("/logout")
26      public String logout(HttpSession session) throws
Exception {
27          //session 过期
28          session.invalidate();
29          return "redirect:index.jsp";
30      }
31  }
```

4.3 拦截器代码

```
1  /**
2   * @author bruceliu
3   * @create 2019-07-20 23:41
4   * @description
5   */
6  public class LoginInterceptor implements
HandlerInterceptor {
7
8      @Override
9      public boolean preHandle(HttpServletRequest
request,
10                     HttpServletResponse response, Object
handler) throws Exception {
11          //如果是登录页面则放行,例如如果用户在浏览器中输入的是
main.jsp页面的请求，那么就会被拦截
            //器拦截到，在这个方法中直接返回false进行拦截。
12          if
(request.getRequestURI().indexOf("login.action") >= 0)
            {
13              return true;
14          }

15          HttpSession session = request.getSession();
16          //如果用户已登录也放行，如果用户在浏览器中请求的是登陆
页面，那么就不会被拦截，否则会被拦截
17          if (session.getAttribute("user") != null) {
18              return true;
19          }
20          //用户没有登录就直接访问需要登录才能看到的页面，那么就
会跳转到登录页面
21          request.getRequestDispatcher("/WEB-
INF/jsp/login.jsp")
                        .forward(request,response);
22          return false;
23      }
24
25      @Override
26      public void postHandle(HttpServletRequest request
                               ,HttpServletResponse
response, Object handler
                               , ModelAndView
modelAndView) throws Exception {
27
28      }
29
30      @Override
```

```
31    public void afterCompletion(HttpServletRequest
request
                                    , HttpServletResponse
response, Object handler
                                    , Exception ex)throws
Exception {
32
33      }
34  }
```