# AspectJ实现对某些功能PV统计

# 1.场景

在项目中，需要对某些页面进行PV统计，那么这些需求一般在项目初期可能不会出现。

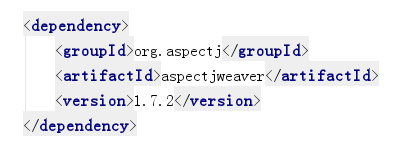
如果项目基本开发完成，或者在迭代的过程中需要增加PV等统计功能，那么，最原始的做法是在原有的代码基础上增加新代码。这样一来，会使工作量增大，维护难度增加。

# 2.思路

可以通过Aop的方式 对原有功能进行增强，不对原有程序进行修改，从而完成PV统计的实现，在Spring中，可以借助AspectJ来实现该需求

# 3实现

在pom.xml中增加 aspectJ的支持



当然，还要有spring-aop的支持

拿我们的项目来说，需要统计，首页，商品，机构，资讯等页面PV

为了方便拓展，将需要统计的类型 做成一个枚举类

*/\*\*  
 \* 统计PV类型枚举  
 \*/***public enum** PvTypeEnum {  
  
 ***DEFAULT***(**"未知"**),  
 ***HOME***(**"首页"**),  
 ***GOODS\_LIST***(**"商品列表"**),  
 ***GOODS\_DETAIL***(**"商品详情"**),  
 ***CAMPUS\_LIST***(**"机构列表"**),  
 ***CAMPUS\_DETAIL***(**"机构详情"**),  
 ***INFORMATION\_LIST***(**"资讯列表"**),  
 ***INFORMATION\_DETAIL***(**"资讯详情"**),  
 ***USERINFO***(**"我的"**);  
  
 **private** String **value**;  
  
 **public** String getValue() {  
 **return value**;  
 }  
  
 PvTypeEnum(String value) {  
 **this**.**value** = value;  
 }  
}

Aop可以设置切点，实现功能的增强，比如在某个方法前执行，方法后，抛出异常时，或者环绕通知，相关内容不做过多解析，可以详细了解AspcetJ

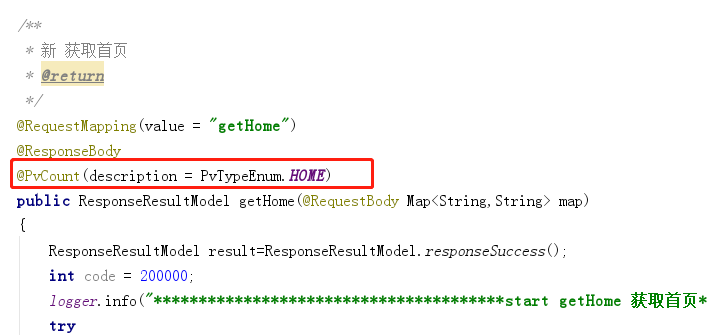
可以通过设置，拦截某一个类中的某些方法，或者某个包下的某些方法，表达式与Spring声名式事物的表达式一致

还有一种方法，可以拦截带有特定注解的方法。完成功能的增强。

本次采用的就是通过注解拦截的方式，首先定义一个 自定义注解

*/\*\*  
 \* 统计PV用注解  
 \*/*@Target({ElementType.***METHOD***})  
@Retention(RetentionPolicy.***RUNTIME***)  
@Documented  
@Inherited  
**public** @**interface** PvCount {  
  
 PvTypeEnum description() **default** PvTypeEnum.***DEFAULT***;  
  
}

在需要统计的功能上增加注解



当然也可以选择放在service层的方法中

编写用于处理逻辑的AOP类

*/\*\*  
 \* PV日志统计  
 \*/*@Component  
@Aspect  
**public class** PvAopLog {  
  
 @Autowired  
 **private** TokenUtil **tokenUtil**;  
  
 @Autowired  
 **private** LogPvMapper **logPvMapper**;  
  
 @Autowired  
 **private** RedisTemplate<String,LogPv> **redisTemplate**;  
  
  
 @Pointcut(value = **"@annotation(com.meilong.annotation.PvCount)"**)  
 **public void** pointcut() {  
 }  
  
 */\*\*  
 \* 拦截带有PvCount注解的方法  
 \* 在执行方法后 执行该切面，并将注解传递到切面中  
 \** ***@param joinPoint*** *\** ***@param pvCount*** *\*/* @After(value = **"pointcut() && @annotation(pvCount)"**)  
 **public void** after(JoinPoint joinPoint,PvCount pvCount){  
 LogPv logPv = **new** LogPv();  
 *//方法参数* Object[] args = joinPoint.getArgs();  
 *//当前请求* HttpServletRequest request = HttpUtil.*getCurrentRequest*();  
 *//类型* logPv.setPvType(pvCount.description().getValue());  
 *//请求ip地址* logPv.setIpAddr(request.getRemoteAddr());  
 *//时间* logPv.setPvTime(**new** Date());  
 *//用户id* **if**(!StringUtils.*isEmpty*(request.getHeader(**"token"**))){  
 logPv.setUserId(**tokenUtil**.getToken(request.getHeader(**"token"**)).getId().intValue());  
 }  
 *//记录数据  
// logPvMapper.insertSelective(logPv);  
 //向redis中保存数据* **redisTemplate**.opsForList().leftPush(RedisKey.***LOG\_REDIS\_KEY***,logPv);  
 }  
  
}

代码比较简洁，在这里面，因为访问页面是比较频繁的操作，如果每次访问都进行记录那么可能会比较耗费性能，所以我选择先将记录存到redis中，在启动一个定时器，定期同步

接下来 需要增加 spring对aspectj的支持，这里要注意，如果需要拦截controller中的某个方法，那么一定要将配置，写到spring-mcv的配置文件中，而不是spring配置文件



默认为jdk代理，更改为true为cglib代理

增加一个用于同步数据的定时任务

@Autowired  
**private** RedisTemplate<String,LogPv> **redisTemplate**;  
  
@Autowired  
**private** LogPvMapper **logPvMapper**;  
  
**public void** insertLogBatch(){  
 List<LogPv> logPv = **redisTemplate**.opsForList().range(RedisKey.***LOG\_REDIS\_KEY***,0,-1);  
 **if**(logPv.size() > 0){  
 *//删除redis中的数据* **redisTemplate**.delete(RedisKey.***LOG\_REDIS\_KEY***);  
 System.***out***.println(**"触发了定时任务:"**+logPv.size());  
 **logPvMapper**.insertBatch(logPv);  
 }  
}

对定时任务类 进行配置

<**bean id="quartzJob6" class="com.meilong.common.quartz.PvLogTask"**></**bean**>

*<!-- pv log -->*<**bean id="jobtask6" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>  
 *<!-- 调用的类 -->* <**property name="targetObject"**>  
 <**ref bean="quartzJob6"** />  
 </**property**>  
 *<!-- 调用类中的方法 -->* <**property name="targetMethod"**>  
 <**value**>insertLogBatch</**value**>  
 </**property**>  
</**bean**>

<**bean id="cronTrigger6" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>  
 <**property name="jobDetail" ref="jobtask6"**></**property**>  
 *<!-- cron表达式 -->* <**property name="cronExpression"**>  
 *<!-- 测试环境 一分钟一次 -->  
 <!--<value>0 0/1 0/1 \* \* ? </value>-->* <**value**>0 0 0/2 \* \* ? </**value**>  
 </**property**>  
</**bean**>

具体调度时间可以根据需求自己设定

搞定。