360云查杀服务：
从零到千亿级PV的核心架构变迁

魏自立
@zieckey

个人简介

- 2009~2011年 HIPIHI 创业团队
  - 虚拟世界、游戏服务器、IM系统
- 2011年 奇虎360
  - /360/核心安全事业部/云引擎团队
- 在360工作期间参与设计开发的系统
  - 云查杀
  - 网盾
  - 360搜索一级引擎
  - 骚扰电话拦截

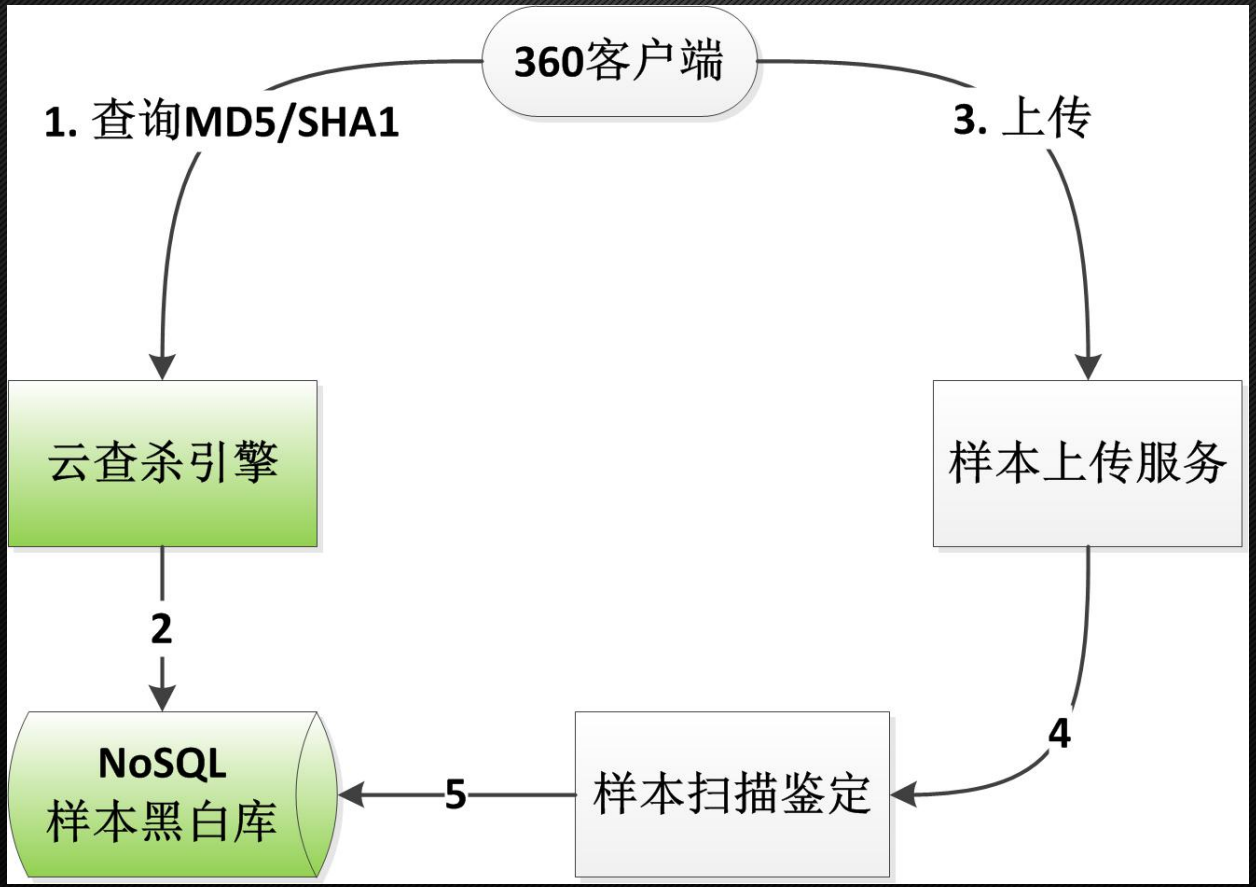
大纲

- 云查杀服务介绍
- 云查杀服务核心在线引擎的架构变迁
  - 七次架构变迁的内因
- 总结

什么是云查杀？

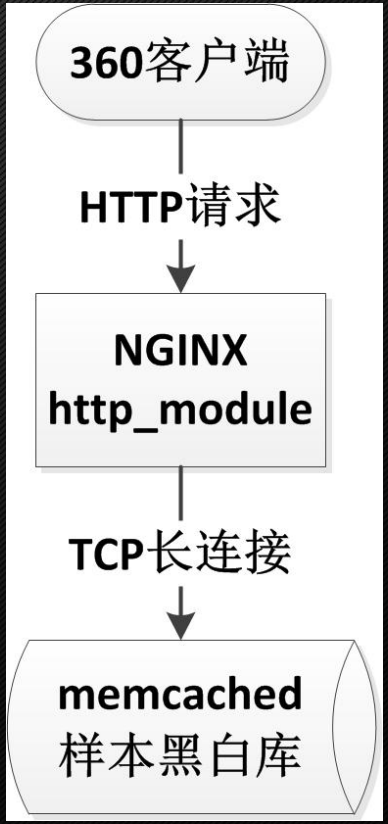
- 传统杀毒模式
  - 查杀时效性严重滞后
- 云查杀
  - 海量终端采集样本
  - MD5/SHA1
  - 黑名单/白名单
  - 云端结果优先
  - 非白即黑

360云查杀系统架构图



第1代架构

- 必须能够快速上线
- 抗住高并发、大压力
- 大方向上尽量考虑长远些
- 框架选型
  - HTTP：Nginx
  - 存储：单机、内存

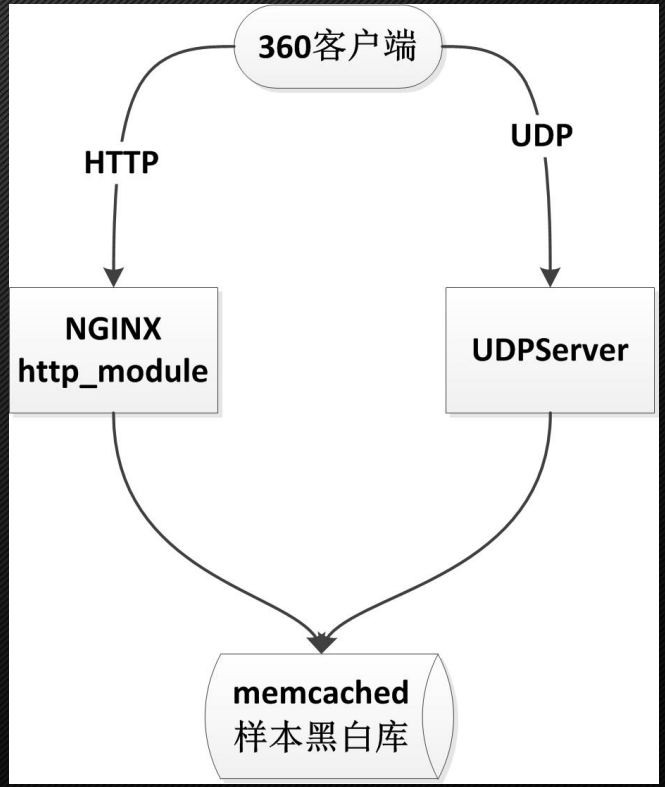


业务快速发展带来的问题

- 服务器性能压力很大
  - CPU负载高
  - HTTP请求连接进不来
- 程序优化
- 参数调优: Nginx、系统
- 扩容服务器？
  - 时间和钱

第2代架构：UDP

- 网络包大小：1KB
- UDP协议更轻量级
- UDP的QPS是HTTP数倍
- 实现
  - Memcached+Nginx

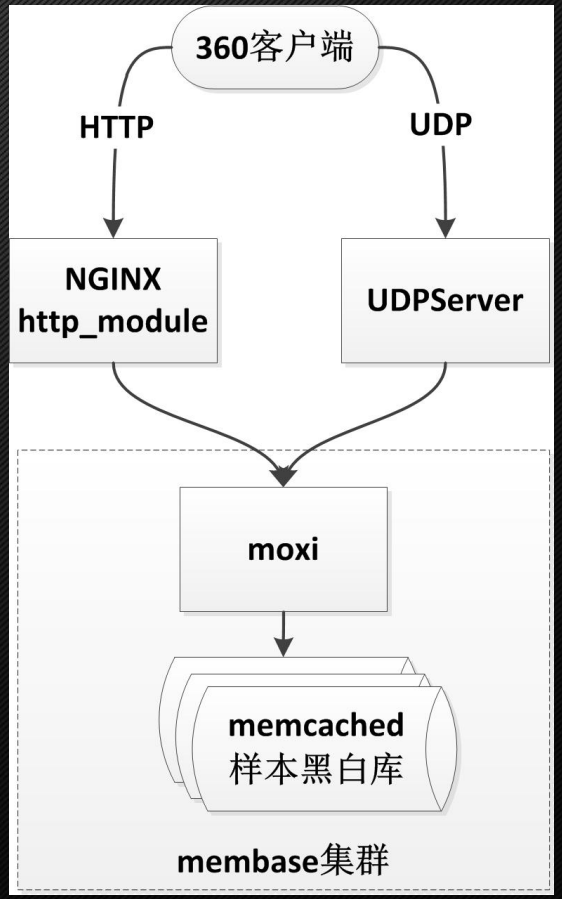


技术债务

- HTTP服务器依赖Nginx
  - ngx_pool/array/hash_t
- 全部移植到UDP服务中？
- 结论
  - 先解决问题
  - 两套完全独立的代码
- 技术债务
  - 无意的
  - 有意的

第3代架构：memcached集群

- 分布式集群
  - 容错
  - 横向扩展
- membase
  - 一致性hash
  - 与memcached协议兼容
  - 透明代理：moxi
  - Failover



当前亟待解决的问题

- 偿还技术债务
  - HTTP/UDP两套逻辑代码
- Membase引入问题
  - moxi带来的处理时延增加
  - moxi有bug，极端情况下会丢数据

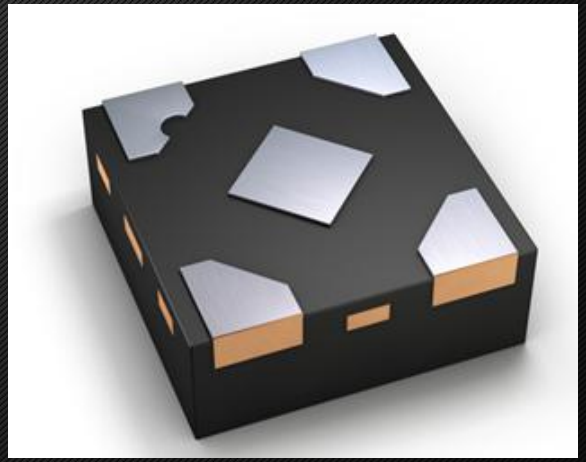
第4代架构：框架统一

- 逻辑抽象：处理输入得到输出

```
struct ngx_pool_s;
class Module {
public:
    virtual bool Initialize(const char* conf_path) = 0;
    virtual void Uninitialize() = 0;
    virtual int  Process(struct ngx_pool_s* pool,
                         const void* input_data,
                         size_t len,
                         std::string& result) = 0;
};
```

第4代架构：框架统一(续1)

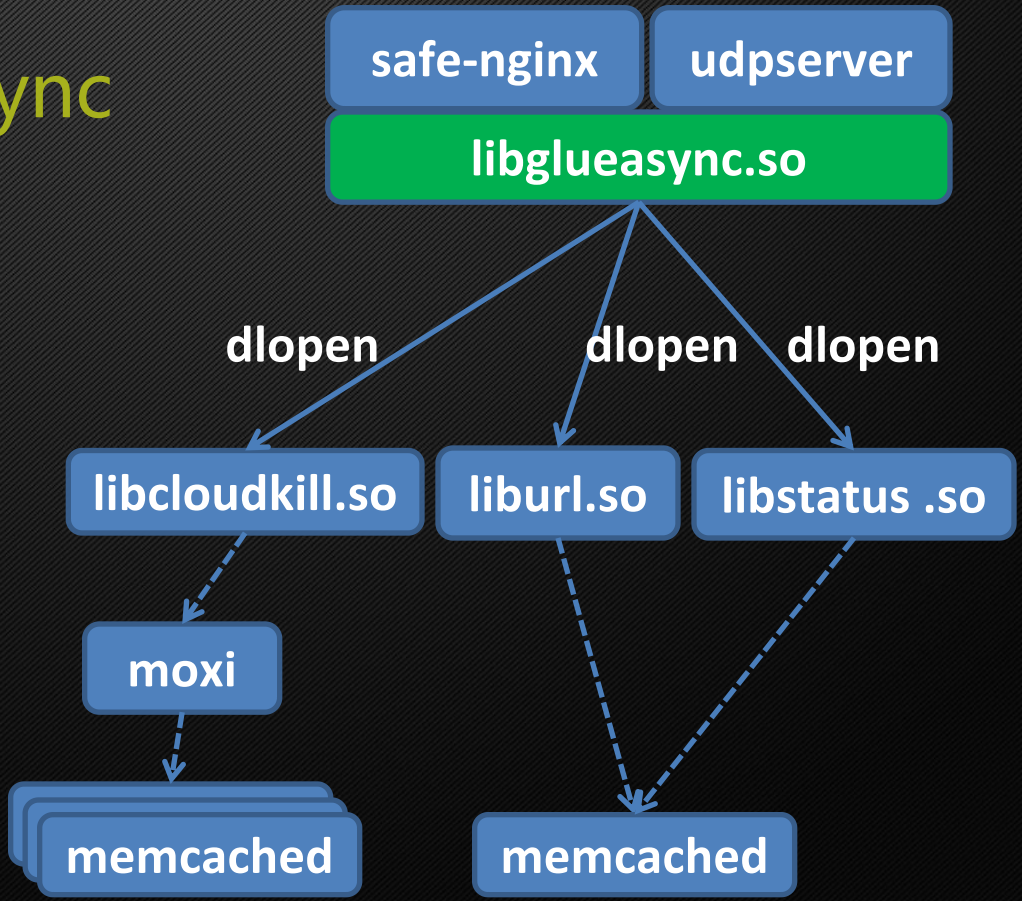
- 将处理代码封装到一个盒子(so)中



```
#ifdef __cplusplus
extern "C" {
#endif
    void* CreateModule();
    void  DestroyModule(void* m);
#ifdef __cplusplus
}
#endif
```

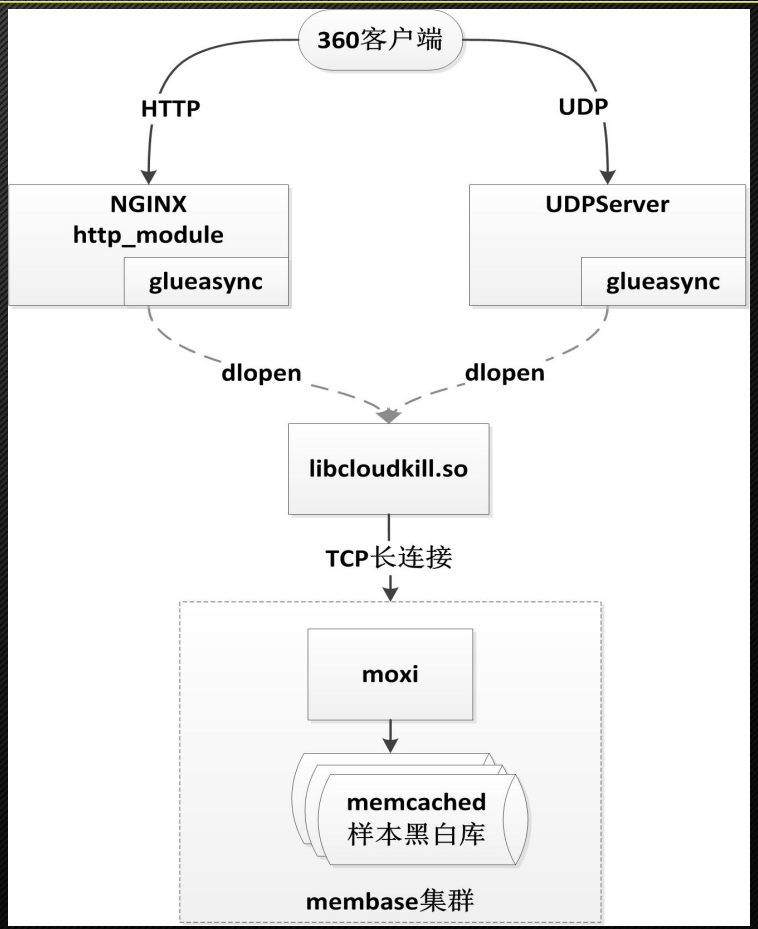
第4代架构：框架统一(续2)

- 透明中间层libglueasync
  - nginx http_module
  - udpserver
  - 业务模块管理
  - 统一加解密



第4代架构：框架统一(续3)

- 移植 ngx_pool_t
  - qh_pool_t
- 用STL替换ngx数据结构
  - ngx_array_t
  - ngx_hash_t
- 业务逻辑合一
  - libcloudkill.so

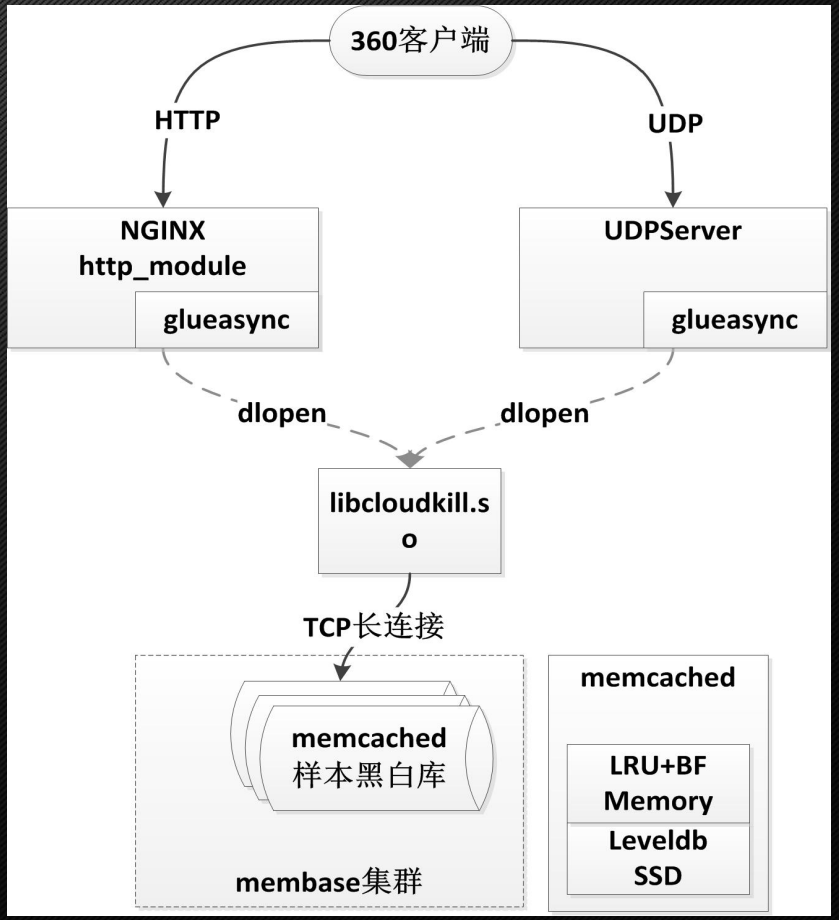


第5代架构：存储持久化

- 样本数据快速增长
  - 扩大集群规模？
  - 持久化改造？
- 数据调研
  - 90%请求是流行样本
- 持久化
  - 热数据在内存中：Cache

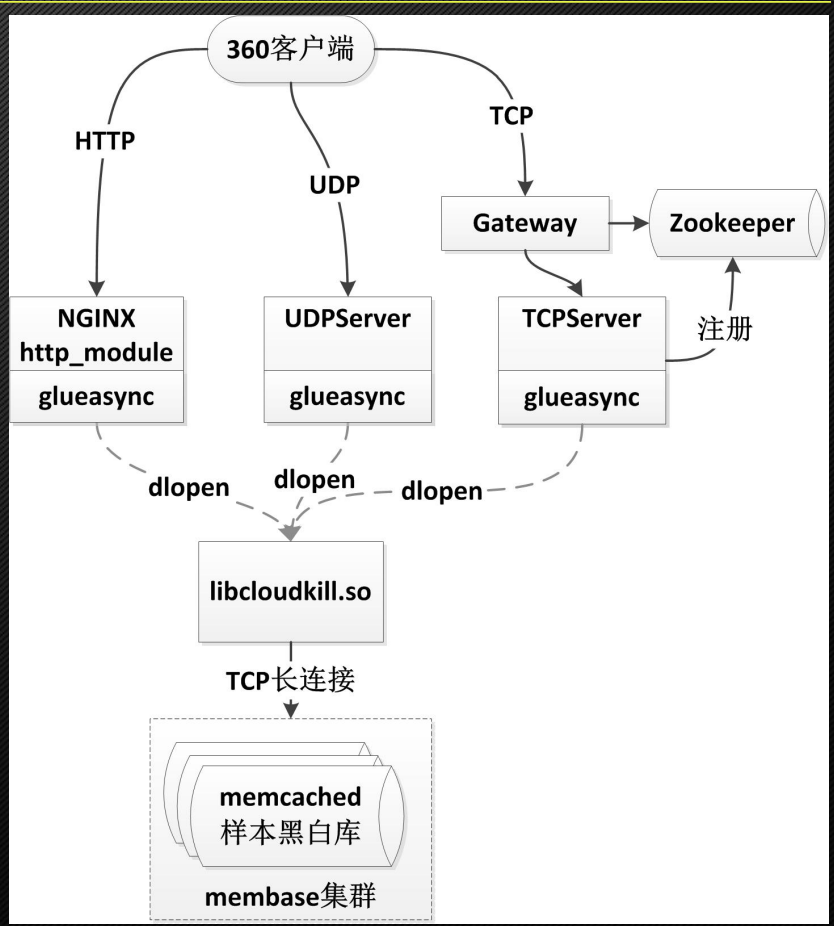
第5代架构：存储持久化(续)

- 持久化改造
  - LRU
  - BloomFilter
  - LevelDB
- 运维简单，数据恢复快
- moxi 问题：丢数、时延
  - 临时策略：增加白名单cache
  - 最终策略：彻底去掉



第6代架构：长连接

- 当前的问题
  - UDP丢包
  - HTTP短连接、重
- 新的需求
  - 任务系统、经验系统
  - 消息推送
- 方案：TCP长连接
- 1秒内网络请求成功率
  - 60% -> 86%



新的问题

- 多进程框架
- 字典文件更新
  - Reload进程
- 三组进程Reload
  - CPU负载高（丢请求）
  - 长连接（如何优雅断开？）

第7代架构：Trident

- 一个框架统一接入
  - TCP、HTTP、UDP
- 多进程 -> 多线程
- 同步 -> 异步
- 资源按需Reload
  - 双缓冲机制

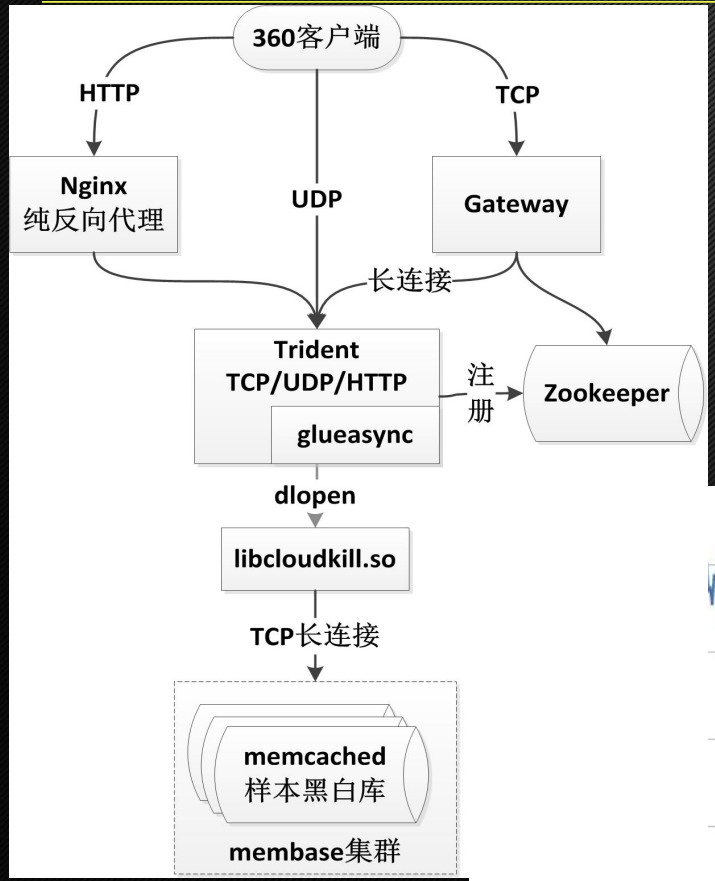
第7代架构：Trident双缓冲机制

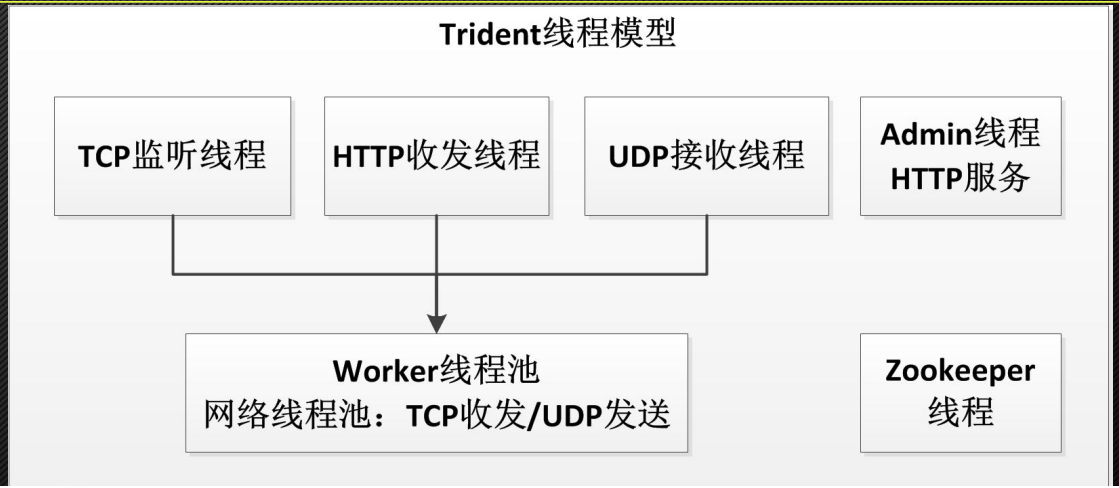
```
class Target : public RefObject {
 public:
  virtual bool Initialize(
        const string& conf) = 0;
  virtual ~Target() {}
};


class Manager
{
 public:
  void Add(const string& name,
           const string& conf,
           TargetCreator f);
  DoubleBuffering* Get(
       const string& name) const;
  bool Reload(const string& name,
              const string& conf);
 private:
  map<string, TargetCreator> creators_;
  map<string, DoubleBufferingPtr> dbufs_;
  Lock mutex_;
};
```

```
class DoubleBuffering {
 public:
  bool Reload(const string& conf) {
    TargetPtr t = creator_();
    if (!t->Initialize(conf)) {
      return false;
    }
    MutexLockGuard g(mutex_);
    current_ = t;
    return true;
  }
  TargetPtr Get() const {
    MutexLockGuard g(mutex_);
    TargetPtr t = current_;
    return t;
  }
 private:
  mutable Lock mutex_;
  TargetPtr current_;
  TargetCreator creator_;
};
```

第7代架构：Trident(续)





总结

- 业务需求与架构的平衡
- 没有完美的架构，只有能解决问题的架构

Thanks !

Q & A