**Filter过滤器API**  
     Servlet过滤器API包含了3个接口，它们都在javax.servlet包中，分别是Filter接口、FilterChain接口和FilterConfig接口。  
**Filter接口（源码）**

public interface Filter {

public void init(FilterConfig filterConfig) throws ServletException;

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException;

public void destroy();

}

所有的过滤器都必须实现Filter接口。该接口定义了init,doFilter0，destory()三个方法：  
  (1)init(FilterConfig filterConfig)  
      在web应用程序启动时，web服务器将根据 web.xml文件中的配置信息来创建每个注册的Filter实例对象，并将其保存在服务器的内存中。Web容器创建Filter对象实例后，将立即调用该Filter对象的init方法。Init方法在Filter生命周期中仅执行一次，web容器在调用init方法时，会传递一个包含Filter的配置和运行环境的FilterConfig对象(FilterConfig的用法和ServletConfig类似)。利用FilterConfig对象可以得到ServletContext对象，以及部署描述符中配置的过滤器的初始化参数。  
  (2)doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

      doFilter()方法类似于Servlet接口的service()方法。当客户端请求目标资源的时候，容器就会调用与这个目标资源相关联的过滤器的 doFilter()方法。其中参数 request, response 为 web 容器或 Filter 链的上一个 Filter 传递过来的请求和相应对象；参数 chain 为代表当前 Filter 链的对象，在特定的操作完成后，可以在当前 Filter 对象的 doFilter 方法内部需要调用 FilterChain 对象的 chain.doFilter(request,response)方法才能把请求交付给 Filter 链中的下一个 Filter 或者目标 Servlet 程序去处理，也可以直接向客户端返回响应信息，或者利用RequestDispatcher的forward()和include()方法，以及 HttpServletResponse的sendRedirect()方法将请求转向到其他资源。这个方法的请求和响应参数的类型是 ServletRequest和ServletResponse，也就是说，过滤器的使用并不依赖于具体的协议。  
 (3)public void destroy()  
       在Web容器卸载 Filter 对象之前被调用。该方法在Filter的生命周期中仅执行一次。在这个方法中，可以释放过滤器使用的资源。  
**FilterChain接口（源码）**

public interface FilterChain {

public void doFilter(ServletRequest request, ServletResponse response) throws IOException, ServletException;

}

(1)doFilter(ServletRequest request，ServletResponse response)  
      此方法是由Servlet容器提供给开发者的，用于对资源请求过滤链的依次调用，通过FilterChain调用过滤链中的下一个过滤  器，如果是最后一个过滤器，则下一个就调用目标资源。  
**FilterConfig接口（源码）  FilterConfig接口检索过滤器名、初始化参数以及活动的Servlet上下文。**

public interface FilterConfig {

//返回web．xml部署文件中定义的该过滤器的名称

public String getFilterName();

//返回调用者所处的servlet上下文

public ServletContext getServletContext();

//返回过滤器初始化参数值的字符串形式，当参数不存在时，返回nul1．name是初始化参数名

public String getInitParameter(String name);

//以Enumeration形式返回过滤器所有初始化参数值，如果没有初始化参数，返回为空

public Enumeration getInitParameterNames();

}

了解了Filter的基本概念和源码，下面具体使用下Filter过滤器来实现登录过滤。

**需求：访问A页面(登录后才能访问的页面)-->未登录-->跳转到登录页面-->登陆成功后，跳转到A页面**

**自定义HttpFilter**

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* HttpFilter

\*/

public abstract class HttpFilter implements Filter{

//保存filterConfig对象

private FilterConfig filterConfig;

/\*\*

\* 直接返回filterConfig对象

\* @return

\*/

public FilterConfig getFilterConfig() {

return filterConfig;

}

/\*\*

\* 不建议子类直接覆盖，若直接失败，将可能导致filterConfig成员变量初始化失败

\*/

@Override

public void init(FilterConfig filterConfig) throws ServletException {

this.filterConfig = filterConfig;

init();

}

/\*\*

\* 供子类继承的初始化方法，刻通过getFilterConfig()方法获得filterConfig对象

\*/

private void init() {}

/\*\*

\* 原生的doFilter方法，在方法内部把ServletRequest和ServletResponse转化化为了HttpServletRequest和HttpServletResponse，

\* 并调用了doFilter(HttpServletRequest request, HttpServletResponse response,FilterChain filterChain)方法

\*/

@Override

public void doFilter(ServletRequest req, ServletResponse res,

FilterChain filterChain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

doFilter(request, response, filterChain);

}

/\*\*

\* 抽象方法，为http请求定制，必须实现的方法

\* @param request

\* @param response

\* @param filterChain

\* @throws IOException

\* @throws ServletException

\*/

public abstract void doFilter(HttpServletRequest request, HttpServletResponse response,

FilterChain filterChain) throws IOException, ServletException;

@Override

public void destroy() {}

}

**web.xml配置CommonFilter**

<filter>

<filter-name>commonFilter</filter-name>

<filter-class>com.gcx.emall.Filter.CommonFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>commonFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

**登录过滤器CommonFilter**

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class CommonFilter extends HttpFilter {

private final Logger log = LoggerFactory.getLogger(CommonFilter.class);

@Override

public void doFilter(HttpServletRequest request,

HttpServletResponse response, FilterChain filterChain)

throws IOException, ServletException {

log.info("==============拦截get请求================");

if ("GET".equalsIgnoreCase(request.getMethod())) {

RequestUtil.saveRequest(request);

}

String requestUri = request.getRequestURI();

String contextPath = request.getContextPath();

String url = requestUri.substring(contextPath.length());

if ("/login".equals(url)) {

filterChain.doFilter(request, response);

return;

} else {

String username = (String) request.getSession().getAttribute("user");

if (username == null) {

log.info("被拦截：跳转到login页面！");

request.getRequestDispatcher("/page/index1.jsp").forward(request, response);

} else

filterChain.doFilter(request, response);

}

}

}

**RequestUtil 保存、获取request并加密请求页面**

public class RequestUtil {

private static final Logger logger = LoggerFactory.getLogger(RequestUtil.class);

private static final Base64 base64 = new Base64(true);

public static final String LAST\_PAGE = "lastPage";//未登录时访问的页面

public static final String REDIRECT\_HOME = "/";//未登录时跳转到首页

public static final String LOGIN\_HOME = "/index.jsp";//登录成功后进入的页面

/\*\*

\* 保存当前请求

\*/

public static void saveRequest(HttpServletRequest request) {

request.getSession().setAttribute(LAST\_PAGE, RequestUtil.hashRequestPage(request));

logger.debug("被拦截的url的sessionID:{}", request.getSession().getId());

logger.debug("save request for {}", request.getRequestURI());

}

/\*\*

\* 加密请求页面

\* @param request

\* @return

\*/

public static String hashRequestPage(HttpServletRequest request) {

String reqUri = request.getRequestURI();

String query = request.getQueryString();

if (query != null) {

reqUri += "?" + query;

}

String targetPage = null;

try {

targetPage = base64.encodeAsString(reqUri.getBytes("UTF-8"));

} catch (UnsupportedEncodingException ex) {

//this does not happen

}

return targetPage;

}

/\*\*

\* 取出之前保存的请求

\* @return

\*/

public static String retrieveSavedRequest(HttpServletRequest request) {

HttpSession session = request.getSession();

if (session == null) {

return REDIRECT\_HOME;

}

String HashedlastPage = (String) session.getAttribute(LAST\_PAGE);

if (HashedlastPage == null) {

return LOGIN\_HOME;

} else {

return retrieve(HashedlastPage);

}

}

/\*\*

\* 解密请求的页面

\* @param targetPage

\* @return

\*/

public static String retrieve(String targetPage) {

byte[] decode = base64.decode(targetPage);

try {

String requestUri = new String(decode, "UTF-8");

int i = requestUri.indexOf("/", 1);

return requestUri.substring(i);

} catch (UnsupportedEncodingException ex) {

//this does not happen

return null;

}

}

}

**LoginCOntroller**

  @RequestMapping(value = "/hello",method = RequestMethod.GET)  
     public String testHello( String test) {  
         log.info("执行了Hello方法！");  
         return "loginSuccess";  
     }   
 @RequestMapping(value = "/login",method = RequestMethod.POST)

public String login(HttpServletRequest request,String userName,String password){

log.info("执行了login方法！");

password = DigestUtils.md5Hex(password);

User user = userService.findUser(userName,password);

if(user!=null){

request.getSession().setAttribute("userId", user.getId());

request.getSession().setAttribute("user", userName);

return "redirect:" + RequestUtil.retrieveSavedRequest(request);//跳转至访问页面

}else{

log.info("用户不存在");

request.getSession().setAttribute("message", "用户名不存在，请重新登录");

return "index";

}

}

最后需要几个jsp页面login.jsp,index.jsp(首页面，任何人都能访问的),loginSuccess.jsp,还需要在controller中加上一个测试testHello方法用于满足之前说的需求。

注意事项：我们过滤的是所有请求，但对于静态资源css,js,image我们应该不拦截，对其放行。我们可以在web.xml中进行指定

<!-- 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>/image/\*</url-pattern>

<url-pattern>/fonts/\*</url-pattern>

</servlet-mapping>

写在后面：本来想把Filter和SpringMVC的interceptor拦截器一起写总结了，但感觉篇幅有些长打算下篇在介绍。