OneStepOffer 系统设计第一讲
TinyURL

短网址服务

使用场景

微博和Twitter都有140字数的限制，如果分享一个长网址，很容易就超出限制，发布出去。短网址服务可以把一个长网址变成短网址，方便在社交网络上传播。

转化后的网址要尽可能短。那么多短才合适呢？

需求一:转化后的url 长度

当前互联网上的网页总数大概是45亿(参考 http://www.worldwidewebsize.com)，45亿超过了 2^32=42949672962

但远远小于64位整数的上限值，那么用一个64位整数足够了。

微博的短网址服务用的是长度为7的字符串，这个字符串可以看做是62进制的数，那么最大能表示62^7=352161460620862个网址，远远大于45亿。所以长度为7就足够了。

所以url长度需求为:长度不超过7的字符串，由大小写字母加数字共62个字母组成

需求二 - 每个输入长网址要去重吗？

一个长网址，对应一个短网址，还是可以对应多个短网址？这也是个重大选择问题

以这个7位长度的短网址作为唯一ID，这个ID下可以挂各种信息，比如生成该网址的用户名，所在网站，HTTP头部的 User Agent等信息，收集了这些信息，才有可能在后面做大数据分析，挖掘数据的价值。短网址服务商的一大盈利来源就是这些数据。

我们选择**允许一个长网址对应多个短网址**

如何计算对应的短网址呢

方法一: 计算输入网址的hash值

先hash字符串得到一个64位整数，将它转化为62进制整，截取低7位即可。

但是哈希算法会有冲突，如何处理冲突呢？

如果一个比较常见的网址: www.youtube.com，哈希算法会有很多次重复，对于计算效率非常不利

哈希算法行不通

如果计算对应的短网址呢

方法二: 全局增加ID

服务器统计当前的全局ID, 赋予新网址ID，并将全局ID+1

优点：连续值不会浪费数值空间，没有冲突，计算效率稳定

缺点：对于多机器较难保持一致性

常用工具：MySQL 内置自增ID

如果计算对应的短网址呢 - UUID

ObjectId占12个字节，由以下几个部分组成，

- 4个字节表示的Unix timestamp,
- 3个字节表示的机器的ID
- 2个字节表示的进程ID
- 3个字节表示的计数器

UUID的有点是每台机器可以独立产生ID，理论上保证不会重复，所以天然是适合分布式系统的

常用工具：**Snowflake**

如何存储

对于单台机器：

以短网址为 primary key, 长网址为value, 可以用传统的关系数据库存起来，例如MySQL, PostgreSQL

创建短网址索引，提高查询效率

新增需求：每天想要统计最受用户欢迎的长网址的域名

新增数据库column：url_domain 如 www.youtube.com

为 url_domain 创建索引，提高数据库运算效率

如何存储

多台机器:

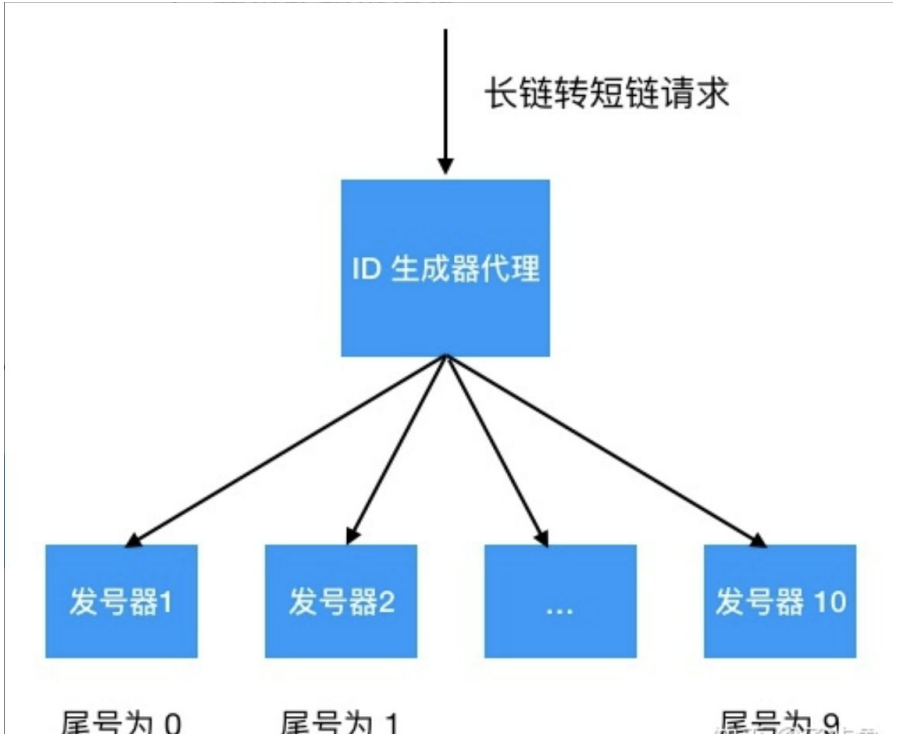
分布式Key-Value数据库，例如Redis

什么时候用分布式架构？

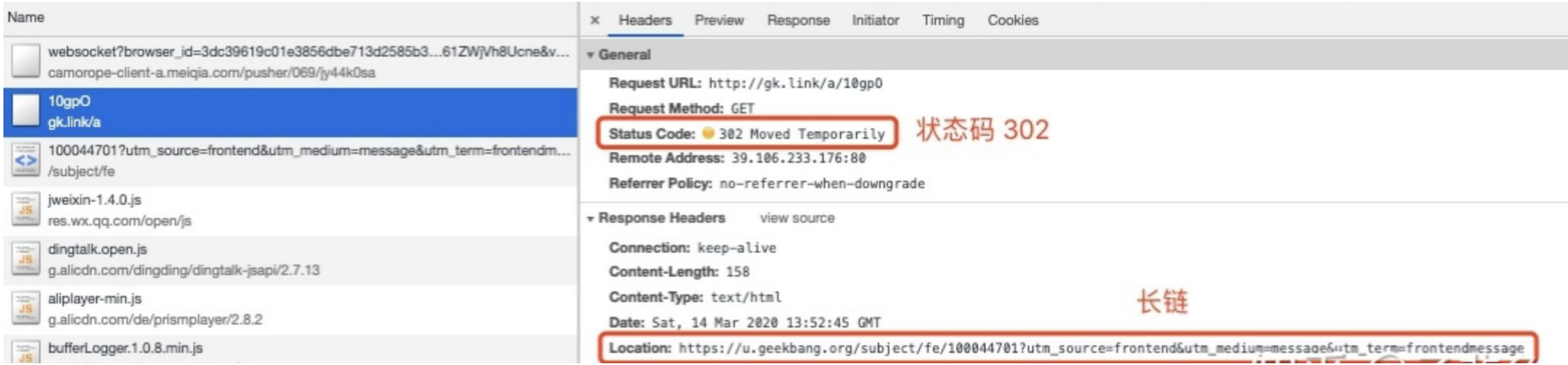
存储：1条 100bytes 则大概能估算出1T的硬盘能大概用1年。

达到微博的规模时需要分布式架构，但对于小规模系统是没有必要的

Redis 单机可支撑 10 w+ 请求，足以应付大部分的业务场景。布置 10 台机器，每台机器分 别只生成尾号 0，1，2，... 9 的 ID，每次加 10即可，只要设置一个 ID 生成器代理随机分配 给发号器生成 ID 就行了。



短链跳转的基本原理



可以看到请求后，返回了状态码 302（重定向）与 location 值为长链的响应，
然后浏览器会再请求这个长链以得到最终的响应（整个流程图 见下页）

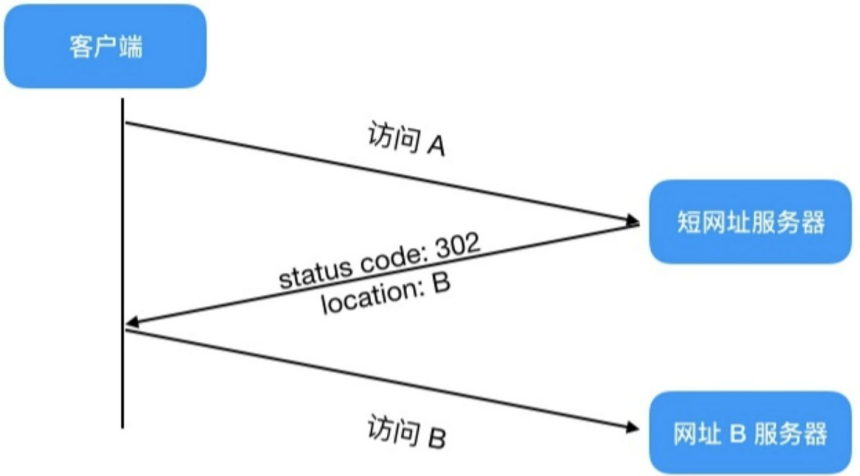
短链跳转的基本原理

整个交互流程图 如右图所示

客户通过访问短链接发请求到短网址服务器，返回stauts code = 302，再访问最终服务器B

A http://gk.link/a/10gpO

B https://u.geekbang.org/subject/fe/1000447O1?utm_source=frontend&utm_medium=messag



为什么返回302？

301 和 302 都是重定向，到底该用哪个？

301，代表 永久重定向，也就是说第一次请求拿到长链接后，下次浏览器再去请求短链的话，不会向短网址服务器请求了，而是直接从浏览器的缓存里拿，这样在我们的 server 层面就无法获取到短网址的点击数了，如果这个链接刚好是某个活动的链接，也就无法分析此活动的效果。所以我们一般不采用 301。

302，代表 临时重定向，也就是说每次去请求短链都会去请求短网址服务器（除非响应中用 Cache-Control 或 Expired 暗示浏览器缓存），这样就便于 server 统计点击数，所以虽然用 302 会给 server 增加一点压力，但在数据异常重要的今天，这点代码是值得的，所以推荐使用 302！

请求分发的系统设计

假设我们要面对10w+的QPS，比如春晚抢红包短链接的分享与访问

openResty：

基于 Nginx 与 Lua 的高性能 Web 平台，由于 Nginx 的非阻塞IO模型，使用 openResty 可以轻松支持 100 w + 的并发数，一般情况下你只要部署一台即可，同时 openResty 也自带了缓存机制，集成了 redis 这些缓存模块，也可以直接连 mysql

请求分发的系统设计

短链接请求直连Redis缓存层或者数据库存储层

绕过业务层中间件，可明显提高性能，又不增加业务层负担