**1. Spring 拦截器**

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

方法：

第1步：继承HandlerInterceptorAdapter类

第2步：重写preHandler方法和postHandler方法和afterCompletion方法。

**项目中使用拦截器，配合Java注解，实现用户行为跟踪**

**postHandle方法里面关键代码：**

1. **if** (handler **instanceof** HandlerMethod) {
2. HandlerMethod method = (HandlerMethod) handler;
3. Desc desc = method.getMethodAnnotation(Desc.**class**);   //获取方法上的注解
4. **if**(desc!= **null** && StringUtils.isNotEmpty(desc.value()) && StringUtil.isNotEmpty(desc.descType())) {
5. uriDesc = desc.value();
6. descType = desc.descType();
7. }
8. }
9. record(request, uri, uriDesc, descType, source);          //用户行为记录到数据库（访问过哪些url，行为描述等）

**2. Spring 过滤器**

## 2.1 Spring Web过滤器

<http://qq-22530757.iteye.com/blog/2177513>

<http://blog.csdn.net/liangxw1/article/details/51095484>

## 2.2 自定义过滤器

<https://www.cnblogs.com/lr393993507/p/5543025.html>

**创建一个Filter 只需两个步骤:**

(1)创建Filter 处理类

(2)在web.xml 文件中配置Filter

创建Filter 必须实现javax.servlet.Filter 接口，在该接口中定义了三个方法。   
• void init(FilterConfig config): 用于完成Filter 的初始化。   
• void destroy(): 用于Filter 销毁前，完成某些资源的回收。   
• void doFilter(ServletRequest request, ServletResponse response,FilterChain chain): 实现过滤功能，该方法就是对每个请求及响应增加的额外处理。

过滤器Filter也具有生命周期：init()->doFilter()->destroy()，由部署文件中的filter元素驱动。

下面是一个自定义过滤器的demo

1、创建Filter处理类

1. **public** **class** XssFilter **implements** Filter {
3. @Override
4. **public** **void** init(FilterConfig config) **throws** ServletException {
5. }
7. **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
8. **throws** IOException, ServletException {
9. XssHttpServletRequestWrapper xssRequest = **new** XssHttpServletRequestWrapper(
10. (HttpServletRequest) request);
11. chain.doFilter(xssRequest, response);
12. }
14. @Override
15. **public** **void** destroy() {
16. }
18. }

2、配置web.xml

1. <filter>
2. <filter-name>xssFilter</filter-name>
3. <filter-**class**>io.renren.common.xss.XssFilter</filter-**class**>
4. </filter>
6. <filter-mapping>
7. <filter-name>xssFilter</filter-name>
8. <url-pattern>/\*</url-pattern>
9. </filter-mapping>

**3. Spring Aop**

**4. 缓存（Redis）**

**5. 全局错误捕获**

<http://blog.csdn.net/eson_15/article/details/51731567>

下面是一个全局捕获异常的demo：继承HandlerExceptionResolver接口，重写resolveException方法。

1. **public** **class** RRExceptionHandler **implements** HandlerExceptionResolver {
2. **private** Logger logger = LoggerFactory.getLogger(getClass());
4. @Autowired
5. **private** EmailCacheHandler emailCacheHandler;
7. @Override
8. **public** ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object o, Exception e) {
9. String uri = request.getRequestURI();
10. logger.error("ajax request error, uri : " + uri + ", message : " + e.getMessage());
11. //打印异常栈信息
12. e.printStackTrace();
13. /\*\*\*\*\*\*\*\*写入redis队列\*\*\*\*\*\*\*\*\*\*/
14. emailCacheHandler.save(e);   //加入到redis，以便后期处理
15. **if** (StringUtils.isEmpty(request.getHeader("X-Requested-With"))) {
16. **try**{
17. response.getWriter().print(e.getMessage());
18. **return** **new** ModelAndView();
19. }
20. **catch** (Exception ex) {
21. logger.error("RRExceptionHandler====异常处理失败", e);
22. }
23. }
25. R r = **new** R();
26. response.setContentType("application/json; charset=utf-8");
27. response.setCharacterEncoding("UTF-8");
28. **if** (e **instanceof** RRException) {
29. r.put("code", ((RRException) e).getCode());
30. r.put("message", ((RRException) e).getMessage());
31. } **else** **if** (e **instanceof** AuthenticationException) {
32. r = R.error("没有权限，请联系管理员授权");
33. }  **else** {
34. r = R.error();
35. }
37. logger.error(e.getMessage(), e);
38. **try** {
39. String json = JacksonUtil.toJson(r);
40. response.getWriter().print(json);
41. } **catch** (Exception ex) {
42. logger.error("RRExceptionHandler====异常处理失败", e);
43. }
45. **return** **new** ModelAndView();
46. }
47. }

**6. 发邮件**

**7. 发短息**

**8. 定时任务**

**9. Spring 实现多数据库及读写分离方法**

**10. Druid连接池实现web监控**

第1步：pom文件

1. **<dependency>**
2. **<groupId>**com.alibaba**</groupId>**
3. **<artifactId>**druid**</artifactId>**
4. **<version>**${druid-version}**</version>**
5. **</dependency>**

第2步：打开Druid的监控统计功能：

Druid的监控统计功能是通过filter-chain扩展实现，如果你要打开监控统计功能，需要配置StatFilter

1. <!—自己定制的StatLogger、可以将druid记录到文件和日志打印等操作--**>**
2. **<bean** id="statLogger" class="com.lizhihui.druid.StatLogger"**></bean>**
3. **<bean** id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
4. init-method="init" destroy-method="close"**>**
5. **<property** name="url"**>**
6. **<value>**${jdbc.url}**</value>**
7. **</property>**
8. **<property** name="username"**>**
9. **<value>**${jdbc.username}**</value>**
10. **</property>**
11. **<property** name="password"**>**
12. **<value>**${jdbc.password}**</value>**
13. **</property>**
14. **<property** name="initialSize"**>**
15. **<value>**${jdbc.initialSize}**</value>**
16. **</property>**
17. **<property** name="maxActive"**>**
18. **<value>**${jdbc.maxActive}**</value>**
19. **</property>**
21. <!—关键：  监控统计拦截的filters --**>**
22. **<property** name="proxyFilters"**>**
23. **<list>**
24. **<ref** bean="stat-filter" **/>**
25. **<ref** bean="wall-filter" **/>**
26. **</list>**
27. **</property>**
28. **</bean>**
30. **<bean** id="stat-filter" class="com.alibaba.druid.filter.stat.StatFilter"**>**
31. **<property** name="slowSqlMillis" value="1000" **/>**
32. **<property** name="logSlowSql" value="true" **/>**  <!-- 慢SQL记录-->
33. **<property** name="mergeSql" value="true" **/>**
34. **</bean>**
36. **<bean** id="wall-filter" class="com.alibaba.druid.wall.WallFilter"**>**
37. **<property** name="dbType" value="mysql" **/>**
38. **<property** name="config" ref="wall-filter-config" **/>**
39. **</bean>**
41. **<bean** id="wall-filter-config" class="com.alibaba.druid.wall.WallConfig"**>**
42. **<property** name="multiStatementAllow" value="true" **/>**
43. **</bean>**
45. <!-- 配置druid监控spring jdbc -->
46. **<bean** id="druid-stat-interceptor" class="com.alibaba.druid.support.spring.stat.DruidStatInterceptor"**></bean>**
47. **<bean** id="druid-stat-pointcut" class="org.springframework.aop.support.JdkRegexpMethodPointcut" scope="prototype"**>**
48. **<property** name="patterns"**>**
49. **<list>**
50. **<value>**io.renren.modules.\*.controller.\***</value>**
51. **<value>**io.renren.modules.\*.service.\***</value>**
52. **</list>**
53. **</property>**
54. **</bean>**
55. **<aop:config>**
56. **<aop:advisor** advice-ref="druid-stat-interceptor" pointcut-ref="druid-stat-pointcut"**/>**
57. **</aop:config>**
59. **<bean** id="transactionManager"
60. class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
61. **<property** name="dataSource"**>**
62. **<ref** bean="dataSource" **/>**
63. **</property>**
64. **</bean>**
66. **<bean** id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>**
67. **<property** name="configLocation" value="classpath:mybatis.xml" **/>**
68. **<property** name="mapperLocations" value="classpath:mapper/\*\*/\*.xml" **/>**
69. **<property** name="dataSource" ref="dataSource" **/>**
70. **</bean>**
72. **<bean** class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>**
73. **<property** name="basePackage" value="io.renren.modules.\*.dao" **/>**
74. **</bean>**

第3步：在web.xml文件后，添加

1. **<servlet>**
2. **<servlet-name>**DruidStatView**</servlet-name>**
3. **<servlet-class>**com.alibaba.druid.support.http.StatViewServlet**</servlet-class>**
4. **</servlet>**
6. **<servlet-mapping>**
7. **<servlet-name>**DruidStatView**</servlet-name>**
8. **<url-pattern>**/druid/\***</url-pattern>**
9. **</servlet-mapping>**

之后，就可以打开 <http://10.206.2.248:8082/druid/sql.html> 看到数据库各种信息了

此处还可以配置登录账户和密码，如果不配置，则无账户密码登录监控页面，如下：

1. **<servlet>**
2. **<servlet-name>**DruidStatView**</servlet-name>**
3. **<servlet-class>**com.alibaba.druid.support.http.StatViewServlet**</servlet-class>**
4. <!—
5. deny优先于allow，如果在deny列表中，就算在allow列表中，也会被拒绝。
6. 如果allow没有配置或者为空，则允许所有访问
7. --**>**
8. **<init-param>**
9. **<param-name>**allow**</param-name>**
10. **<param-value>**128.242.127.1/24,127.0.0.1**</param-value>**
11. **</init-param>**
12. **<init-param>**
13. **<param-name>**deny**</param-name>**
14. **<param-value>**192.168.1.118**</param-value>**
15. **</init-param>**
16. <!-- 在StatViewSerlvet输出的html页面中，有一个功能是Reset All，执行这个操作之后，会导致所有计数器清零，重 新计数 -->
17. **<init-param>**
18. **<param-name>**resetEnable**</param-name>**
19. **<param-value>**false**</param-value>**
20. **</init-param>**
22. **<init-param>**
23. **<param-name>**loginUsername**</param-name>**
24. **<param-value>**druid**</param-value>**
25. **</init-param>**
26. **<init-param>**
27. **<param-name>**loginPassword**</param-name>**
28. **<param-value>**druid**</param-value>**
29. **</init-param>**
30. **</servlet>**
32. **<servlet-mapping>**
33. **<servlet-name>**DruidStatView**</servlet-name>**
34. **<url-pattern>**/druid/\***</url-pattern>**
35. **</servlet-mapping>**

第4步：可以自定义类，实现数据库信息log记录，只要在第1步的配置：

1. **<bean** id="statLogger" class="com.lizhihui.druid.StatLogger"**></bean>**

然后编写这个类的代码

1. /\*\*
2. \* druid监控sql信息(activeCount)日志打印
3. \* Created by lizhihui on 2017/10/17.
4. \*/
5. **public** **class** StatLogger **extends** DruidDataSourceStatLoggerAdapter {
6. **private** **static** Log         LOG    = LogFactory.getLog(StatLogger.**class**);
7. **private** Log                logger = LOG;
9. @Setter
10. @Getter
11. **private** Map<String, Object> map   = **new** LinkedHashMap<>();
13. **public** StatLogger(){
14. **this**.configFromProperties(System.getProperties());
15. }
17. /\*\*
18. \* @since 0.2.21
19. \*/
20. @Override
21. **public** **void** configFromProperties(Properties properties) {
22. String property = properties.getProperty("druid.stat.loggerName");
23. **if** (property != **null** && property.length() > 0) {
24. setLoggerName(property);
25. }
26. }
28. **public** Log getLogger() {
29. **return** logger;
30. }
32. @Override
33. **public** **void** setLoggerName(String loggerName) {
34. logger = LogFactory.getLog(loggerName);
35. }
37. @Override
38. **public** **void** setLogger(Log logger) {
39. **if** (logger == **null**) {
40. **throw** **new** IllegalArgumentException("logger can not be null");
41. }
42. **this**.logger = logger;
43. }
45. **public** **boolean** isLogEnable() {
46. **return** logger.isInfoEnabled();
47. }
49. **public** **void** log(String value) {
50. logger.info(value);
51. }
53. @Override
54. **public** **void** log(DruidDataSourceStatValue statValue) {
55. **if** (!isLogEnable()) {
56. **return**;
57. }
58. **int** activeCount = statValue.getActiveCount();
59. map.put("activeCount", activeCount);
60. String maxActive = PropertyUtils.getProperty("hibernate.properties", "jdbc.max\_size");
61. **if** (StringUtil.isNotEmpty(maxActive) && (activeCount == Integer.valueOf(maxActive))) {
62. String text = JSONUtils.toJSONString(map);
63. log(text);
64. }
65. }
66. }

**11. MyBatis**

**12. Spirng多环境打包方法**

**13. 权限管理（shrio）**

**14. Http协议**

**15. Linux基本使用**

**16. 项目发布脚本**

**17. Mysql知识及Mysql压测方法**

**18. poi导出百万数据的方法**

**19. spring mvc中常用的封装类**

**19.1 BeanUtils**

**20.如何区分ajax请求**

<http://blog.csdn.net/javajiawei/article/details/50563154>

1、以X打头的头域作为非HTTP标准协议，一般是某种技术的出现而产生或者某个组织指定的， X-Requested-With头域就是用来判断一个请求是传统的HTTP请求，还是Ajax请求。也就是说Ajax的请求一般都会带上X-Requested-With头域。

2、可以利用下面的代码区分是否是ajax请求：

1. **if**(request.getHeader("X-Requested-With") != **null**
2. &&request.getHeader("X-Requested-With").equalsIgnoreCase("XMLHttpRequest")){
3. out.print("该请求是 AJAX 异步HTTP请求。");
4. } **else** {
5. out.print("该请求是传统的 同步HTTP请求。");
6. }

# 21. Spring MVC请求处理流程（待完善）

<https://www.cnblogs.com/dreamworlds/p/5396112.html>

<https://www.cnblogs.com/hujiapeng/p/5765636.html>