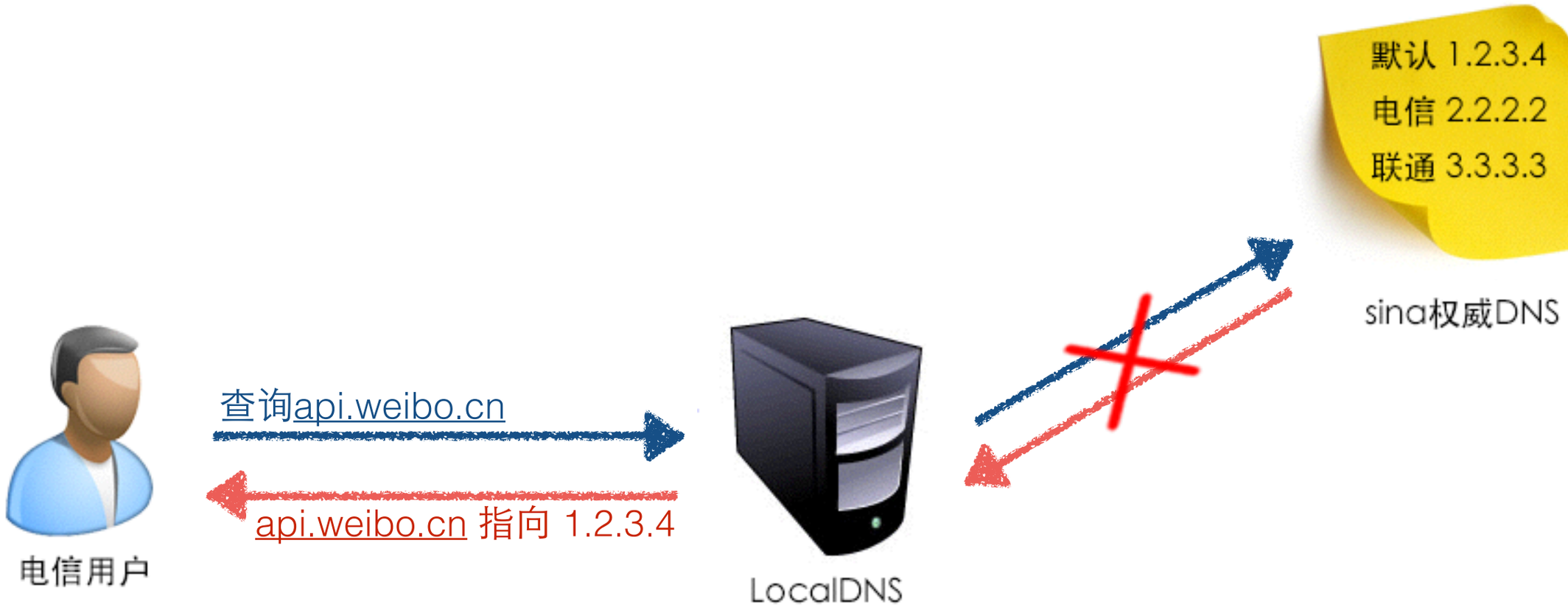
解读企业运维下的DNS劫持痛扰-HttpDNS Lib库

冯磊&赵星宇

一、问题根源：

- **LocalDNS**造成的用户访问异常可以归为下三类：
- 1）域名缓存
- 2）解析转发
- 3）**LocalDNS**递归出口**NAT**

域名缓存:



域名缓存很好理解，就是LocalDNS缓存了域名的解析结果，不向权威DNS发起递归.

- 保证用户访问流量在本网内消化：国内的各互联网接入运营商的带宽资源、网间结算费用、IDC机房分布、网内ICP资源分布等存在较大差异。为了保证网内用户的访问质量，同时减少跨网结算，运营商在网内搭建了内容缓存服务器，通过把域名强行指向内容缓存服务器的IP地址，就实现了把本地本网流量完全留在了本地的目的。

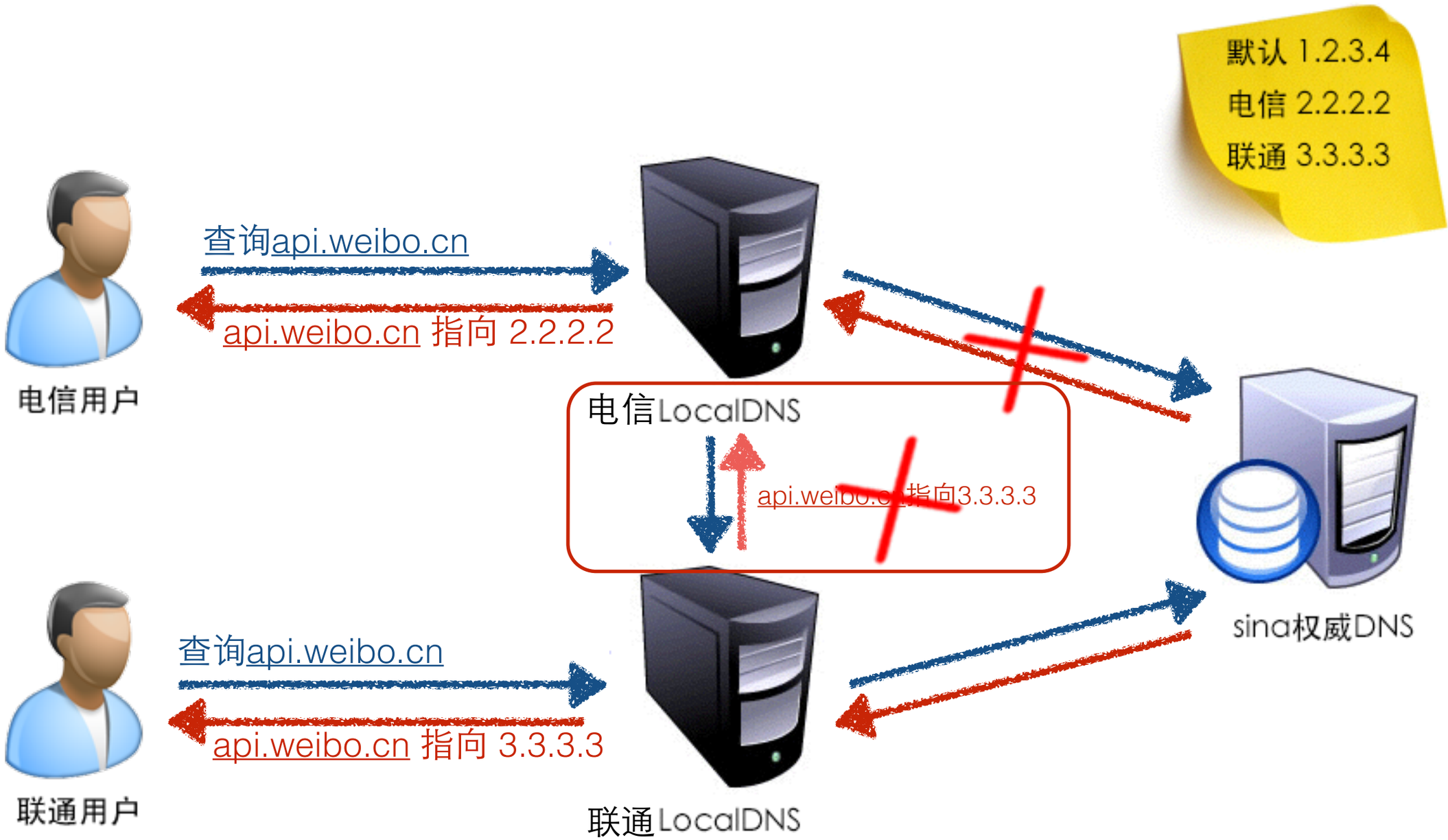
- 推送广告：有部分LocalDNS会把部分域名解析结果的所指向的内容缓存，并替换成第三方广告联盟的广告。

- 这种类型的行为就是我们常说的域名缓存，域名缓存会导致用户产生以下的访问异常：

- A、仅对80端口的http服务做了缓存，如果域名是通过https协议或其它端口提供服务的，用户访问就会出现失败。比如支付服务、游戏通过指定端口连接connect server服务等。

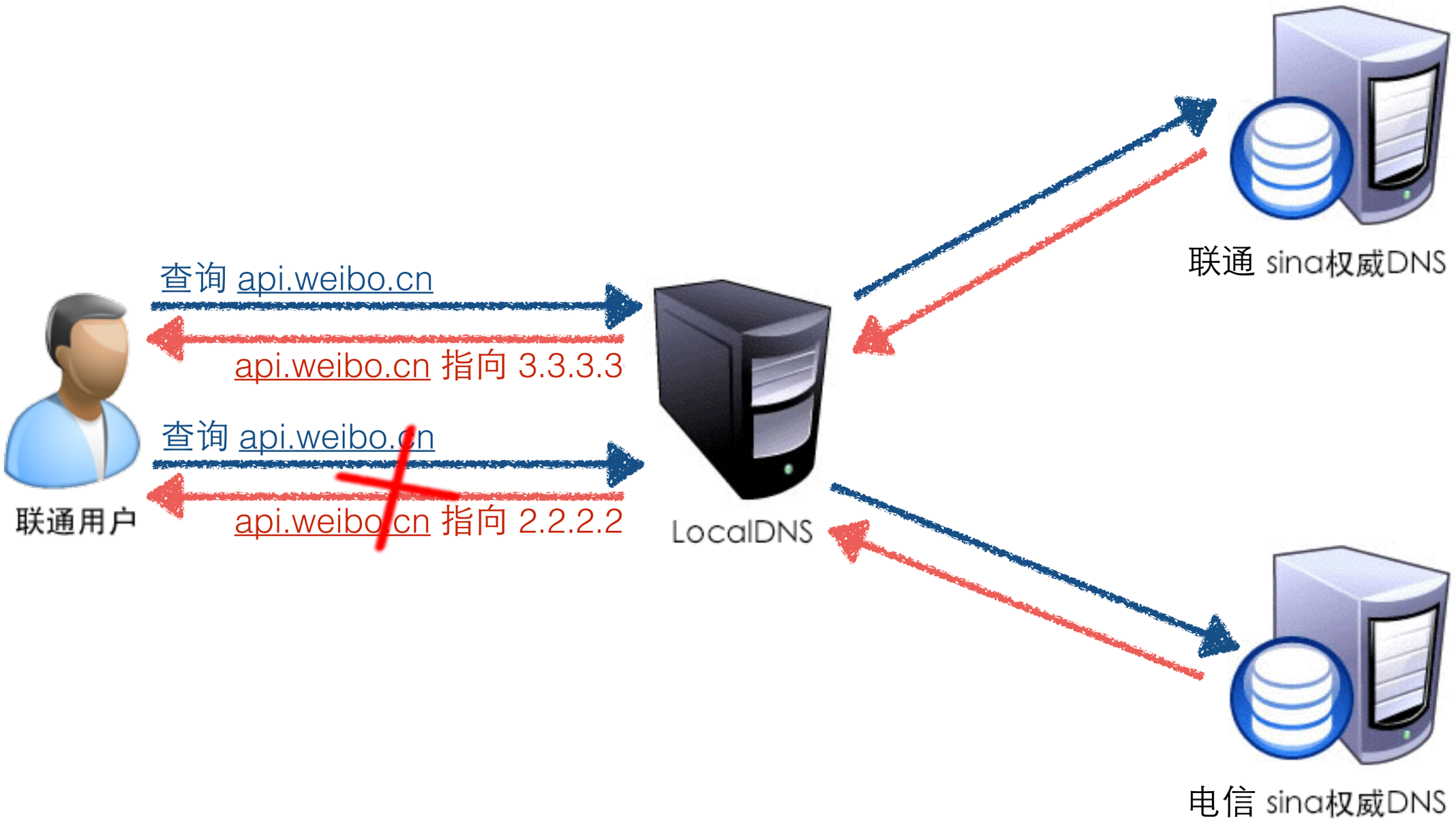
- B、缓存服务器的运维水平参差不齐，时有出现缓存服务器故障导致用户访问异常的问题。

解析转发


默认 1.2.3.4
电信 2.2.2.2
联通 3.3.3.3
查询api.weibo.cn
api.weibo.cn 指向 2.2.2.2
电信用户
电信LocalDNS
api.weibo.cn指向3.3.3.3
sina权威DNS
查询api.weibo.cn
api.weibo.cn 指向 3.3.3.3
联通用户
联通LocalDNS

- 而部分小运营商为了节省资源，就直接将解析请求转发到了其它运营的递归LocalDNS上去了
- 这样的直接后果就是权威DNS收到的域名解析请求的来源IP就成了其它运营商的IP，最终导致用户流量被导向了错误的IDC，用户访问变慢。

LocalDNS递归出口NAT



- LocalDNS递归出口NAT指的是运营商的LocalDNS按照标准的DNS协议进行递归，但是因为在网络上存在多出口且配置了目标路由NAT，结果导致LocalDNS最终进行递归解析的时候的出口IP就有概率不为本网的IP地址

- 这样的直接后果就是GSLB DNS收到的域名解析请求的来源IP还是成了其它运营商的IP，最终导致用户流量被导向了错误的IDC，用户访问变慢。

二、现有的解决方案及存在的问题

- 实时监控+商务推动
- 绕过自动分配DNS，使用114dns或Google public DNS
- 完全抛弃域名，自建链接进行流量调度

实时监控+商务推动

- 这种方案是目前圈子内的运营团队一直在使用的方案。这种方案就是周期比较长，毕竟通过行政手段来推动运营商来解决这个问题是比较耗时的。另外我们通过大数据分析，得出的结论是Top 3的问题用户均为移动互联网用户。对于这部分用户，我们有什么技术手段可以解决以上的问题呢

使用114dns或Google DNS

- 这个方案看上去很美好，114dns是国内最大的中立缓存DNS，而Google又是不作恶理念的互联网工程帝国巨鳄，权威DNS支持edns功能，能直接识别使用Google publicDNS解析该域名的用户的IP地址，不会出现流量调度失效。但是问题来了：

- （1）如何在用户端构造域名请求：对于PC端的用户来说，构造一个标准的DNS请求包并不算什么难事。但在移动端要向一个指定的LocalDNS上发送标准的DNS请求包，而且要兼容各种iOS和android的版本的话，技术上是可行的，只是兼容的成本会很高。

- （2）推动用户修改配置极高：如果要推动用户手动修改PC的DNS配置的话，在PC端和手机客户端的WiFI下面还算勉强可行。但是要用户修改在移动互联网环境下的DNS配置，其难度不言而喻。

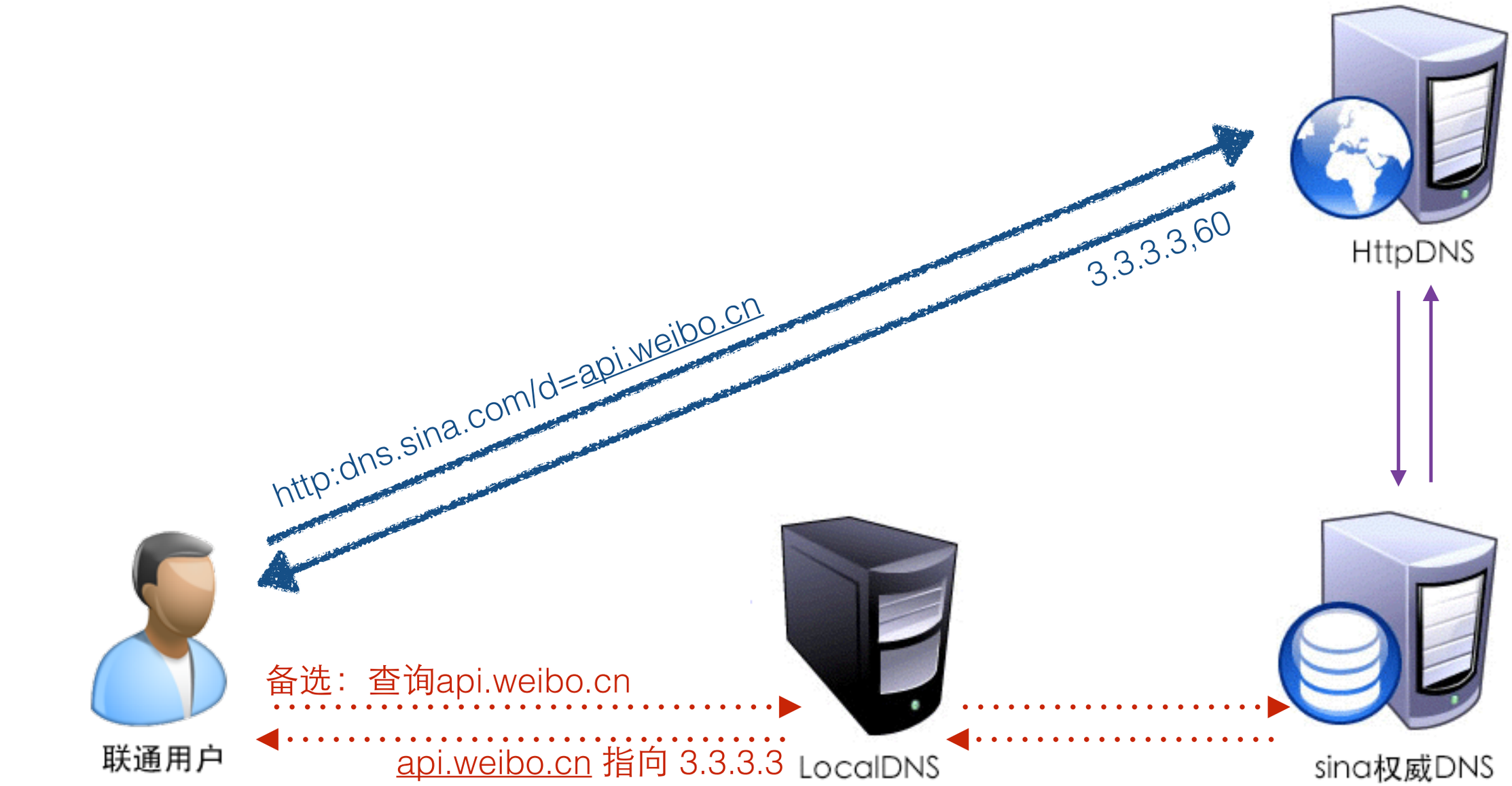
完全抛弃域名，进行流量调度

- 如果要采用这种这种方案的话，首先你就得要拿到一份准确的IP地址库来判断用户的归属，然后再制定个协议来做调度，在然后对接入层做调度改造。这种方案和第2种方案一样，不是不能做，只是成本会比较高。

三、利用HttpDNS解决用户域名解析异常

- HttpDNS基本原理
- HttpDNS优势

HttpDNS基本原理



- HttpDNS的原理非常简单，主要有两步：

- A、客户端直接访问HttpDNS接口，获取业务在域名配置管理系统上配置的访问延迟最优的IP。（基于容灾考虑，还是保留次选使用运营商LocalDNS解析域名的方式）

- B、客户端使用获取到的IP，直接往此IP发送业务协议请求。以Http请求为例，通过在header中指定host字段，向HttpDNS返回的IP发送标准的Http请求即可。

HttpDNS优势

- 从原理上来讲，HttpDNS只是将域名解析的协议由DNS协议换成了Http协议，并不复杂。但是这一微小的转换，却带来了无数的收益:

- A、根治域名解析异常：由于绕过了运营商的LocalDNS，用户解析域名的请求通过Http协议直接打到了HttpDNS服务器IP上，用户在客户端的域名解析请求将不会遭受到域名解析异常的困扰。

- B、调度精准：HttpDNS能直接获取到用户IP，避免了DNS出口IP和业务出口IP不同网段问题。

- C、实现成本低廉: 目前Android版本SDK已经开源，IOS版本最后测试阶段。

- D、扩展性强：SDK可接入多方HttpDND平台，甚至可以直接接入公共DNS服务器。

已知使用HttpDNS

- DNSPod D+ 移动解析服务D+ （免费服务）
- 114.114.114.114/d?dn=domain（没有正式开放）
- 阿里（内部使用）
- 新浪（内部使用）

智能 HttpDNS Lib库

Git：https://github.com/SinaMSRE/HTTPDNSLib

- HttpDNS主要解决三类问题:
- 1）LocalDNS劫持
- 2）平均访问延迟下降
- 3）用户连接失败率下降

- 用户平均访问延迟下降超过10%
- 访问失败率下降了超过五分之一

LocalDNS劫持:

- 由于HttpDNS是通过ip直接请求http获取服务器A记录地址，不存在向本地运营商询问domain解析过程，所以从根本避免了劫持问题。（对于http内容tcp/ip层劫持，可以使用验证因子或者数据加密等方式来保证传输数据的可信度）

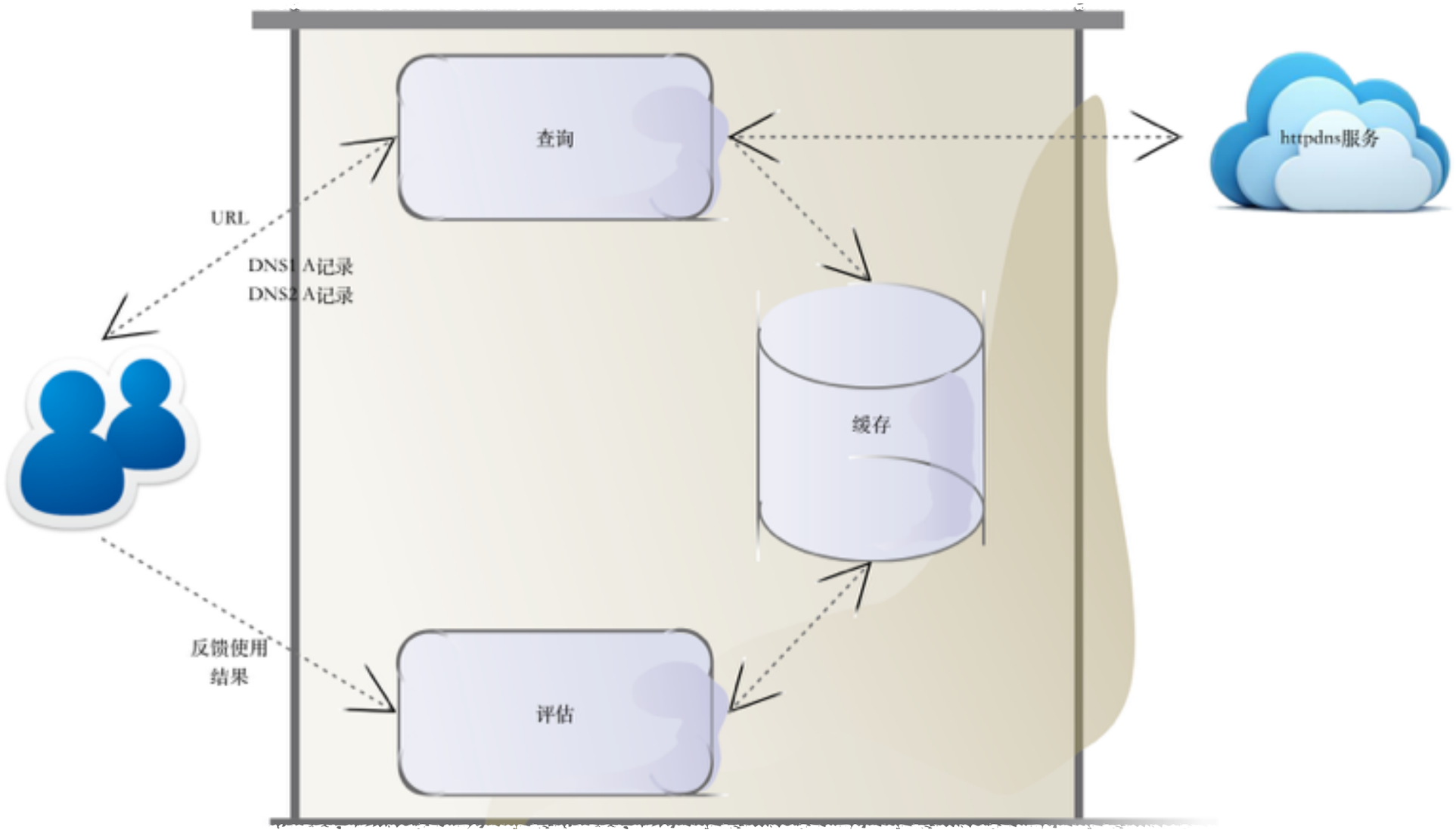
平均访问延迟下降

- 由于是ip直接访问省掉了一次domain解析过程，（即使系统有缓存速度也会稍快一些‘毫秒级’）通过智能算法排序后找到最快节点进行访问。

用户连接失败率下降

- 通过算法降低以往失败率过高的服务器排序，通过时间近期访问过的数据提高服务器排序，通过历史访问成功记录提高服务器排序。如果ip(a)访问错误，在下一次返回ip(b)或者ip(c) 排序后的记录。（LocalDNS很可能在一个ttl时间内（或多个ttl）都是返回一个A记录

HttpDNS本地Lib库交互流程



HttpDNSLib库组成

- 查询模块
- 数据模块
- 评估模块

查询模块

- 检测本地是否有相应的域名缓存
- 如果没有 则从本地LocalDNS获取然后从httpdns更新domain记录
- 有数据则检测是否过期 已过期则更新记录，返回LocalDNS 记录， 未过期则直接返回缓存层数据。
- 从HttpDNS 接口查询本次app开启后使用过的domain 记录定时访问，更新内存缓存，数据库缓存等记录

数据模块

- 根据sp（或wifi名）缓存域名信息
- 根据sp（或wifi名）缓存服务器ip信息、优先级
- 记录服务器ip每次请求成功数、错误数
- 记录服务器ip最后成功访问时间、最后测速
- 添加 内存 -》数据库 之间的缓存层

评估模块

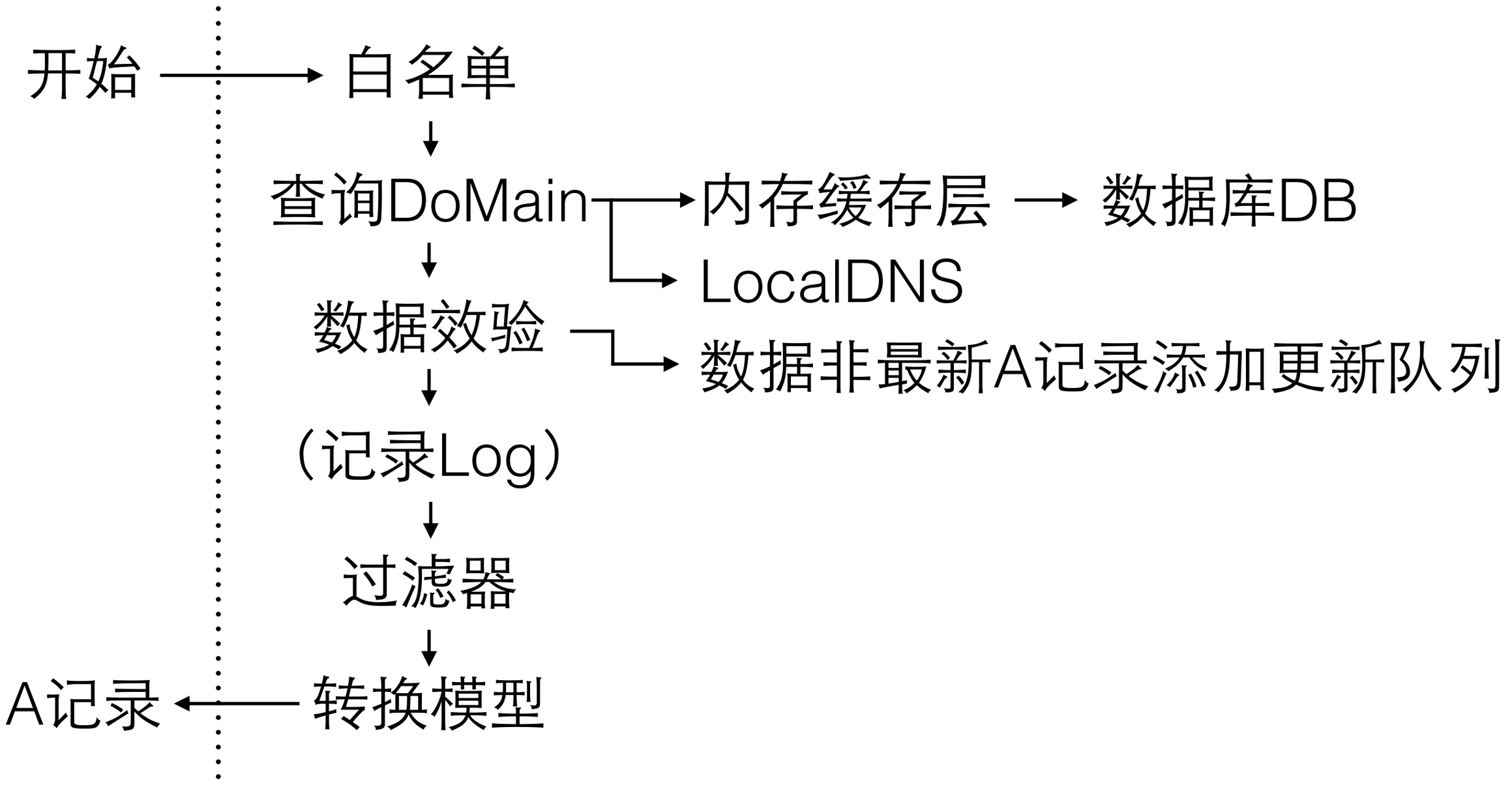
- 根据本地数据，对一组IP排序（智能排序）
- 给HttpDns服务端智能分配A记录提供数据依据
- 处理用户反馈回来的请求明细，入库
- 针对用户反馈是失败请求，进行分析上报预警

智能排序模块

- 本次测速 - 对ip组的每个ip测速打分
- 官方推荐 - HttpDns接口 A记录中返回的优先级
- 历史成功 - 该ip历史访问成功次数
- 历史错误 - 该ip历史访问错误次数
- 最后成功时间 - 该ip最后一次成功时间，阈值24小时

Lib库主要交互流程

获取A记录交互流程


开始
白名单
查询DoMain
内存缓存层
数据库DB
LocalDNS
数据效验
数据非最新A记录添加更新队列
（记录Log）
过滤器
转换模型
A记录

定时器逻辑

定时器开始

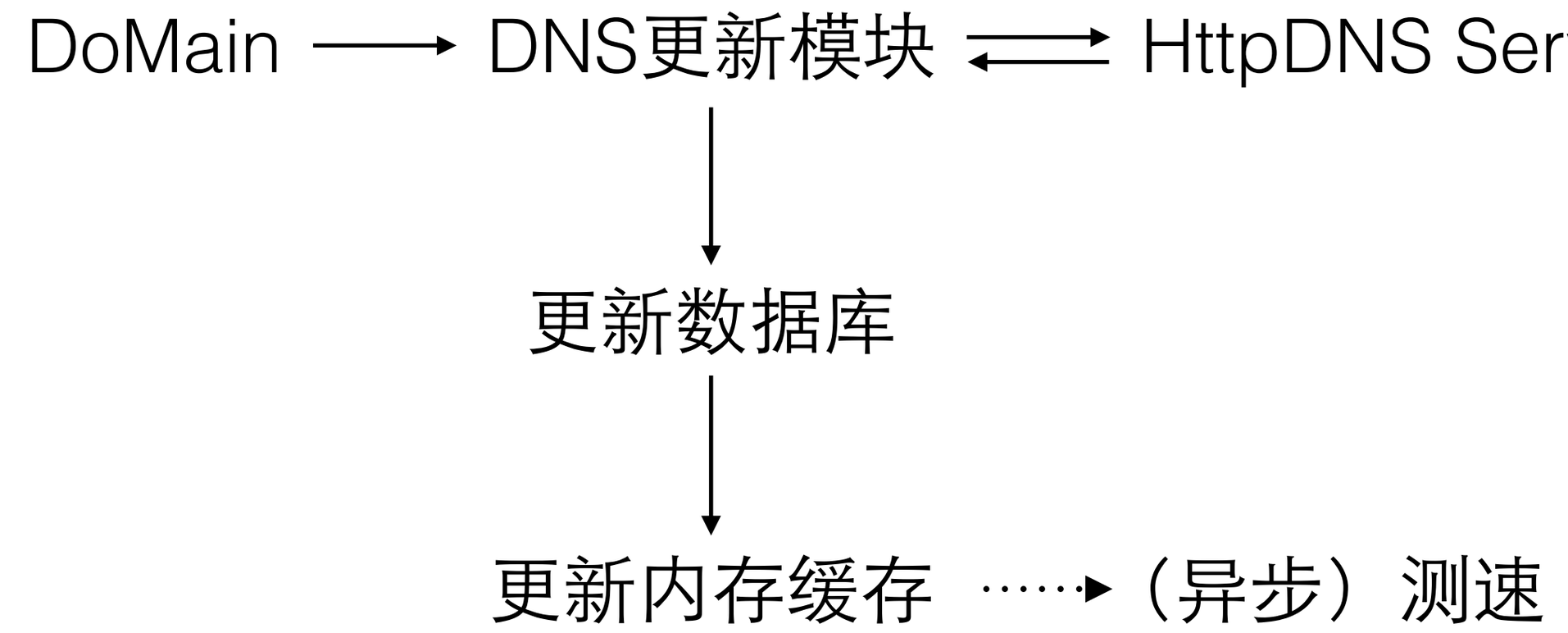
1.更新过期DoMain数据

2.服务器IP测速逻辑

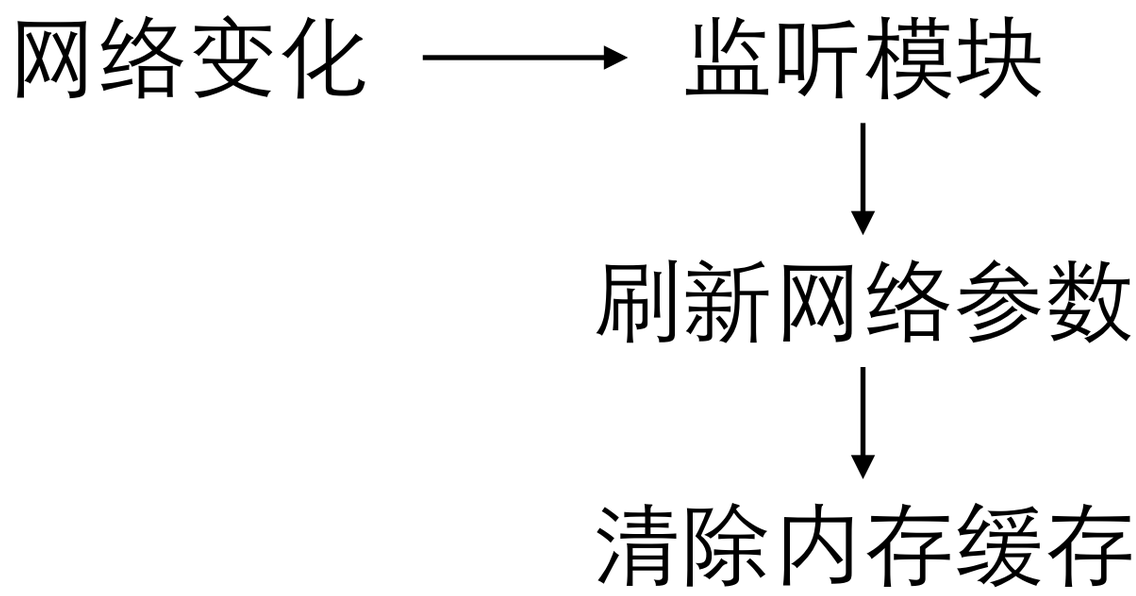
3.日志管理模块（日志上传、清理等）

定时器结束

A记录更新策略



网络变化策略



Lib库主要功能点

初始化

- 初始化SDK配置
- 初始化网络状态信息
- 注册网络变化监听
- 预加载DoMain记录（用户调用）

DNS更新A记录-模块

- 自定义DNS服务器（可扩展智能评估）
- DNSPOD
- UDPDNS查询（114 \ 8.8.8.8 等其他中立DNS商）
- LocalDNS

测速模块

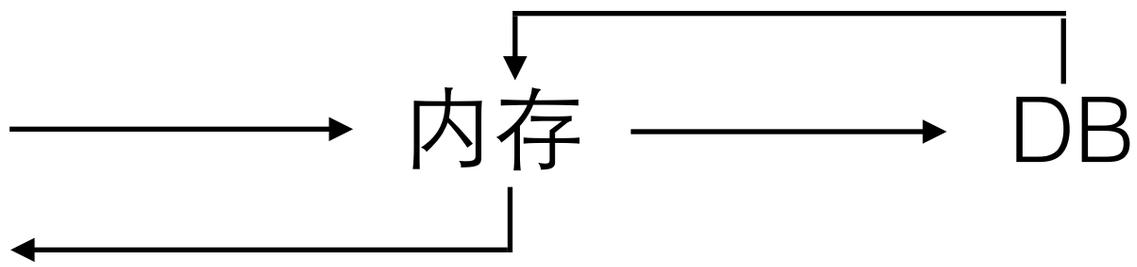
- Socket 80 （访问HttpServer80端口，建立连接）
- Ping（发送ICMP包，获取RTT值）

排序模块（插件因子）

- 速度（基于ping 或者 socket 得出的 RTT ）
- 优先级（根据监控系统、链路反馈、负载均衡得出）
- 成功次数（每台服务器历史使用成功次数）
- 错误次数（每台服务器历史使用失败次数）
- 最后成功时间（越接近现在时间，分数越高）

查询模块

- 内存层
- 本地数据库
- LocalDNS



配置模块 - 开关

- HttpDNSLib库
- 智能排序算法
- 自定义HttpDNS
  - DNSPod （HttpDNS API 接口）
  - UDP （DNS Server 获取数据）
  - 云端更新配置文件
  - Debug调试信息

配置模块 - 参数

- 自定义HttpDNS API 接口地址
- DNS Pod 相关配置（Api接口、企业KEY、企业ID）
- UDP请求DNS服务器IP地址
- 域名白名单（NULL时 支持所有域名）
- 日志采样率、日志上传时间
- 定时器轮询间隔时间
- IP测速间隔时间
- A记录 TTL 差值时间
- 排序算法的插件权重值

未来

- 智能动态节点加速
- 监控系统整合，动态节点切换
- 故障负载均衡切换
- 国际网络接入点路由优化
- 网络库（HTTP2整合）

一路走来，掉进去的坑！

更新A记录并发、测速机制变革、插件化改革。

Thanks

fenglei1@staff.sina.com.cn
xingyu10@staff.weibo.com

张杰 微博:@木泽水
王春生 微博:@平凡的香草
胡波 微博:@胡波_
韩超 微博:@超朝炒觥
赵星宇 微博:@淘淘不逃008
聂钰 微博:@古夜
黄振栋 微博:@BG2BKK
冯磊 微博:@冯磊424

1、手机网络从3G 切换到 Wifi下处理了什么?

- NetworkStateReceiver类来监听网络是否发生变化，在网络有变化的时候，会刷新 NetworkManager类中的网络环境，在客户端内如果是手机网络可以知道网络类型（2G、3G、4G）也可以知道当前SP（移动、联通、电信），如果是Wifi网络环境可以知道SSID（wifi名字）在刷新网络环境后，会重新查询缓存内是否有当前链路下的最优A记录，如果没有则从LocalDNS获取第一次，然后马上更新httpdns记录。

2、网络发生变化后，返回的A记录还一样么?

- 数据库中缓存的数据，是根据当前sp来缓存的，也就是说当自身网络环境变化后，返回的a记录是不一样的 。手机网络下会根据当前sp来缓存 a记录服务器ip，如果是wifi网络环境下 根据当前ssid来缓存a记录，因为wifi环境下库自己没有办法明确判断出自己的运营商，但相同的ssid不会发生频繁的网络运营商变化。 所以在wifi下请求回来的a记录直接关联ssid名字即可，即使wifi sp发生变化，最多延迟一个ttl时间就更新成最新的a记录了。

3、怎样进行测速?

- 在从HttpDNS获取回来a记录的时候进行测速，测速的方式有两种： ping和空的http请求。考虑倒有些服务器不支持ping来进行链路质量评判，可以使用空的http请求，仅仅是两个http头的流量开销，而ping的方式流量开销就更小了。 这个功能可以在库中自己配置。这里的测速其实是模拟首保接收的时间来做的， 同时对于流量控制严格的可以在库中配置测速频繁度，比如一台服务器在5分钟内有过测速记录则不进行测速。

4、域名ttl刚刚过期，库还没有从HttpDNS拉取回来数据怎么办?

- ttl过期的前10秒去请求数据，在ttl过期后的10秒内库也认为当前a记录是有效的，会给你直接返回。

5、lib库目前只能使用 dnspod 服务商么？ 支持dnspod 企业版本么?

- 目前库可以支持自定义的 HttpDNS api接口， 只需要实现IHttpDns接口类即可，在配置了dnspod企业key和id 的时候自动启用企业版本加密传输，支持企业版本。

6、使用这个库会不会降低应用请求网络的访问速度?

- 从目前的测试数据来看是不会的，HttpDNS库返回a记录的时间平均在5毫秒以内，有时会出现内存缓存中没有该域名记录，数据库中也没有的时候会从LocalDNS获取a记录，时间会稍长一些
- 一旦从LocalDNS获取后，会缓存倒内存中，在HttpDNS获取数据后会更新内存中得domain记录。 从库中获取a记录会比从LocalDNS获取a记录快一些。 在访问网络的时候由于是使用ip直链，可以起到一些加速效果，lib库获取domainA记录 + ip直接访问服务器 耗时小于 直接域名请求服务器。相关数据图片

数据报表

全部任务

加速：7个

延迟：3个

Lib库耗时

最快：1毫秒

最慢：11毫秒

任务速度对比

最大速度提升：143毫秒

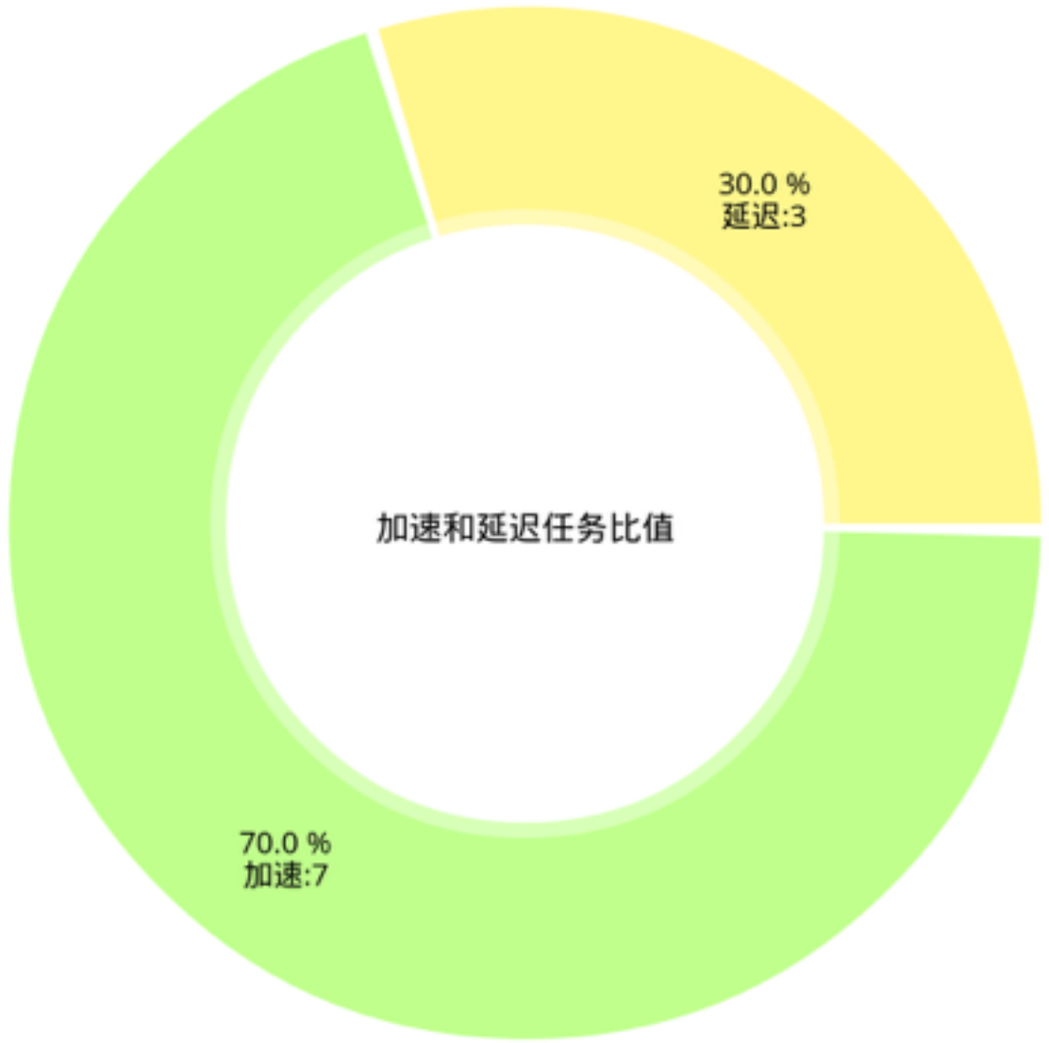
全局速度提升：816毫秒

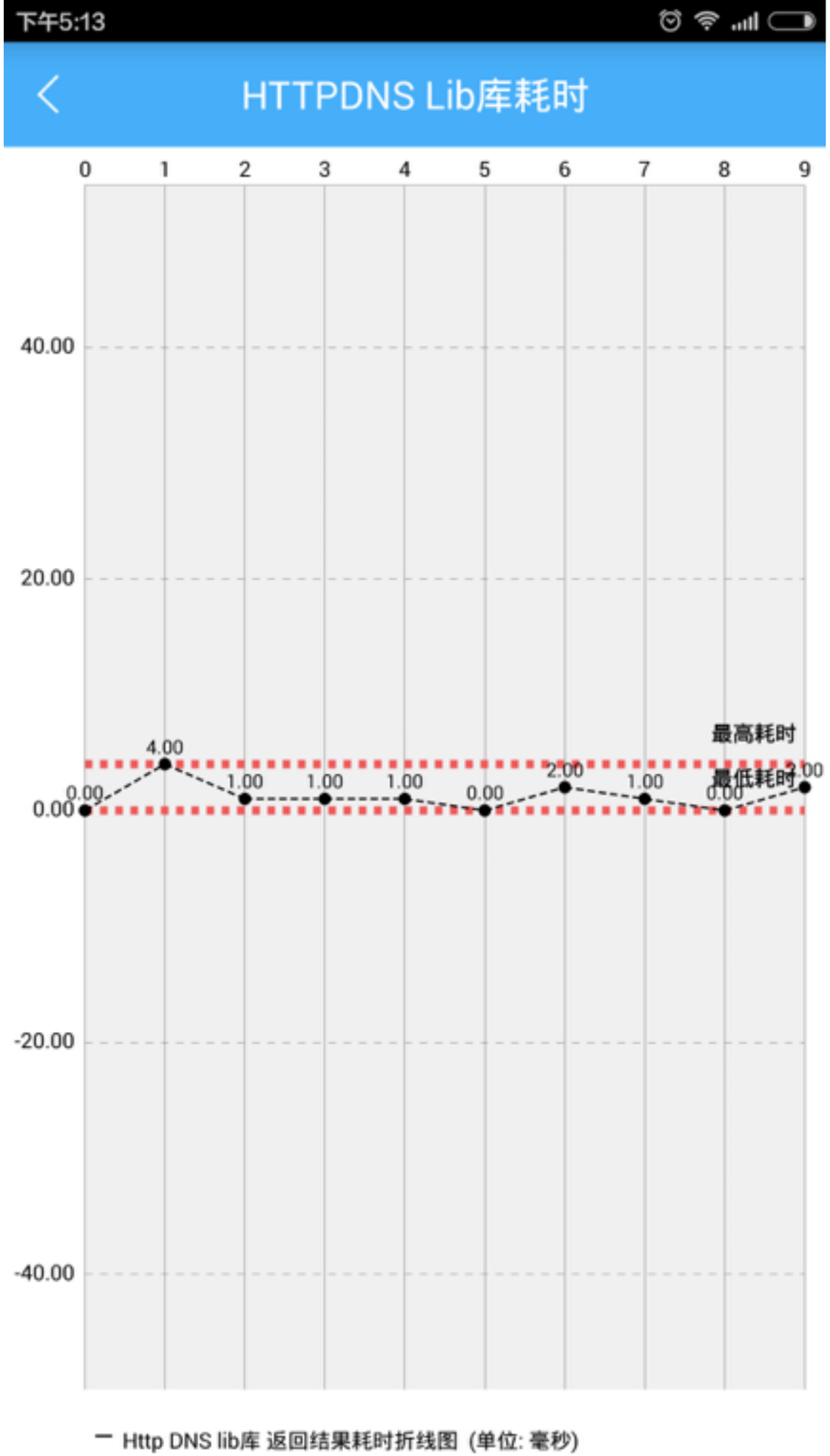
服务器速度对比

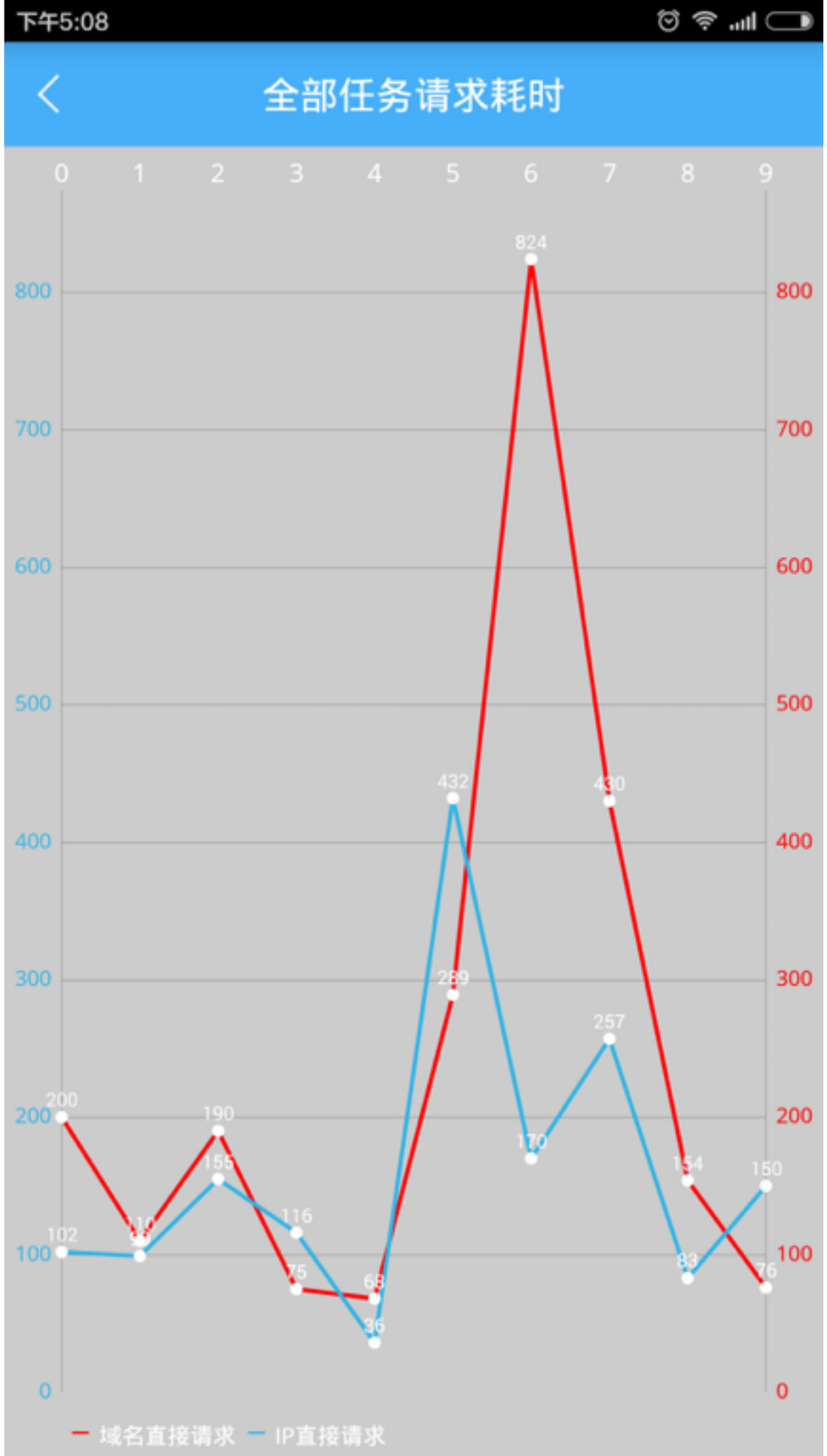
最快服务器：111.161.68.61

最慢服务器：202.108.7.239

全部任务速度比值




下午5:13
HTTPDNS Lib库耗时
0 1 2 3 4 5 6 7 8 9
40.00
20.00
0.00
-20.00
-40.00
最高耗时
最低耗时
4.00
0.00
1.00
1.00
1.00
0.00
2.00
1.00
0.00
2.00
Http DNS lib库 返回结果耗时折线图 (单位: 毫秒)


下午5:08
全部任务请求耗时
0 1 2 3 4 5 6 7 8 9
800
700
600
500
400
300
200
100
0
824
432
430
289
257
200
190
170
155
154
150
116
102
83
76
75
68
36
域名直接请求
IP直接请求

7、lib库里面访问网络使用的是哪个网络库？ json库用的是哪个?

- 考虑到该库的轻量级，使用的是android系统的org.apache.http.client.HttpClient库访问网络，如果需要切换到工程在使用的网络库可以实现INetworkRequests 接口即可切换网络库。 json解析使用的也是android系统自带的org.json.JSONObject如有需求切换json解析库，可直接实现IJsonParser接口即可切换。

8、使用该网络库会给我的app带来多大的体积增加?

- 目前该lib库没有引用任何外部的库文件。一切本着使用系统自身的api为原则，来保证库的轻量级，和兼容性。 目前lib库打包后70多k，代码在5千行左右。测试工程代码在6千行左右。

9、lib库的配置必须要通过修改源代码的方式来进行配置么?

- 任何参数都可以在库调用方配置，DNSCacheConfig类是整个库的配置文件。 并且支持云端动态更新配置 需要实现DNSCacheConfig.ConfigText_API 更新地址。 具体配置api接口请参考 dome工程中设置库的方式。

10、缓存domain记录是存储成文件还是数据库，或者android内部的一些存储方式?

- lib缓存数据是通过数据库存储的。SQLiteDatabase， 具体的表接口和sql语句请参考DBConstants 类文件。

11、 评估模块有什么功能?

- 评估模块目前由五个插件组成， 速度插件、推荐优先级插件、历史成功次数插件、历史错误数插件、最近一次成功时间插件 。 每一个a记录服务器ip，都会经过这五个插件进行评估排序后返回给使用者。 所有插件评估分值比重可以配置， 根据自己的需求以及不同的使用场景，调整出最合理的权重分配。

ID	IP	本次速度	优先级	成功次数	错误次数	最后访问时间
1	111.111.111.111	63	5	22	6	201503242358
2	222.222.222.222	27	3	28	0	201503241858
3	333.333.333.333	79	3	15	9	201503240758
4	444.444.444.444	18	2	62	1	201503232258
5	555.555.555.555	98	2	17	25	201503242258

本次测速Speed (40%)	25.2 10.8 31.6 7.2 39.2	历史成功次数 (10%)	3.52 4.48 2.4 9.92 2.72
官方推荐优先级 (30%)	30 18 18 12 12	历史错误次数 (10%)	2.4 -> 7.6 0 -> 10 3.6 -> 6.4 0.4 -> 9.6 10 -> 0
最后一次成功访问时间|阀值：24小时 当前时间：201502242358 (10%)	10 7.95 3.44 0 9.59	最后结果 (名次)	76.34 (1) 51.23 (4) 61.84 (3) 38.72 (5) 63.51 (2)

最终ip排序结果：1、5、3、2、4

12、速度插件具体算法?

- 比如速度插件评分体系， 满分100分， 那么有3个服务器ip， 1号服务器http请求耗时10毫秒， 2号服务器20毫秒， 3号服务器30毫秒。 那么经过插件计算后 1号服务器100分， 2号服务器50分， 3号服务器25分。

13、优先级插件又是什么?

- 如果是自定义的服务器，可以返回服务器优先级字段，该的字段代表推荐使用该服务器的权重， 比如该字段服务端可以和监控系统结合起来，甚至是用来分流。 相应的权重值， 也会算出来不同的分值。

14、历史成功次数插件是什么？历史错误次数插件是什么？

- 在当前sp当前链路下， 会记录访问过的该服务器ip的成功次数， 成功次数越多认为该服务器相对稳定。 会在排序的时候根据权重比值进行影响最终排序结果， 该插件权重不建议过高。 同理历史错误插件也是记录当前链路下服务器出过错误的次数，次数越高认为越不稳定。 排序尽量靠后。 同样该插件权重不建议过高。

15、最后一次成功时间?

- 如果该服务器在 很近的时间内访问过，那么评估系统则认为他链路是通的，则会给一个分值， 越接近现在的时间的服务器 分值越高。 24小时以前访问的分值为0 。

16、评估插件就这五个吗?

- 该五个插件属于抛砖引玉， 可以自定义插件 只需要实现 IPlugIn 接口即可。 所有的插件启用和停止都在配置文件中可以修改，以及配置每个插件的权重比。

17、智能评估一定会带来好的效果吗?

- 首先httpdns返回的a记录已经是 经过当前地域和当前sp返回的最优记录结果集， 至少不会降低效率。

18、我可以不使用智能评估模块么?

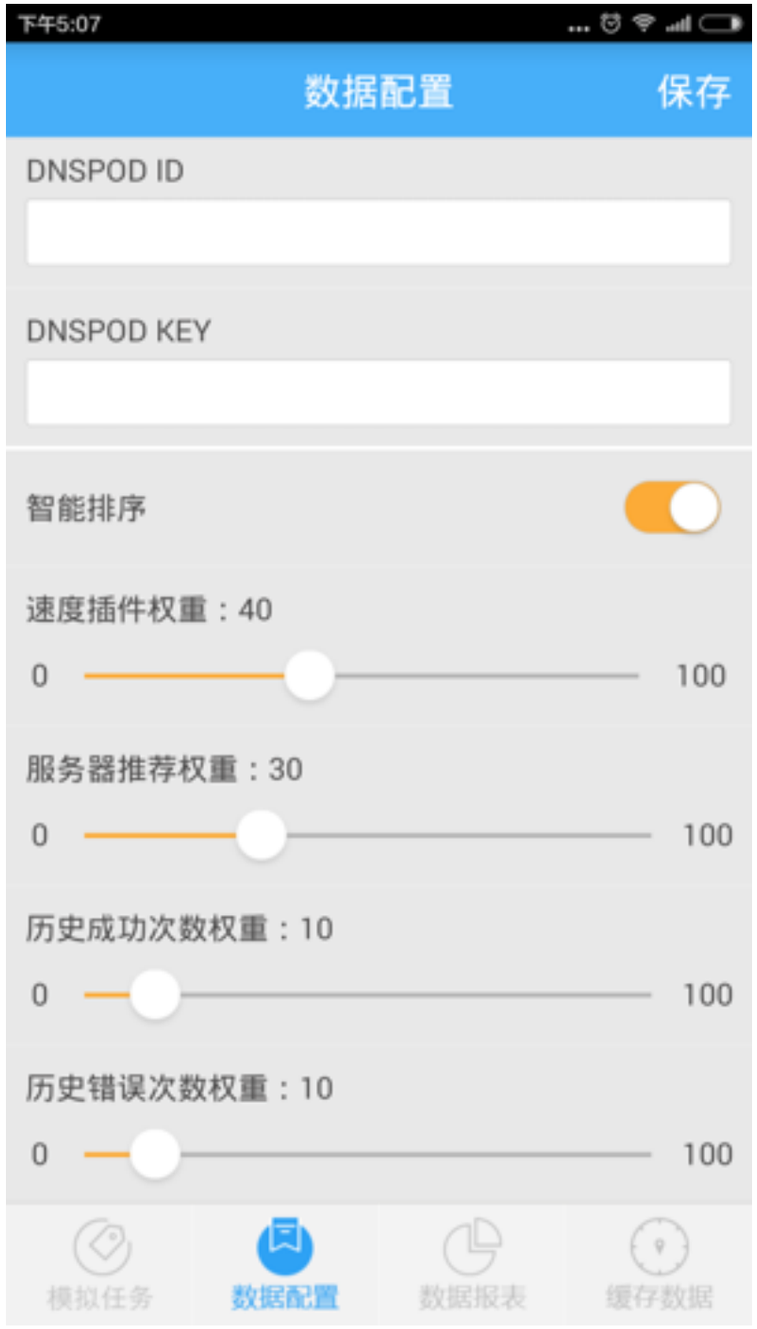
- 可以的在配置文件中关掉智能评估即可，具体代码参照demo即可。 关掉智能评估模块后，会在多个a记录中随机排序返回。

19、该Lib库块兼容性如何?

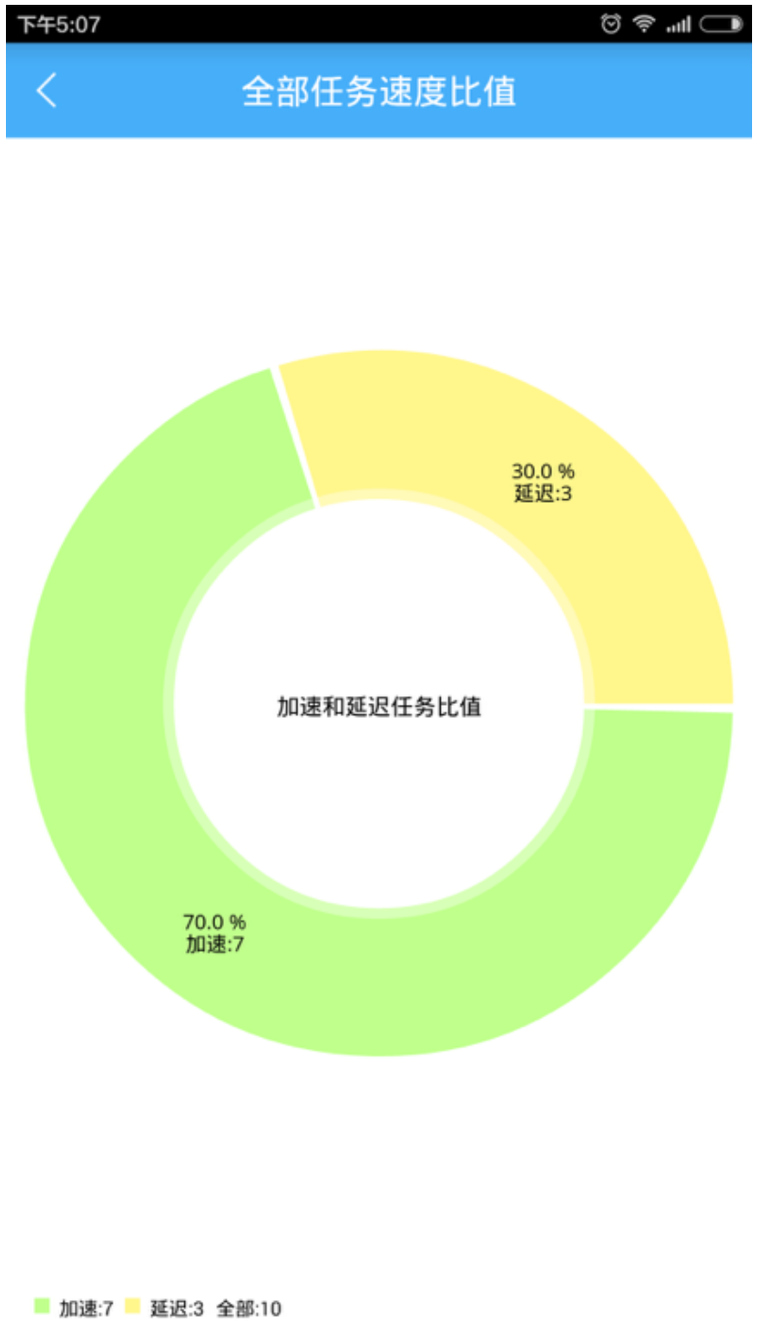
- 使用testin兼容性测试 测试兼容性结果：99.49%。Android平台全部由java代码开发，没有使用任何特殊特性，覆盖全部系统版本。

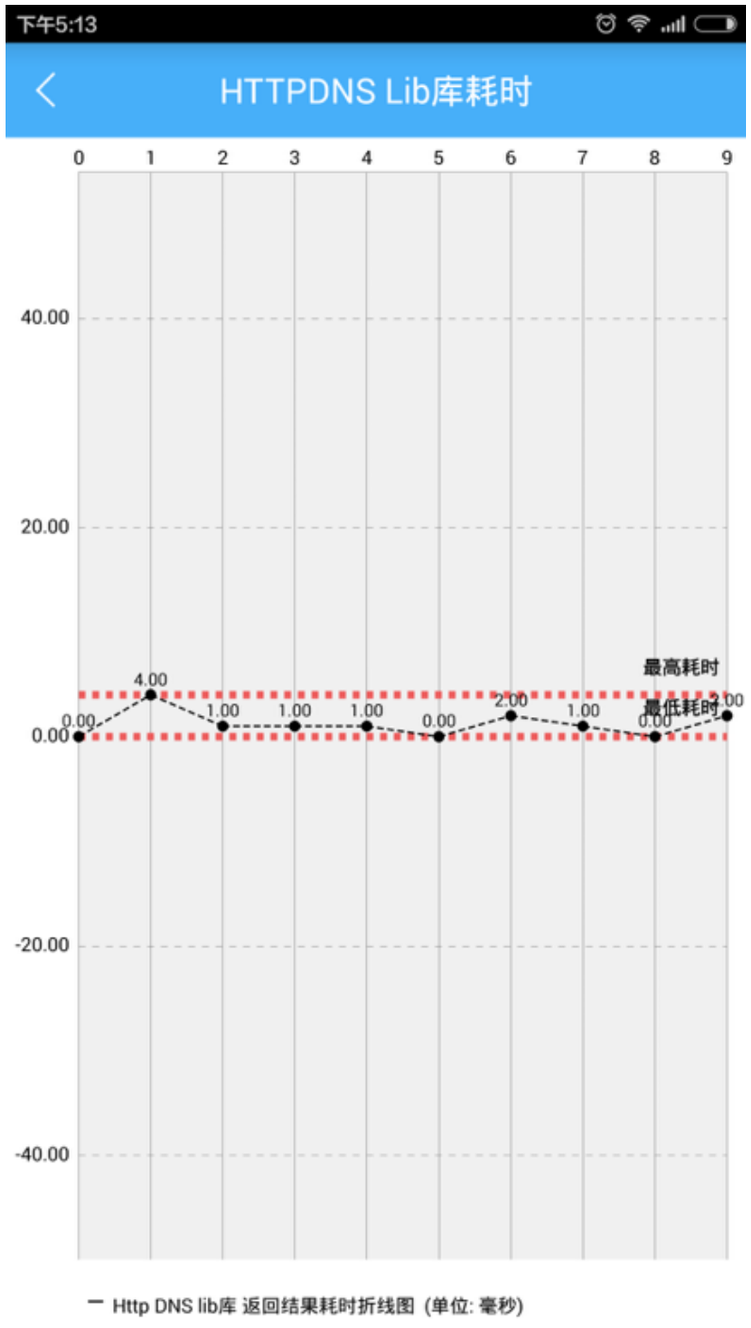
测试工程效果图

模拟了客户端访问http请求，
分别标识了每个任务的详细信息。

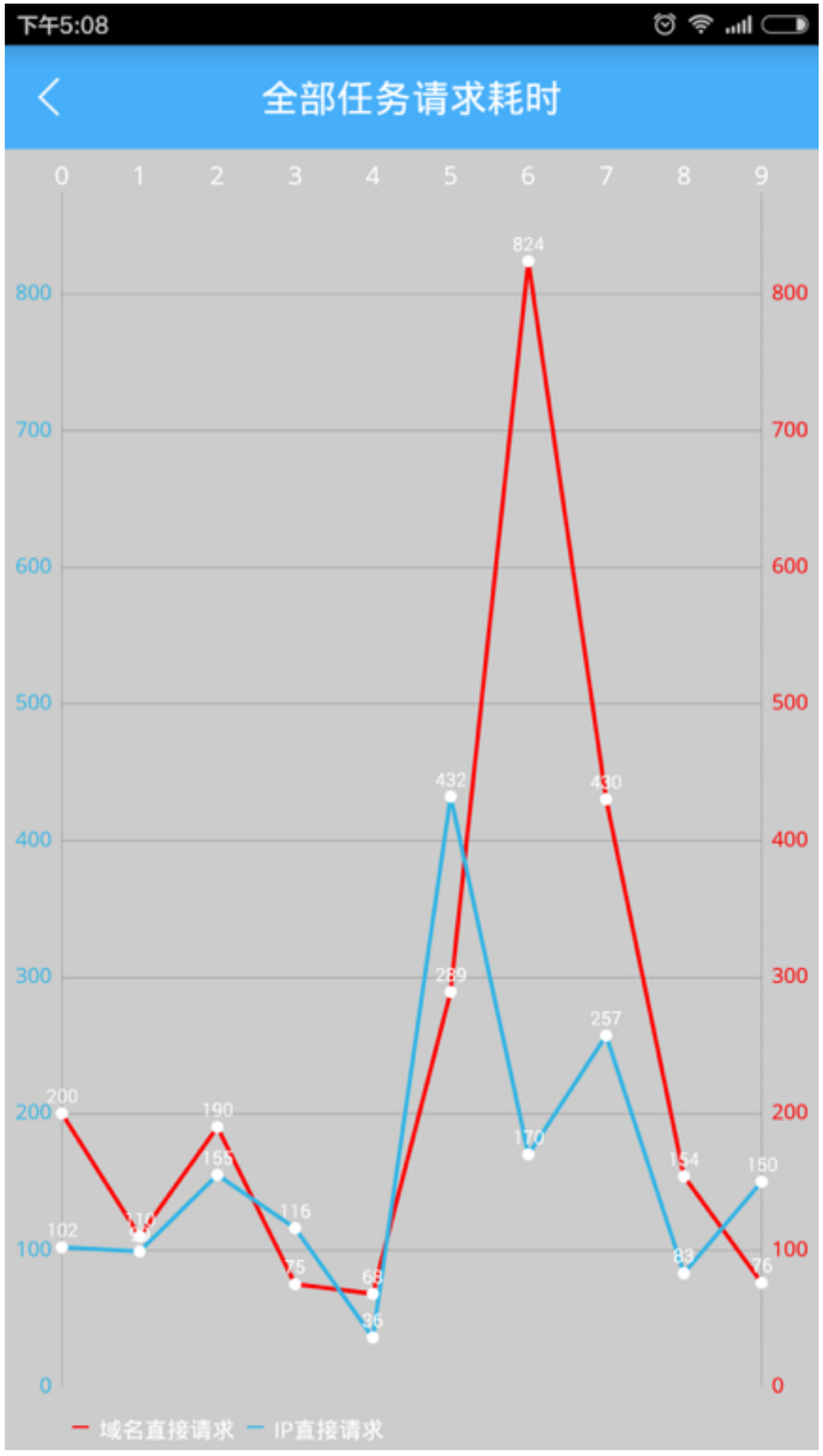


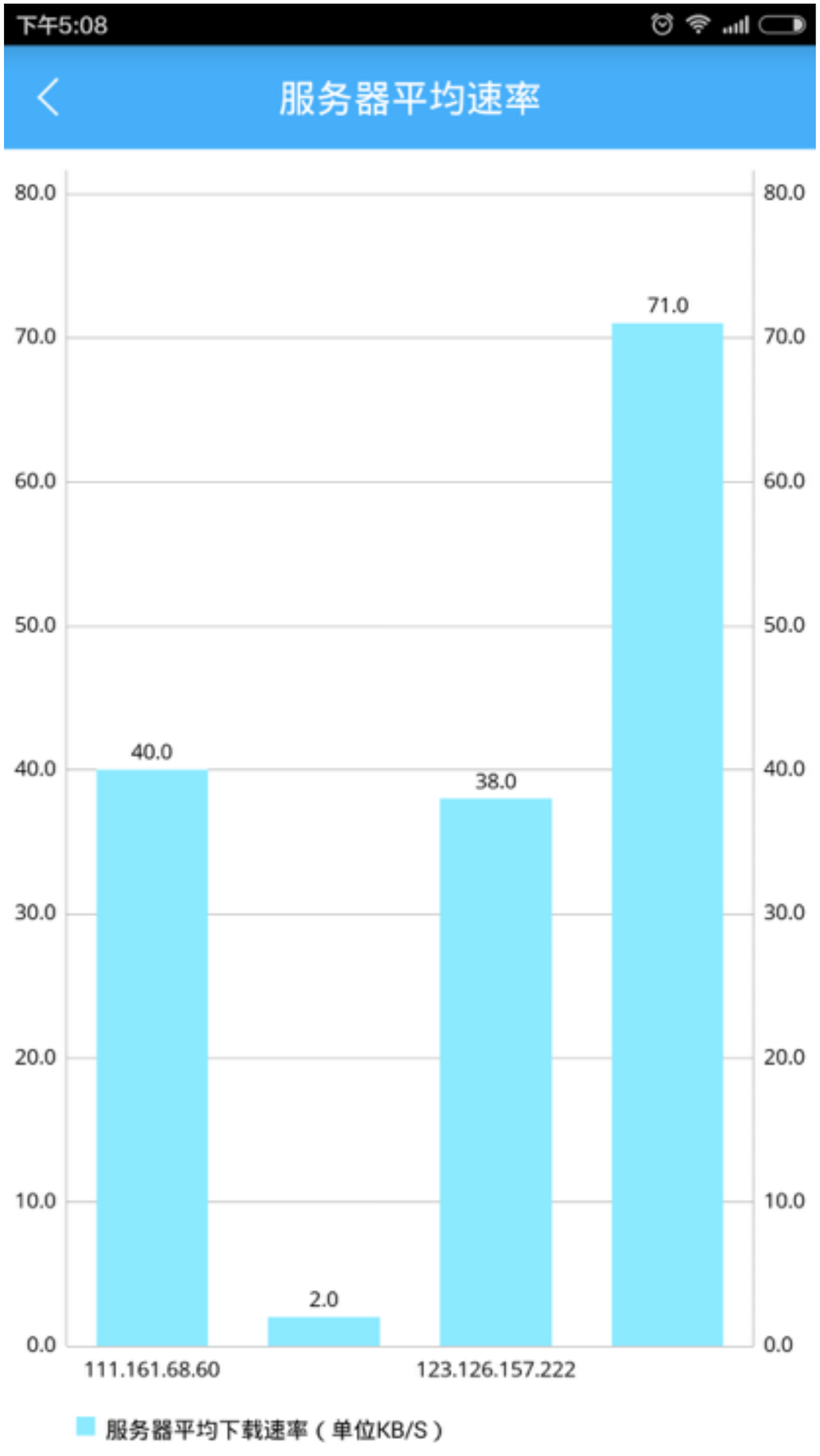
这个页面全都是数据库相关配置，
在代码中可以直接找到具体设置库文件的接口。





数据报表入口，包含全部任务加速效果延迟效果数据记录， lib库耗时走向，每个ip直接访问请求和domain访问请求速度对比， 统计了服务器平局速度。


下午5:08
全部任务请求耗时
0 1 2 3 4 5 6 7 8 9
800
700
600
500
400
300
200
100
0
824
432
430
289
257
200
190
155
116
102
75
68
36
170
154
150
83
76
域名直接请求
IP直接请求


下午5:08
服务器平均速率
80.0
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
71.0
40.0
38.0
2.0
111.161.68.60
123.126.157.222
服务器平均下载速率（单位KB/S）

缓存数据标签中包含了 当前库的所有状态， 能实时的看到内存缓存层的所有数据状态，包括数据库中得所有数据状态。 每秒钟刷新一次。 在这里可以清空缓存层数据、数据层数据、已经当前测试工程的数据。 在这里你可以清楚的看到 ip 和 domain的对应关系， 以及数据库表中 每项的关系。 和所有的domain 以及 ip 的状态。

Thanks

fenglei1@staff.sina.com.cn
xingyu10@staff.weibo.com

张杰 微博:@木泽水
王春生 微博:@平凡的香草
胡波 微博:@胡波_
韩超 微博:@超朝炒觥
赵星宇 微博:@淘淘不逃008
聂钰 微博:@古夜
黄振栋 微博:@BG2BKK
冯磊 微博:@冯磊424