

# Openresty游戏反外挂应用

罗宇翔

poembro@126.com

# 目录

- 业务场景
- 产品介绍
- 问题 & 总结
- QA

# 业务场景

- 大多游戏开发商把精力放在游戏业务场景上，忽略了安全模块。如防止外挂，防盗号
- 游戏账号租赁



⚠ 您已被 Steam 封禁

重新连接

# 针对这些现象我们实现了反外挂产品：

- 外挂规则库
- 用户游戏环境检查
- TCP网络校验
- 特征码校验
- 扫内存代码校验

# 特征码扫描：

在windows环境下，所有exe或者dll（linux .so 动态库文件）文件，都是一个固定格式的pe文件格式，

先拿到服务端特征码  
全盘扫描正在运行的PE文件  
匹配到特征则上传至服务端做校验。

4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ?.....yy..
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	?.....@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00	.....
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..?..???L?Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program cann
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode.....\$.....
37 E5 10 59 73 84 7E 0A 73 84 7E 0A 73 84 7E 0A	7?Ys其.s其.s其
E4 40 00 0A 77 84 7E 0A 54 42 03 0A 5F 84 7E 0A	鉸..w其.TB..其
54 42 10 0A DC 84 7E 0A 54 42 13 0A 8A 84 7E 0A	TB..驍~.TB..始~
B0 8B 23 0A 6C 84 7E 0A 73 84 7F 0A 9A 85 7E 0A	壘#.1其.s?.疎~
60 8C 17 0A F6 84 7E 0A B0 8B 21 0A F8 84 7E 0A	?..鯨~.壘!.鶯~
54 42 0C 0A 9C 84 7E 0A 54 42 02 0A 72 84 7E 0A	TB..漢~.TB..r其
54 42 06 0A 72 84 7E 0A 52 69 63 68 73 84 7E 0A	TB..r其.Richs其
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....PE..L..
00 00 00 00 00 00 00 00 50 45 00 00 4C 01 02 00	糴鮫.....?
B7 84 F5 49 00 00 00 00 00 00 00 00 E0 00 03 01	.....,..?.....
0B 01 08 00 00 1E 2C 00 00 F8 17 00 00 00 00 00	.....0,....0
00 10 00 00 00 10 00 00 00 30 2C 00 00 00 40 00	.....
00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00	.....`K.....
04 00 00 00 00 00 00 00 00 60 4B 00 00 04 00 00	?.....
C1 34 13 00 02 00 00 00 00 00 10 00 00 10 00 00	.....
00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00	.....
00 00 00 00 00 00 00 00 94 42 4B 00 C7 03 00 00	.....
00 B0 4A 00 BA 8A 00 00 00 00 00 00 00 00 00 00	.....搵K.?
	...鷹.簪.....

# text代码段校验：

上面讲的编译好的PE会有固定的结构，其中有一个区段text是存放执行代码的，在正常情况下是不会更改的，而外挂可能会修改游戏代码来改变游戏的逻辑，所以我们可以校验text段的代码是不是被修改了，校验的方式一般有2种，一种是跟文件比较，另外一种是用hash整个代码段的值跟正确的值比较。

# 保护游戏进程：

防止dll注入到游戏进程空间；

阻止外部程序打开游戏 监控游戏dll加载情况 监控外部程序往游戏内部分  
配内存情况

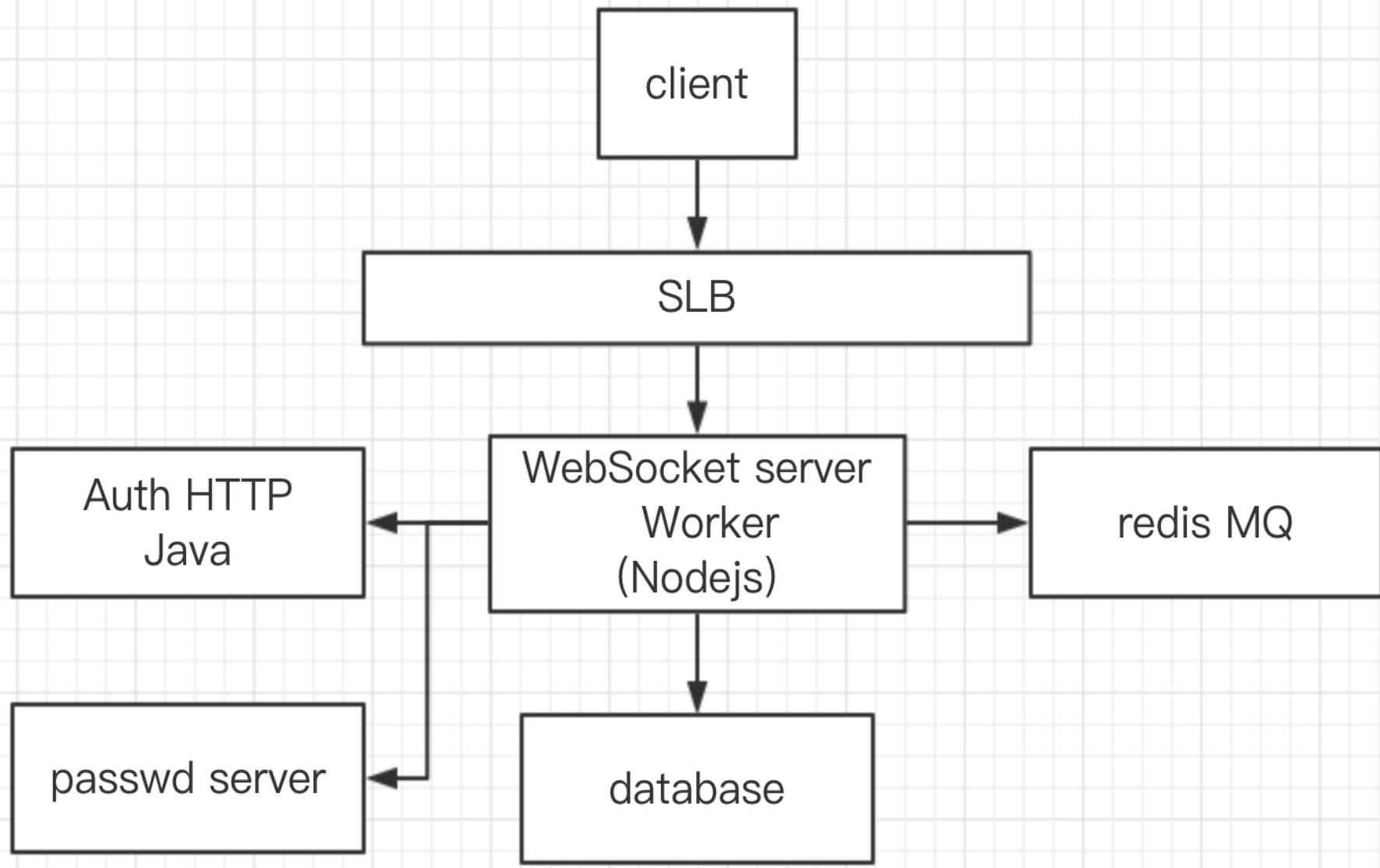
# 检查函数调用堆栈

某些外挂可能会自己来调用游戏的函数，比如实现自动捡物，自动走路的功能，这个时候需要在关键函数位置回溯堆栈判断是否有非法的调用

服务端

# V1版本

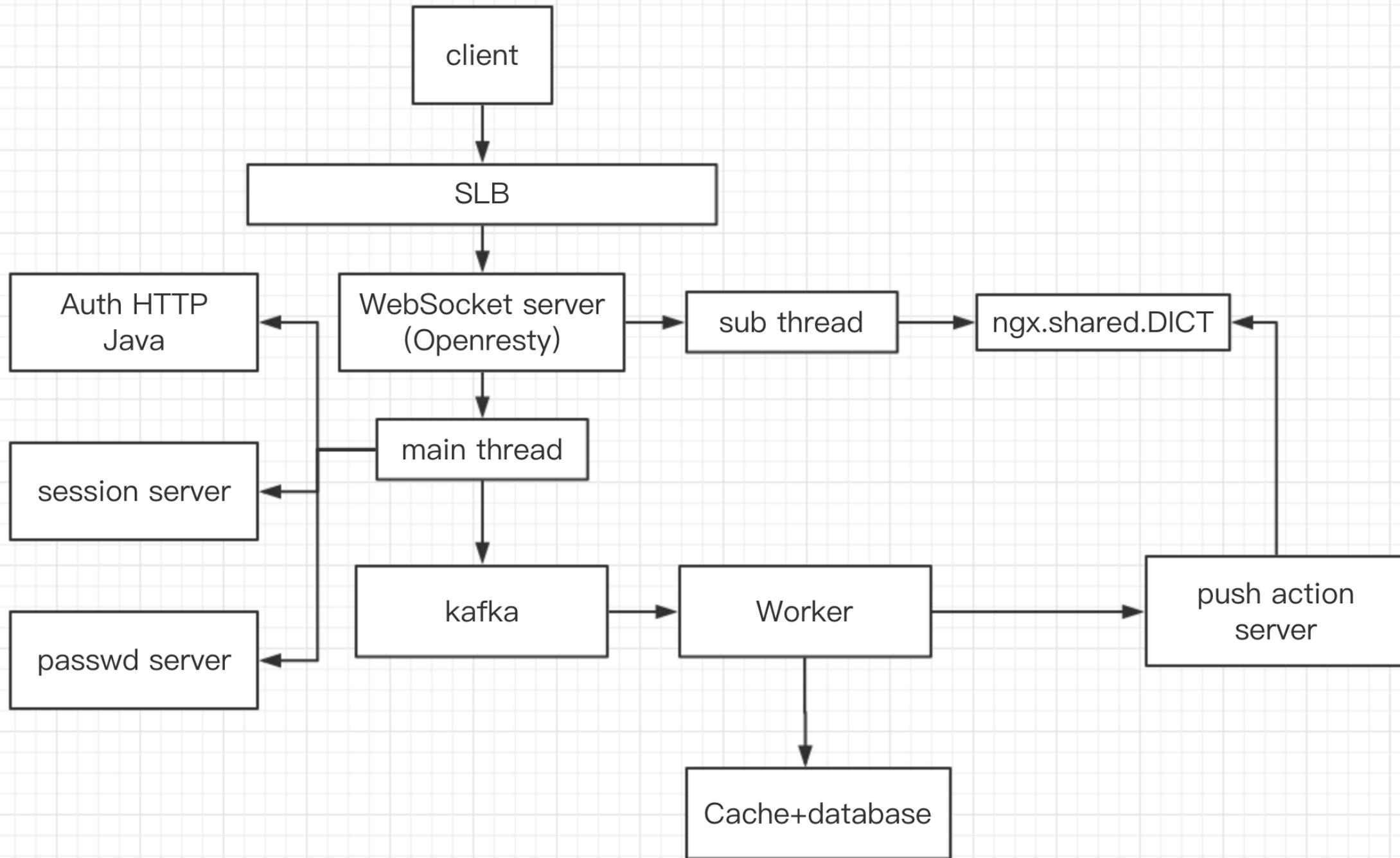
- 开发周期短
- 业务新颖
- 迭代快
- 重点是自动发现作弊用户



# 关于可靠性问题

- 突发流量
  - 1.Redis内存消耗高
  - 2.扫描时间过长
  - 3.服务掉线频繁
- 外部服务消息推送介入复杂
- 业务功能耦合

# 重构接入层

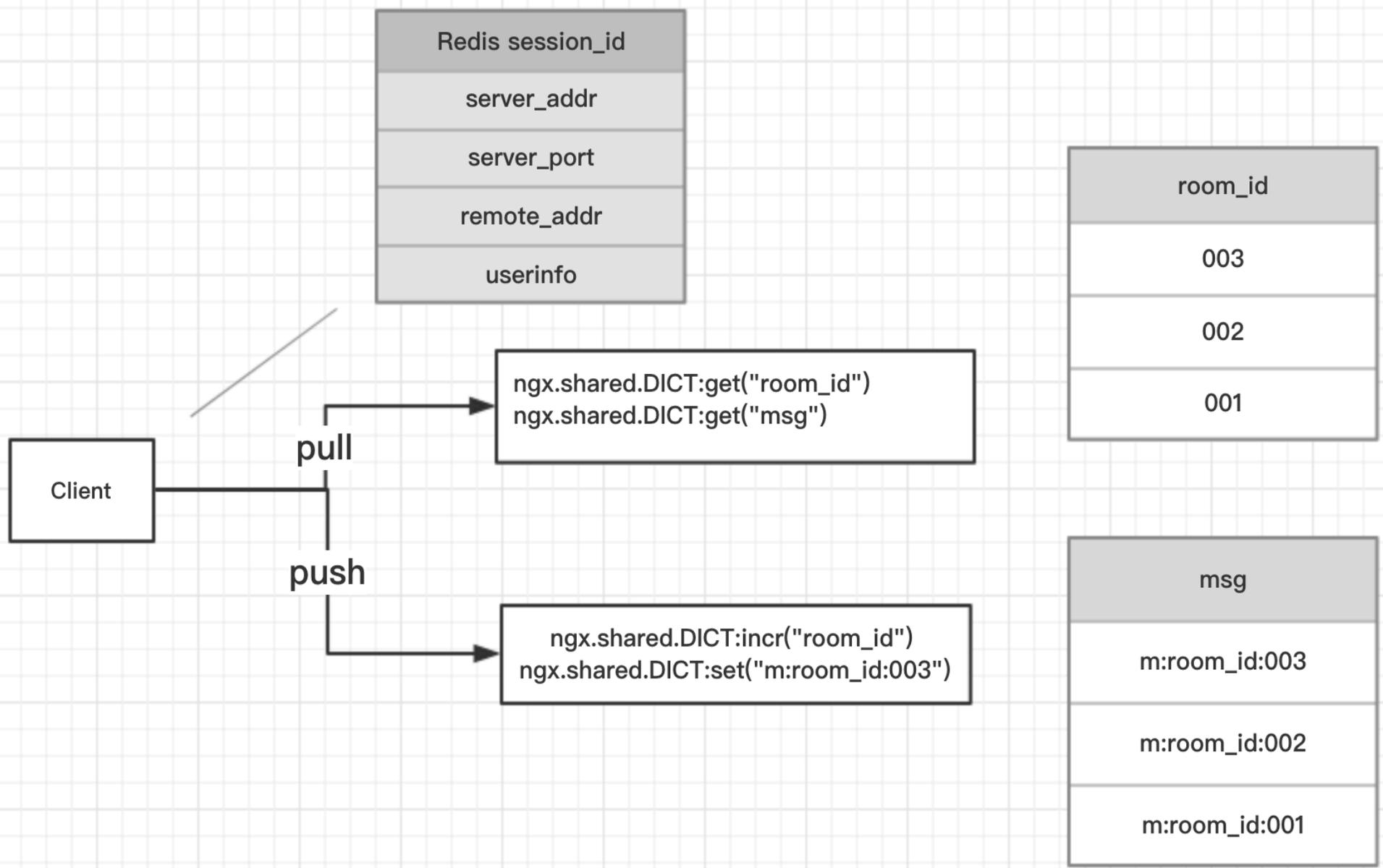


# 反外挂

- 接入层
- 用WebSocket协议做长连接
- 负责：
  1. 用户游戏环境信息检查
  2. 流控，异常告警
  3. 大文件分片上传
  4. FFi调用加、解密库与客户端保持一致
  5. 已知外挂拦截
  6. 广播消息

# 关于推送：

- 两个线程推拉分开
- 在线数据用redis zset
- 消息用ngx.shared.DICT
- 消息日志记录，防止漏消息问题排查
  
- <https://github.com/poembro/openresty-websocket-demo>



Redis session_id
server_addr
server_port
remote_addr
userinfo

Client

pull

```
ngx.shared.DICT:get("room_id")
ngx.shared.DICT:get("msg")
```

push

```
ngx.shared.DICT:incr("room_id")
ngx.shared.DICT:set("m:room_id:003")
```

room_id
003
002
001

msg
m:room_id:003
m:room_id:002
m:room_id:001

## 解决痛点：

- 开发效率
- 高并发
- 资源占用少
- 外部服务消息走HTTP介入推送
- 热更新，已在线不受影响

# 问题&总结

# lua-resty-string to\_hex 方法反转

```
local str =  
"00000001A849CD33975EB1D1A8BF7937B39CD8D100000002A849CD33975E  
B1D1A8BF7937B39CD8D2"  
  
function fromhex(str)  
    return (str:gsub('.', function (cc)  
        return string.char(tonumber(cc, 16))  
    end))  
end  
  
ngx.say( fromhex(str) )
```

# 关于MessagePack v4

使用

<https://github.com/catwell/luajit-msgpack-pure>

如右图：

进行unpack时遇到多级数据  
出现 unimplemented

推荐:

<https://github.com/chronolaw/lua-resty-msgpack>

```
local cJSON = require "cjson"
local mp = require "resty.luajit-msgpack-pure"
function from_hex(str)
    return (str:gsub('.', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end
```

```
local data =
from_hex("1401000100c501001f8b08000000000000bad7cd9aee34a775e8617391ef3ff96bb
c57970722e8a9344cd244549543672c099144771102505017219f8c6807de5004172990..." )
```

```
local offset, res = 0
while true do
    offset, res = mp.unpack(data, offset)
    if not offset then break end
    if type(res) == "table" then
        print(offset, type(res), cJSON.encode(res))
    else
        print(offset, type(res), res)
    end
end
end
```

# 关于超时 error.log 狂刷lua tcp socket read timed out

```
local data, typ, err
while true do
    data, typ, err = wb:recv_frame()
    if not data then
        if not string.find(err, 'timeout ', 1, true) then
            ngx.log(ngx.ERR, '--> timeout :', err)
            break
        end
    end
end

if typ == 'close' then
    break
end
ngx.sleep(1) --多余的, 有lua_yield(L, 0);
end
```

QA