[](http://image.baidu.com/i?ct=503316480&z=0&tn=baiduimagedetail&word=%D6%D0%B9%FA%D2%C6%B6%AF+logo&in=8288&cl=2&cm=1&sc=0&lm=-1&pn=4&rn=1&di=508018540&ln=140)

**中国移动专利申请**

**技术交底书**

|  |  |
| --- | --- |
| **公司编号** |  |
| **发明名称** | 一种基于行为轨迹重建的用户体验监控方法 |
| **申报单位** | 中国移动通信集团山东有限公司 |
| **申报类型** | 发明 |
| **发明人** | 何启明 |
| **技术联系人** | 何启明 邮箱: [heqiming@sd.chinamobile.com](mailto:heqiming@sd.chinamobile.com)  电话: 15069071152 |

**中国移动通信集团公司**

一、发明名称

一种基于行为轨迹重建的用户体验监控方法

二、技术领域

本发明涉及业务运营支撑系统BSS领域。

三、现有技术的技术方案

网站用户行为轨迹是访客登录网站后按时间顺序先后排列的操作序列，重建此操作序列即通过一系列方法对操作序列数据重构，真实再现过去某个时间点发生的情景，并统计形成有价值的用户体验、业务转化率和访问热点等指标数据，因此重建过程包含用户行为数据采集和数据分析两个方面。重建完整的网站用户行为轨迹有利于掌握网站的用户需求、习惯和喜好，通过和预先设定的标准行为轨迹比较，可以了解到影响交易成功率的关键环节，统计出业务转化率；分析用户行为轨迹有利于监控系统真实响应速度，传统的服务端APM数据相对于用户行为来说是脱节的，通过扩展标签技术打破用户行为轨迹数据孤岛与APM数据孤岛之间的屏障，盘活后端监控数据，可以按照系统响应构成项进行拆分，并获取客观的系统性能评价；分析用户行为轨迹还有助于检索非法恶意操作，诊断应用程序漏洞，如今面向应用业务的CC攻击占有互联网攻击类型的80%以上，构成了目前IT网站安全的最大挑战，而通过用户行为分析可过滤出一些突破业务逻辑限制，绕过应用安全策略的行为。

如今，已商用的采集技术手段有：1、用户行为采集程序与应用无缝集成，甚至在应用产品设计的初期考虑到用户行为数据输出接口，采集程序或集成在界面javascript方法中；2、在不同的业务模块中自定义外部拦截器，需要手动在页眉、页脚的共享javascript中添加外部拦截器，并在界面onload时自动加载；3、在服务端响应报文中注入通用外部拦截器，在Servlet响应客户端的HTML报文中自动注入拦截器；4、在浏览器底层通过HOOK拦截；

数据分析不是简单得基于操作序列的统计，而是在此基础上总结出业务转化率、拆分用户行为时间构成项以及完整业务聚合，实现对不同业务类型的操作复杂度、交易促成障碍热点分析。已商用的分析技术手段有：1、基于关系型数据库的筛选、剔重和排列；2、基于规则向量的聚类；3、基于扩展标签技术XTags实现的用户行为、APM数据关联，实现业务模块的智能识别和耗时构成项的拆分。

四、现有技术的缺点及本申请提案要解决的技术问题

传统的数据采集方案如下缺点：1、用户行为采集程序与应用高度集成，对于正在服役的老应用系统，改造成本较高，采集程序与应用程序代码融合，增加了代码复杂度，解耦难度增加；2、自定义外部拦截器可以对现有应用程序改造，但依然需要对各业务模块逐一改造，改造成本较高；3、从服务端统一注入拦截器的方式，避免了对代码改造，但是其无法确保通用性，无法拦截所有界面元素的操作；4、浏览器HOOK技术利用了底层的植入能力，但数据可读性差，无法与业务模块对应，因此数据利用价值偏低。

传统的数据分析方案有如下缺点：1、APM多为黑盒，采用Java Instrumentation（代码仪器）实现服务器路径跟踪，在获取业务标识时采用AOP技术，可能会递归复用应用组件中的GET方法，造成了APM在采集数据的过程中间接调用应用的方法，存在潜在的性能风险；2、APM获取的是完整的交易拓扑，实际上在进行数据关联时，只需要其中小部分数据，若APM提取这部分数据需要配置复杂的业务分析规则，变相增加了APM负担，数据处理能力下降；

为解决以上方案的缺陷，本发明即提出一种基于行为轨迹重建的用户体验监控方法，在页面公共区域注入通用外部拦截器，并使用事件触发递归、事件冒泡捕获等手段提升数据采集效率，实现一次注入，处处生效。采用Kryo系列化并输出较大体积的业务标识，对传统的APM数据进行优化过滤，最终输出用户操作序列、URI序列和服务器Purepath，再使用Pandas进行实时数据关联。

五、本申请提案的技术方案的详细阐述

1、采集模块的注入

在服务端注入用户轨迹采集模块，采用JVM Instrumentation技术在JVM继承中依附代理程序，代理程序实现了一个HttpServletResponse过滤器，并追加到公共Response链表的末端，此过滤器首先获取Response写入器，通过写入器将采集模块JavaScript引用标签写入字节流的对应位置（此处位置为主框架的</body>前端，放在此处的缘由是能够在大部分界面元素加载后加载，初始阶段便能捕获大部分用户操作），修改后的字节流最终下发到用户浏览器中，实现基于用户浏览器的数据采集，其伪代码如下：

HttpEntity entity = response.getEntity();

int offset = EntityUtils.search("</body>") - 1;

response.getWriter().write("<script type=\"text/JavaScript\" src=\"${Project.customPath}/js/ coding.js\"></script>", offset);

response.getWriter().flush();

response.getWriter().close();

这里为提升采集模块注入效率，本方案提前获取了字节流的注入位置，并在此位置写入代码，避免了响应字节流全覆盖，降低了内存开销。为提升字节流定位速度，使用了改善的并行Boyer-Moore字符串搜索算法（以下简称BM算法），首先通过字符串均匀裁减成w个滑窗，以降低搜索的数据规模，多线程并行搜索子串时使用BM算法快速搜索，当未找到字符串时将w个滑窗整体向后平移1个字符，再重复上述并行BM算法，直到向后平移m-1次，平移的目的是防止分割点正好位于待搜索字符串内部，其中m为待搜索字符串的长度，因此其计算复杂度为：

，最差为

2、用户操作序列采集模块

当前模块定义为Injection\_Core，主要分3个子模块，基于冒泡实现的主体行为捕获、系统提示拦截和对jQuery的popwin弹窗的拦截，jQuery的popwin无法通过正常页面遍历获取到，因此需要对popwin单独处理。

1）基于冒泡实现的主体行为捕获：采集模块部署到主页面框架最外层，随着用户访问CRM系统时而自动加载，并自动拦截window.load事件，启动递归流程traverser，traverser按顺序执行以下流程：

① 首先从界面元素中搜索随servlet自动注入的采集模块是否已存在，若存在则说明已具备了采集能力，没有必要进一步递归，则退出；

② 拦截当前窗体的document对象的click、change事件，即为冒泡法拦截当前document内所有已创建元素和即将创建元素的click、change事件，冒泡法于传统逐一元素处理的效率更高，覆盖更全面，而无需单独处理每一个元素，对性能影响较小。拦截采用jQuery.bind绑定事件处理函数，以获取对jQuery的良好支持，事件处理函数的主要功能为将操作时间、操作人信息和操作内容以JSON格式异步发送到服务端。 此时驻留在top对象中的uem\_json属性对象已经记录了登录的操作员工号、单位等基础数据，通过克隆此属性对象创建一个模板，并在此模板基础上写入拦截的用户操作内容、终端时间，此后即可发送数据了，发送数据采用AJAX异步方式，不影响正常业务，同时设置cookie属性XTags为随机生成的UUID，用于和服务器端的URL关联。

③ 由于操作内容获取传统的方式为getReadableInfo，即获取可读信息，即通过分析元素自身属性、内部innerText、前后兄弟属性、父元素或叔伯元素得到可读性文字，其准确性受到严重影响。如今改造为鼠标截屏、OCR文字识别的方式，提升元素名称可读性。鼠标截屏是在鼠标点击按钮等元素时，自动识别到光标所在区域的外围边界，识别方法为事件target元素的端点坐标，并以此边界做为截图的坐标范围，进行图像裁切，使用基于HTML5的第三方js：ImgCut和Html2Canvas裁减，如：

var imgCut = new ImgCut("divHidden", "./target.jpg", 150, 200, 100, 100);

再调用OCR的JavaScript实现，获取文字信息，采用第三方Tesseract.js实现。

④ 在当前document中递归其他iframe、frame对象，使用jQuery进行搜索，对于找到的iframe、frame对象，首先判断其就绪状态，若为complete，则回到①进行递归遍历；否则说明未加载完成，绑定load事件，等待其加载完成，再触发遍历。有些静态iframe、frame是动态添加到当前页面中的，在进行消息拦截时，还未出现，因此需要定义监视器，以观察即将添加的元素。在监视器定义中，首先判断新增元素的标签名是否为IFRAME或FRAME，若是则进行递归插码；若新增元素为DIV或其他嵌套元素，则进行二次定义监视器，并进行二次搜索；为避免监视对象泛滥，造成内存溢出，需要在当前document.body['monitors ']中记录监视对象，所有监视对象随body的销毁而关闭，因此需要捆绑unload事件，代码如下：

$(doc.body).bind('unload', function (e) {

while (e.target['monitors'].length > 0) {

e.target['monitors'].pop().disconnect();

}

});

2）系统提示拦截：由于CRM为定制开发的软件，其系统提示格式不尽相同，以<form/>、<frame/>、<iframe/>的，使用jQuery搜索到具体的ID后捕获；对于当前窗口直接抛异常的，从当前属性获取；对于模态框、非模态框、alert和confirm的系统提示，需要进行重写；

3）最后对于iFrame::popwin的窗体，首先添加自定义append事件，因为popwin窗体为使用append方法追加的主页面尾部，append事件回调的逻辑为：首先获取最后被添加的元素，获取popwin的document对象；其次，检查document的就绪状态，若已完成加载，则进行递归用户操作捕获，否则绑定load事件，再触发递归用户操作捕获；

至此，用户操作序列采集模块输出关系型数据结构为{XTags（主键）、终端ID、操作员ID、操作类型、元素位置、时间戳}，通过AJAX请求发送到服务端。

3、服务端URI采集模块

采集URI数据的传统方式为网络抓包，因网络数据包的解码系统开销较高，且无法进行时间构成项拆分。本提案采用HTTP Server HOOK方式采集，其中涉及到多个钩子处理方法，分别是：连接阶段的挂钩：ap\_hook\_create\_connection、ap\_hook\_pre\_connection和ap\_hook\_process\_connection；请求阶段的挂钩：ap\_hook\_create\_request、ap\_hook\_post\_read\_request。其中建立连接阶段挂钩伪代码如下：

module AP\_MODULE\_DECLARE\_DATA uri\_collect\_module = {

STANDARD20\_MODULE\_STUFF, NULL, NULL, NULL, NULL, NULL, register\_hooks

}

static void register\_hooks(apr\_pool\_t \*pool) { // 钩子注册

ap\_hook\_pre\_connection(uri\_collect\_handler, NULL, NULL, APR\_HOOK\_LAST); // 挂钩到处理程序

}

钩子处理程序uri\_collect\_handler的实现如下：

char temp[128] = {0};

now = localtime(&timer); // 获取系统时间

sprintf(temp, "{服务端时间 : %s, ", now->tm\_year + 1900, now->tm\_mon + 1, now->tm\_mday, now->tm\_hour, now->tm\_minute, now->tm\_second, now->tm\_millesecond);

sprint(temp, "IP五元组 : <%s, %s, %s, %s, %s>", req\_rec->conn->client\_addr, req\_rec->conn->client\_port, req\_rec->conn->proto\_id, req\_rec->conn->local\_addr, req\_rec->conn->local\_port);

在建立连接中引入IP五元组的原因是希望与操作系统底层TCP数据进行关联，将建立连接时间进一步细化为SYN-SENT、SYN-RECEIVED和ESTABLISHED构成项，以对用户体验的网络构成项进行深入剖析。

本提案仅截获Apache Http Server的操作系统级TCP调用，因此采用Ring3级动态库函数劫持方式进行Hook，connect函数重构为：

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen) {

static int (\*func)(int, const struct sockaddr \*, socklen\_t);

FN(func, int, "connect", (int, const struct sockaddr \*, socklen\_t));

return (\*func)(sockfd, addr, addrlen);

}

使用LD\_PRELOAD修改动态库的加载顺序，使重构的代码生效，通过HOOK分别在Apache Http Server和OS底层TCP采集了两个关系型数据结构：前者为{XTags（主键）, IP五元组, URI, 建立连接时间, 请求读取完成时间（为客户端请求上行时间，或认为网络传输时间）, 后端请求处理时间}，后者为{ IP五元组（主键）, 数据包类型（三次握手包还是数据交换包）, [三次握手包时间序列：SYN到达时间, SYN/ACK发出时间, 确认ACK接收时间], [数据交换包时间序列：PSH,ACK到达服务端时间, 接收完发送ACK时间]}。二者通过IP五元组进行关联。

4、数据传输模块

以上2、3节产生两个线条的数据分别由Apache HTTP Server的两个模块处理，action\_module和uri\_module，为保证数据传输的稳定性和较低的开销，整个传输链到分析处理前无文件落地到磁盘，两个module通过管道与Flume建立关联，配置Flume的数据源类型为Avro，通道Channel类型为Memory Channel，数据下沉为Failover Sink Processor，最后落地为Oracle12c或Mongo DB等支持JSON数据查询功能的数据库，最终查询在原始JSON数据上检索，无须进一步处理。

其中，在Apache两个模块的配置中，需要分别指定CustomLog为Flume Avro Client。

Flume Server端：bin/flume-ng agent -c ./conf/ -f conf/flume-conf.properties -n agent

Apache客户端：CustomLog bin/flume-ng avro-client -c ./conf/ -H 0.0.0.0 -p 41414 /

5、数据实时关联模块

本提案输出的数据存在以下特点：1）错位关联，由于用户操作数据的XTags标记发生在URI之后，因此当前操作的XTags与下一个操作的URI相对应；2）大部分URI都有用户操作对应，可正常关联，没有用户操作对应的数据为界面自动发送的探测请求或机器人请求，将进入审计消息队列；3）两个线条的数据均为流式数据。

传统思路为使用流式数据库PipelineDB+Kafka处理，但由于以上本提案的1）、2）条特点，需要进行查询较多特殊定制，因此采用普通批量脚本处理。上述关系型数据不妨设为：tbl\_actions(SEQID, XTags, TermId, OperId, OperType, ElemLoc, TimeStamp, STATUS), tbl\_uris(SEQID, XTags, URI, TimeComposition…, STATUS)，根据小时数据量规模，按照SEQID的前10位YYYYMMDDHH24分区，后22位为自然增加的序列值，入库时需要考虑客户端时间与服务端时间的差异，时间调整一致后入库。

关联算法如下，以用户操作数据为驱动，首先找到用户操作XTags对应的URI序列的第一条URI的SEQID，再以此URI为边界向上回溯到最近一个有用户操作对应的单个URI，及对应的XTags，最后根据此XTags找到对应的URI序列。对于处理完成的数据，更新状态为已处理。伪代码如下：

DECLARE

v\_seq VARCHAR2(32) := NULL; -- 临时序列值

v\_xtags VARCHAR2(16) := NULL; -- XTags

BEGIN

FOR cur IN (SELECT \* FROM tbl\_actions PATITION(P\_yyyymmddhh24)) LOOP

SELECT MIN(SEQID) INTO v\_seq FROM tbl\_uris PARTITION(P\_yyyymmddhh24) t WHERE t.XTags = cur.XTags;

SELECT t.XTags INTO v\_xtags FROM tbl\_uris PARTITION(P\_yyyymmddhh24) t WHERE t.SEQID < v\_seq AND t.XTags IS NOT NULL; -- 此处已过滤掉无操作的URI，有一项要求是对于浏览器自动发送URI的行为需要特殊处理，即将XTags置空，否则会将浏览器自动发送的URI归类到用户操作中

SELECT cur.\*, URI, TimeComposition… BULK COLLECT INTO s\_result FROM tbl\_uris PARTITION(P\_yyyymmddhh24) t WHERE t.XTags = v\_xtags; -- 关联结果

UPDATE TABLE tbl\_actions PARTITION(P\_yyyymmddhh24) t SET status = 0 WHERE t.XTags = cur.XTags; -- 更新处理标记为已处理

UPDATE TABLE tbl\_uris PARTITION(P\_yyyymmddhh24) t SET status = 0 WHERE t.XTags = cur.XTags; -- 更新处理标记为已处理

END LOOP;

END;

分区边界数据处理，当用户操作或URI发生在分区边界时，检索当前分区可能会出现数据丢失，需要向上或向后推移一个分区，进行检索。

当当前分区已无状态为待处理的数据时，删除此分区。

6、业务聚合模块

业务聚合即根据具体操作的业务标识进行操作划分，即获取某时间点在某终端上某位操作员办理给某位客户办理某项业务时的操作。

由于服务器捕获了全体营业员的操作数据，因此同一时间段，不同终端的操作数据混杂在一起。本步骤的实现方案为：

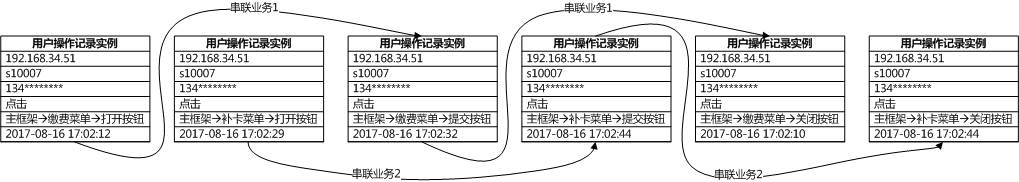
1）按照浏览器cookie中的会话标识、终端IP、操作员工号进行第一轮拆分，得到每个操作员在某台终端上的一组操作。

由于操作数据流到达服务器时已经按照先后顺序排列好了，因此对操作数据流按照先进先出队列方式处理，后续不再涉及排序。

2）从待处理用户操作队列中取出一个操作记录，以会话信息（如终端ID、操作员ID、业务受理对象等）在用户事务链表中检索，若找到匹配项，则将操作记录追加到用户事务链表中对应项的已聚合操作序列上，当此序列已匹配到菜单关闭项时，则将链表数据记录数据库；若找不到匹配项，则在链表中新增一个用户事务。



经过业务聚合最终的匹配结果类似下图所示：



本提案步骤1中获取的操作数据是带有操作控件在页面的相对位置的，因此含有详细的业务信息，在业务聚合阶段无需和URL关联。

3）以菜单打开、关闭操作作为业务开始、结束标志，对序列进行二次拆分，得到唯一的一笔业务操作序列。

至此，业务聚合过程执行完毕，本提案所提供的业务聚合流程相比于基于服务端插件监控方式更能适应多笔业务重叠的情形，当操作员受理第一笔业务过程中，交叉受理了其他业务，服务段插件能获取到的请求头业务信息仍然是第一笔业务的菜单ID和第一笔业务的受理对象，因此对交叉受理的业务无法正确识别；而本方法是通过页面控件相对其他控件的位置标注业务信息的，不依赖于具体请求头中的业务标识。

六、本申请提案的关键点和欲保护点

1）在服务端中间件Servlet中注入数据采集脚本，使用冒泡方式在页面顶端捕获成员节点的操作，降低了动态页面的全覆盖的难度，为创新点应予以保护；

2）本案提到的通过递归方式获取节点与主框架相对位置，使得操作目标元素的业务涵义更加清晰，提升了用户操作数据的可读性，属于创新点应予以保护；

3）本提案利用用户事务链表、已聚合操作序列等最终实现业务聚合，属于创新点，应予以保护；

4）本提案通过cookies::XTag方式将用户操作和服务端动态请求关联，利用HTTP服务的挂钩技术捕获用户请求的几个时间构成项，属于创新点，应予以保护。

5）本提案使用无落地的数据传输模块，提升了数据传输稳定性，降低了系统开销，为创新点，应予以保护；

6）通过带错位的实时数据关联实现了数据的异步处理，为创新点，应予以保护；

七、与第三条中最接近的现有技术相比，本申请提案有何技术优点

本发明实现了基于页面的用户轨迹重建，对所有菜单界面元素实现了用户操作捕获，仅需很小的程序改造，对正常业务逻辑没有影响。相比商业的用户体验管理和独立界面插码方式，本方案大幅降低了维护成本、程序改造成本，通过定制可以覆盖多项用户行为，通过记录页面元素漫游路径提高了用户操作数据的可读性。

八、其他有助于理解本申请提案的技术资料