Table of Contents

# 1. 总则

下载: [Word](日志规范.docx)

所有服务器日志发往 [rsyslog](http://www.rsyslog.com)，使用 facility LOCAL6。

不同服务的日志通过程序名区分，也就是 man 3 syslog 中 openlog 的第一个参数。有的 syslog 库里也称为 tag 或者 programname。对应命令行工具 logger 的 -t 参数。

程序名**必须**遵守下面的格式

PROJECT.APP.VERSION.ENV.IDENTIFIER

* PROJECT: 项目名
* APP: 应用名，比如 web, worker, server, pusher
* VERSION: 版本，必须是主 git 仓库有效的 revision。生产环境使用部署的最新 git commit sha1 hash 的前 7 位。
* ENV: 环境，比如 ci, staging, xy (玄云机房), prod 等等
* IDENTIFIER: 用来区分相同环境中同一个应用的不同实例，比如服务器可以用 CENTER 的服务器 ID

一个简单的使用命令号发送日志的例子：

logger -t ops.cmd.latest.test.1 -p local6.info test

已经使用 [skynetx](https://gitlab.3pjgames.com/xi/skynetx) 的项目只需要配置以下环境变量即可

* SX\_LOGSERVICE 配置成 sx\_syslog
* SX\_LOGGER 配置成程序名

并保证 skynet 的启动配置文件里，logservice 和 logger 使用的是对应的环境变量

logger = "$SX\_LOGGER"  
logger = logger ~= "" and logger or nil  
logservice = "$SX\_LOGSERVICE"  
loglevel = "$SX\_LOGLEVEL"

每条日志可选包含 skynet 服务地址，必须加在日志开头，使用16进制，字母小写，包括在 [: 和 ] 中，并和之后的日志使用一个英文空格分开。

[:0000001b] Hello

日志的主体是一个 JSON Object，否则会转化成一个只包含 msg 字段的的 JSON Object，值就是日志的内容，比如上面的日志和下面是等效的

[:0000001b] {"msg":"Hello"}

使用 skynetx 的项目只需要使用 sx.logger 来输出日志即可。

剩余的文档会对日志的字段进行规范化。

# 2. 通用

* 优先使用字符串和数字做为字段的值。除非要做大小判断，和累加等操作才使用数字，比如用户 ID 等应该使用字符串为值。
* 同一个字段名不能一会是字符串，一会是数字，会导致日志索引出错还不被收录。
* 尽量少使用嵌套
* 避免把内部的复杂数据结构直接输出日志，整理出哪些字段是要做查询的，单独提出来，其余的通过 inspect, cjson 等先转成字符串。
* 有新字段请同时更新本文档
* 本文档未收录的字段建议有 extra\_ 开头以避免和未来添加的字段规范冲突

# 3. 字段命名

字段名必须使用小写字母开头，允许包含字母，数字和下划线。使用容易理解的字段名，因为查询日志的人并不是写日志的人，减少使用晦涩的简称。

数字字段的名字必须是能通过后缀推断出来的，比如 \_total, \_diff, \_seconds, \_count, \_amount 等等。

日志中的所有嵌套字段首先会被摊平，使用 \_ 连接字段名，比如

{  
 "res": {  
 "gem\_diff": 100,  
 "gem\_total": 500  
 }  
}

和下面的结构是等价的

{  
 "res\_gem\_diff": 100,  
 "res\_gem\_total": 500  
}

所有字段根据前缀分组成模块，需要记录的数据如果符合某个模块必须使用对应的模块输出日志。如果没有模块能满足，使用项目名做为自定义模块，即和程序名中的 PROJECT 相同，或者使用 extra 模块。

因为日志最终会被摊平，模块可以使用嵌套的结构，也可以使用摊平之后的结构。注意使用嵌套的模块格式的话，slogger 并不会递归合并字段。

对于通用性高的可以按照本文档定义新的模块并提交 Merge Request。

注意在 Graylog 中所有 JSON 中提取的字段都会添加 json\_ 的前缀，等项目日志规范下来之后会去掉该前缀。

# 4. 日志模块

## 4.1. 保留

保留模块的字段不需要添加前缀，用于日志内部一些字段，和在框架自动记录的字段。应用中使用这些字段不保证会被记录到最终的日志中。

### skynet\_addr (string)

服务地址，日志在解析时会自动从日志开头提取类似 [:0000001b]  这样的地址并保存成 skynet\_addr 字段，表示输出日志所在的 skynet 服务地址。

### name (string)

日志对象的名字，一般用于记录日志所在的文件。

### encode, printer

用于 slogger 初始化配置的保留字段。并且在初始化时，level 是用来配置要现实的最大日志等级，time 用来配置获得当前时间的函数。

### level (integer)

日志等级，数字越小严重程度越高，具体对应关系为

debug = 7  
 info = 6  
 warn = 5  
 error = 4  
 fatal = 3

### application\_name (string)

程序名，就是 openlog 中第一个参数，或者命令行中 -t 选项指定的值，或者是 skynet 中 logservice 使用 sx\_syslog 后 logger 中的值。

### app\_project, app\_name, app\_version, app\_env, app\_identifier (string)

依次对应程序名规范中 PROJECT.APP.VERSION.ENV.IDENTIFIER

### source (string)

日志产生所有的主机名

### time (string)

时间

### timestamp

Graylog 内部用来记录日志时候的字段，会从 time 字段中解析，如果没有 time 字段会使用日志记录的时间。

## 4.2. 全局

全局字段包含最常用的一些字段，同样不需要加前缀。

### msg (string)

可读的日志消息

### msg\_zh (string)

中文日志消息，用这个字段以后可以支持中文分词，让搜索更精确。

### stacktrace (string)

调用栈，帮助定位错误。Lua-xi 中的 slogger 提供了 xpcall\_handler 方便记录发生错误时的调用栈。

### fingerprint (string)

使用 “,” 分隔的用于 Sentry 错误自定义分组。详细参考 Sentry 关于 fingerprint 的说明。

### file (string), line (number)

日志所在位置的文件名和行号，同样用于定位错误。Lua-xi 提供了 fileline 方法。

### module (string)

用于表示程序中的功能模块，比如副本，邮件等。

### submodule (string)

功能子模块。

### func (string)

当前执行的命令，比如删除邮件

### args (string)

命令的参数，用于区分同一个命令多次执行，比如删除邮件可以是要删除的邮件 ID 列表。

### resp (string)

请求返回结果

## 4.3. 请求 (request)

请求模块用于记录一条 API 请求相关信息。每个请求应该有一条主日志包含所有该次请求相关的信息，一些信息无法整合在一条日志的再单独记录，比如道具数量。一次请求的多条日志使用相同的 request\_id。

### request\_id (string) 必须

请求的唯一 ID。如果一个请求产生多条日志，他们**必须**有相同的 request\_id。

### request\_duration (number)

请求响应时间，单位是秒，如果精度上允许可以使用小数。

### request\_status (string)

请求的处理状态，使用 OK 表示成功，其它的状态由项目自己确定。

### request\_error (string)

请求成功的话不加该字段，出错了加上错误信息。

## 4.4. 玩家 (user)

### user\_id (string) 必须

用户 ID，Center 返回的 token 解码后得到。对应 Center 中的用户 ID

### user\_provider (string)

玩家登录渠道。

## 4.5. 档案 (profile)

档案表示玩家在某个区中的游戏帐号，一般一个玩家在一个区中只能有一个档案。项目可以根据情况增加额外字段。

### profile\_id (string)

档案 ID，对应 Center 上的游戏档案 ID

### profile\_server\_id (string)

档案所有服务器 ID，合服后这个应该是档案原来所在的服务器 ID

### profile\_level (integer)

玩家等级

### profile\_nickname (string)

玩家昵称

### profile\_online\_duration (number)

玩家在线时长，建议每次请求的时候都记录一次，值是距离上一次记录以来在线时间的秒数。精度允许的话可以使用小数。

### profile\_register\_date (integer)

注册日期，使用 Linux timestamp 加上服务器时间时区偏移 (东8区是 28800) 整除 86400 得到。

### profile\_last\_login\_date (integer)

玩家上次登录日期, 使用 Linux timestamp 加上服务器时间时区偏移 (东8区是 28800) 整除 86400 得到。

## 4.6. 客户端 (client)

记录客户端的信息，最好客户端登录的时候上报信息到服务器方便定位错误

### client\_id (string)

客户端 ID，Center 分配的 API Key 开头的数字

### client\_version (string)

客户端版本信息，使用打包时的 git commit sha1 hash 前 7 位加上热更资源版本号组成。

## 4.7. 资源 (res)

资源用来统计充值，代币等数量。资源名称根据项目确定，通用的资源请使用下面的名称：

* currency 玩家充值到游戏中的真实货币
* gem 钻石，代替货币在游戏中流通的代币
* coin 金币，次级代币，一般只能从 gem 到 coin 单向兑换

下面字段说明使用 NAME 代理实际的资源名，比如 res\_NAME\_change 对应钻石就是字段 res\_gem\_change

大部份字段可选，如果要记录变动，**必须**提供 change 和 total 两个字段。

资源变动的原因可以使用 res\_reason，搭配其它模块一起使用，比如全局中的 module, submodule, func 和 args 等字段，订单 order 模块中的字段。

### res\_reason (string)

资源变动的原因，对 module, submodule, func 和 args 的补充，如果这 4 个字段已经描述得足够清楚可以不需要这个字段。

### res\_NAME\_change (number)

资源本次变动，正数为增加，负数为减少。注意对于 currency 的话充值是正数。

### res\_NAME\_total (number)

资源本次变动后剩余总数。

### res\_NAME\_unit (string)

资源单位，一般不需要，对于 currency 使用 ISO4217 中规范的 3 位字母标识货币种类，比如 人民币 CNY；美元 USD；欧元 EUR。

## 4.8. 道具 (item)

可以是消耗物品，或者服务等等

### item\_id (string)

物品，服务等等的 ID

### item\_type (string)

道具类型

### item\_reason (string)

本次变化原因，和 res\_reson 相同，如果全局字段已经足够就不需要再添加此字段。

### item\_change (number)

本次数量变化

### item\_total (number)

本次数量变化后玩家剩余的数量

## 4.9. 任务 (mission)

用于跟踪任务，成就等等

### mission\_id (string)

任务唯一标识

### mission\_type (string)

任务类型

### missing\_started (number)

如果要记录任务的开始状态，在任务开始的时候记录一次 1

### missing\_completed (number)

如果要记录任务的完成状态，在任务完成的时候记录一次 1

### missing\_failed (number)

如果要记录任务的失败状态，在任务失败的时候记录一次 1

### mission\_progress

使用 0~1 表示任务目前的进度，0 表示刚开始，1 表示已经完成

## 4.10. 关卡 (stage)

用于跟踪各种关卡

### stage\_id (string)

关卡唯一标识

### stage\_type (string)

关卡类型

### stage\_entry (string)

进入途径，手动还是扫荡等等

### stage\_started (number)

关卡开始的次数，手动就是 1，扫荡就是扫荡的次数

### stage\_completed (number)

关卡成功的次数

### stage\_failed (number)

关卡失败的次数

## 4.11. 订单 (order)

订单引起的资源变化使用资源 (res) 模块记录

### order\_id (string)

Center 订单号

### order\_product (string)

玩家购买的商品

## 4.12. 引导 (guide)

### guide\_id (string)

玩家完成的引导 ID

# 5. 事件规范

事件规范是对游戏中一些重要事件的格式做出定义，方便数据统计。

## 注册/登录

玩家每次登录的时候记录一次，必须字段：

* module 必须为 auth
* func 如果是新注册的帐号填 register，否则填 login
* user\_id
* profile\_id
* profile\_server\_id
* profile\_nickname
* profile\_register\_date 需要在数据库记录，每次登录的时候都需要带上注册日期用于留存计算。使用 Linux timestamp 加上服务器时间时区偏移 (东8区是 28800) 整除 86400 得到。
* profile\_last\_login\_date 设置成当天的日期, 使用 Linux timestamp 加上服务器时间时区偏移 (东8区是 28800) 整除 86400 得到。

### profile\_last\_login\_date (integer)

* request\_status OK 表示登录成功，失败使用对应的 auth 状态码，比如 403

## 异常

异常使用 msg 和 stacktrace 两个字段记录。可以额外使用 fingerprint 来做自定义的分组。

* msg 格式如下：“file:lineno: error message”。如果使用 pcall 和 xpcall 直接使用返回的错误消息就可以了
* stacktrace 使用 debug.traceback 返回的调用栈
* fingerprint 用于自定义错误消息分组

可以直接使用 slogger.xpcall\_handler 来简化。

# 5. Skynetx Tips

## 转发配置

Skynetx 中提供了 sx.logger 来输出日志。部署的时候需要配置环境变量，下面是三国内测服务器的例子：

SX\_LOGGER=cc.server.a6d64d5.prod.18  
SX\_LOGLEVEL=debug  
SX\_LOGSERVICE=sx\_syslog

其中 SX\_LOGGER 的格式为

PROJECT.APP.VERSION.ENV.IDENTIFIER

各字段说明如下：

* PROJECT: 项目名.
* APP: 应用名，比如 web, worker, server, pusher.
* VERSION: 版本，可以用 git commit sha1。注意不能有 . 出现
* ENV: 环境，比如 ci, staging, xy (玄云机房).
* IDENTIFIER: 用来区分相同环境中同一个应用的不同实例，比如服务器可以用 CENTER 的服务器 ID.

公司内项目组自己维护的机器如果想接入日志转发，请按照下面步骤配置

1. 创建 /data/log 目录，日志会在这个目录下保存一份本地备份，以防日志系统本身出问题。

* sudo mkdir -p /data/log  
   sudo chown syslog:adm /data/log

1. 新建文件 /etc/rsyslog.d/99-salt-rsyslog.conf

* $template PJFile,"/data/log/%programname%.log"  
   $FileOwner syslog  
   $FileGroup syslog  
   $CreateDirs on  
   $DirCreateMode 0755  
   $FileCreateMode 0640  
   local6.\* ?PJFile;RSYSLOG\_SyslogProtocol23Format  
   local6.\* @office.3pjgames.com:1514;RSYSLOG\_SyslogProtocol23Format

目前公司和 UCloud 机房只会转发 Sentry 还没有配置日志搜索。

## 日志记录

在创建日志对象的时候可以转入一些通用的字段，使用 child 创建子日志器的时候能继承所有父日志器的字段。比如当用户登录之后可以创建日志器

local sxlogger = require "sx.logger"  
local logger = sxlogger.create({  
 user\_id="1",  
 user\_server\_id="1",  
 profile\_id="1"  
})  
agent.logger = logger

而每次请求可以创建子日志器，子日志器中新加的字段不会影响父日志器

local logger = agent.logger:child({  
 request\_id = generate\_request\_id()  
})

已经创建的日志器，想要添加或者修改通用字段可以直接修改 context 成员，比如登录玩家换了新的客户端

logger.context.client\_id = "12"

注意继承字段时，包括输出日志和 context 中已有的字段合并的时候都不会递归合并 table。

# 6. 日志搜索

这里只介绍了部分最常用的功能，详细文档参考 [Graylog2 的相关章节](http://docs.graylog.org/en/2.1/pages/queries.html)

## 搜索语法

搜索语法类似搜索引擎，但是引入了根据字段搜索的操作符

搜索所有包含 ssh 的日志

ssh

搜索所有包含 ssh **或** login 的日志

ssh login

搜索所有精确包含短语 “ssh login” 的日志，使用英文引号

"ssh login"

搜索所有邮件模块的日志

module:mail

搜索有 res\_gem\_change 字段的日志

\_exists\_:res\_gem\_change

搜索没有 res\_gem\_change 字段的日志

\_missing\_:res\_gem\_change

### 逻辑操作符

默认多个条件使用空格分开是或的关系，也可以显式地使用 OR，使用 AND 连接条件是与的关系，使用 NOT 是返回所有不满足指定条件的日志。使用括号可以改变结合的优先级。

("ssh login" AND (source:example.org OR source:another.example.org)) OR \_exists\_:always\_find\_me

上面例子返回『包含短语 “ssh login” 并且主机名是 example.org 或者 another.example.org』的日志，或者是『有字段 always\_find\_me 的日志』

AND, OR, NOT 都必须是全大写的英文字母。

### 通配符

使用 \* 可以匹配 0 到任意个字符，比如

soruce:cc\*-xy

可以匹配 source 是 cc-xy, cc1-xy 和 cc10-xy 的日志。

使用 ? 可以匹配任意 1 个字符，比如

source:cc?-xy

在上面三个例子中只会匹配 cc1-xy

### 数字比较

只能用于数字类型的字段

区间，中括号 [ 和 ] 是包含边界的，花括号 { 和 } 是不包含边界的。

res\_gem\_change:[100, 200]  
res\_gem\_change:{100, 200}  
res\_gem\_change:[100, 200}  
res\_gem\_change:{100, 200]

也可以只指定单边的范围

res\_gem\_change:>0  
res\_gem\_change:>=0  
res\_gem\_change:<0  
res\_gem\_change:<=0

多个条件可以用逻辑操作符组合

res\_gem\_change:(>0 AND <100)

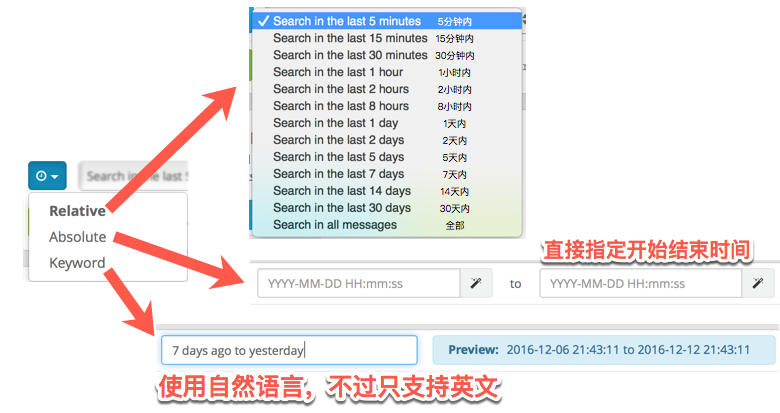
### 转义

以下特殊字符必须在之前添加 \ 进行转义

& | : \ / + - ! ( ) { } [ ] ^ " ~ \* ?

### 时间范围

时间范围有三种方式指定，只是查看最近的记录可以使用 Relative，想要精确查看某个时间段的可以使用 Absolute 来指定。



时间范围