一、背景

**1.1原由**

* 当前公司架构分的比较细，项目比较多，导致了一个报错，不知道如何下手去调试，也不知道该找哪个相应的人员；
* 基于运营的需求，会在各个业务点进行一些数据收集和数据统计，普遍的做法就是各个业务都新建自己的日志表，这样可能会导致日志表越来越大，直到影响到业务数据；

**1.2目标**

* 出现BUG的时候，我们不需要再去echo或者print\_r进行一步步调试，而是通过看日志就知道是哪里出了问题，该去找哪个相应的负责人；
* 统一规范，统一记录、统一查询，日志和业务分离；

**二、日志记录点**

我们罗列了以下8处需要记录的日志点，这8处包含了一个请求从发起到结束的所有过程（注意：这不代表每次请求都一定有8处日志记录）

| **序号** | **记录日志的种类** | **发起数据方** | **接收数据方** | **记录日志的代码方** | **记录方式** |
| --- | --- | --- | --- | --- | --- |
| 1 | 客户端入口请求数据 | 前端用户请求 | \*-client | \*-client | 框架自动记录 |
| 2 | 客户端请求服务器端数据 | \*-client | \*-service | \*-client | 框架自动记录 |
| 3 | 服务端接收客户端数据 | \*-client | \*-service | \*-service | 框架自动记录 |
| 4 | 服务端返回客户端数据 | \*-service | \*-client | \*-service | 框架自动记录 |
| 5 | 客户端接收服务端数据 | \*-service | \*-client | \*-client | 框架自动记录 |
| 6 | 客户端输出返回数据 | \*-client | 前端用户请求 | \*-client | 框架自动记录 |
| 7 | 数据源操作日志（数据库） | 无 | \*-service | \*-service | 框架自动记录 |
| 8 | 用户操作跟踪日志 | 无 | \* | \* | 手动记录 |

名词解释：

* \*-client：比如xf-api xf-web  xf-manage等等；
* \*-service：比如xf-service  hft-service等等；
* 框架自动记录：我们在框架里面已经封装了一层自动记录日志的功能，无需coder手动记录；
* 手动记录：需要coder自己去写日志记录；

**三、怎么记录日志**

对于框架自动记录的日志，我们可以完全当做黑盒处理，who cares... ...所以，在这里我们详细的描述一下，需要coder手动记录的日志（也就是用户操作跟踪日志）该怎么去记录

**3.1更新框架**

我们对laravel的日志系统进行了进一步的封装，如果你的代码仓库是首次引入日志系统，那么需要更新和配置一些东西：

[技术文档 Laravel 日志服务接入](http://pms.ipo.com/pages/viewpage.action?pageId=13017229)

**3.2日志种类**

框架提供的日志级别：**debug**、**info**、**notice**、**warning**、**error**、**critical** 和 **alert；**

我们常用的日志级别：**debug（开发环境调试用，生产环境关闭）、info（开发环境、生产环境都记录）、error（开发环境、生产环境都记录）、warning(开发环境、生产环境都记录)；**

主动报警日志：如果生产环境出现问题，基本上都是运维发现了之后再通知我们。但是对于有一些情况（比如：我们依赖的某个服务接口，突然报白页了），我们需要在第一时间就发现并给出相应的解决方案，这个时候我们就需要主动报警功能了。我们把**warning**级别的日志接入了运维了主动报警系统，只要是有**warning**级别的日志，您的将会在第一时间收到短信、微信以及邮件。

**3.3日志记录**

日志记录统一方法：

Log::debug($sTopic, ['context' => 'Other helpful information']);

Log::info($sTopic, ['context' => 'Other helpful information']);

Log::error($sTopic, ['context' => 'Other helpful information']);

参数1：记录的标题信息,具体格式为：大模块#小模块#动作#片段;

参数2：具体格式根据不同场景可以自定义，但一定是要一个array；

比如如下：

|  |
| --- |
| use Estate\Support\Msg; //用来组合sTopic字段    public function tryPay($aParams){          Log::debug('开始支付');          //测试阶段，需要跟踪整个传递流程；上线之后，可以关闭debug          Log::info(Msg::join('user', 'pay', 'trypay', 'input'), $aParams);            //记录不同的条件判断过程          if(!isset($aParams['iUserID'])){              Log::info(Msg::join('user', 'pay', 'trypay', 'check'), ['用户ID不能为空']);              return false;          }          $aUserInfo = User::getUserInfo($aParams['iUserID']);          if(!$aUserInfo){              //按照正常流程，是需要有用户信息才能继续流程的              Log::info(Msg::join('user', 'pay', 'trypay', 'check'), ['不存在相关信息']);              return false;          }          //记录不同的分支条件情况          switch($iStatus){              case 1:                  Log::info(Msg::join('user', 'pay', 'trypay', 'check'), ['用户状态为1']);                  //TODO SOMETHING                  break;              case 2:                  Log::info(Msg::join('user', 'pay', 'trypay', 'check'), ['用户状态为2']);                  //TODO SOMETHING                  break;              case 3:                  Log::info(Msg::join('user', 'pay', 'trypay', 'check'), ['用户状态为3,不能支付']);                  //TODO SOMETHING                  break;              default:                  break;          }          //记录调用不同接口情况          Log::info(Msg::join('user', 'pay', 'trypay', 'trypay'), $aParams);          $bResult = PayModel::pay($aParams);          if($bResult){              Log::debug('支付成功');          }else{              Log::debug('支付失败，原因：XXX');          }          Log::info(Msg::join('user', 'pay', 'trypay', 'success'), ['aResult'=>$bResult]);          return $bResult;  } |

日志记录如下：

|  |
| --- |
| {      \_id: "56177aa2d031dc0e098b456b",      time: "2015-10-09 16:28:18.491000",      level: "INFO",      trackID: "1B8530FF-3BB4-1E7D-B094-8BE6F2FC85AB",      from: "search-service",      env: "dev",      customerip: "10.59.92.116",      requestip: "10.59.72.21",      url: "[http://search.linhaisong.s.dev.anhouse.com.cn/rpc"](http://search.linhaisong.s.dev.anhouse.com.cn/rpc),      method: "POST",      message: "response",      context: {status: 200,      response: "{          "bSuccess":true,"aData":{"aList":[{"id":1},{"id":10325},{"id":10326},{"id":10327},{"id":10328},{"id":10329},{"id":10330},{"id":10331},{"id":10332},     {"id":10333}],"iTotal":21252,"iPerPage":10,"iCurrentPage":1,"iLastPage":2126,"iFrom":1,"iTo":11}}"      },      extra: [ ],      updated\_at: "2015-10-09 16:28:18.725000",      created\_at: "2015-10-09 16:28:18.725000"  }, |

**3.4执行时间记录**

为了方便大家查看代码的执行效率，我们统一封装了一层代码记录执行时间、消耗内存的代码的日志记录，方式如下：

|  |
| --- |
| use Estate\Support\PerformanceRecoder;    public function testTime(){      //注意1：start和stop需要成对出现，会产生一条日志记录      //注意2：底层封装的时候，是使用Log::debug(),所以生产环境是不记录该日志的      //注意3：两个方法的参数统一定义，以方便之后的数据统计      PerformanceRecoder::start($sTopic);      //TODO SOMETHING      PerformanceRecoder::stop($sTopic);  } |

日志格式如下：

|  |
| --- |
| {      \_id: "56177aa2d031dc0e098b4568",      time: "2015-10-09 16:28:18.462000",      level: "DEBUG",      trackID: "1B8530FF-3BB4-1E7D-B094-8BE6F2FC85AB",      from: "search-service",      env: "dev",      customerip: "10.59.92.116",      requestip: "10.59.72.21",      url: "[http://search.linhaisong.s.dev.anhouse.com.cn/rpc"](http://search.linhaisong.s.dev.anhouse.com.cn/rpc),      method: "POST",      message: "testTime-testTime",      context: {          start\_time: 1444379298.4624,          end\_time: 1444379298.4625,          time\_usage: 0.000079154968261719,          start\_memory: 8388608,          end\_memory: 8388608,          memory\_usage: 0      },      extra: [ ],      updated\_at: "2015-10-09 16:28:18.719000",      created\_at: "2015-10-09 16:28:18.719000"  }, |

**四、日志的查询**

日志统一记录在日志系统，即journal-service系统。

我们之后可能会有一个管理平台，但是目前只提供一个简单的接口以供查询。

domain：<http://journal.polaris.s.dev.anhouse.com.cn/2.0/journal?sRecorder=overseas-service&sLevel=INFO>

请求方式： get

筛选条件：

* sRecorder——日志记录方；
* sTopic——使用Log::info记录时的第一个参数；
* sTrackID——追踪ID（由框架自动生成），一次请求使用同一个sTrackID串联起所有的请求，使用此参数，你可以取得某一次请求产生的所有记录；
* sLevel——INFO DEBUG 或者ERROR等；
* iPageSize——每页数量，默认15