**短URL系统设计方案**

作者：iammutex

链接：https://www.zhihu.com/question/29270034/answer/46446911

来源：知乎

著作权归作者所有，转载请联系作者获得授权。

看了一下 @Tang minyi 的回答，也非常不错，如果面试者这么回答，我也会很高兴的。

对于我用词比较激烈的问题，我面试时真实想法既是如此，也难得润色了。

--------------------------

这个问题看到就想答。

个人相关：三年前在公司做过一个短地址服务，目前在线上跑。

而这个问题，也是我现在招聘面试题里面必考的一道，这一道题里面有很多可考的地方，能够相对综合的考察候选人的功力。

最烂的回答

实现一个算法，将长地址转成短地址。实现长和短一一对应。然后再实现它的逆运算，将短地址还能换算回长地址。

这个回答看起来挺完美的，然后候选人也会说现在时间比较短，如果给我时间我去找这个算法就解决问题了。但是稍微有点计算机或者信息论常识的人就能发现，这个算法就跟永动机一样，是永远不可能找到的。即使我们定义短地址是100位。那么它的变化是62的100次方。62=10数字+26大写字母+26小写字母。无论这个数多么大，他也不可能大过世界上可能存在的长地址。所以实现一一对应，本身就是不可能的。

再换一个说法来反驳，如果真有这么一个算法和逆运算，那么基本上现在的压缩软件都可以歇菜了，而世界上所有的信息，都可以压缩到100个字符。这～可能吗。

另一个很烂的回答

和上面一样，也找一个算法，把长地址转成短地址，但是不存在逆运算。我们需要把短对长的关系存到DB中，在通过短查长时，需要查DB。

怎么说呢，没有改变本质，如果真有这么一个算法，那必然是会出现碰撞的，也就是多个长地址转成了同一个短地址。因为我们无法预知会输入什么样的长地址到这个系统中，所以不可能实现这样一个绝对不碰撞的hash函数。

比较烂的回答

那我们用一个hash算法，我承认它会碰撞，碰撞后我再在后面加1，2，3不就行了。

ok，这样的话，当通过这个hash算法算出来之后，可能我们会需要做btree式的大于小于或者like查找到能知道现在应该在后面加1，2，或3，这个也可能由于输入的长地址集的不确定性。导致生成短地址时间的不确定性。同样烂的回答还有随机生成一个短地址，去查找是否用过，用过就再随机，如此往复，直到随机到一个没用过的短地址。

正确的原理

上面是几种典型的错误回答，下面咱们直接说正确的原理。

正确的原理就是通过发号策略，给每一个过来的长地址，发一个号即可，小型系统直接用mysql的自增索引就搞定了。如果是大型应用，可以考虑各种分布式key-value系统做发号器。不停的自增就行了。第一个使用这个服务的人得到的短地址是http://xx.xx/0 第二个是 http://xx.xx/1 第11个是 http://xx.xx/a 第依次往后，相当于实现了一个62进制的自增字段即可。

几个子问题

1. 62进制如何用数据库或者KV存储来做？

其实我们并不需要在存储中用62进制，用10进制就好了。比如第10000个长地址，我们给它的短地址对应的编号是9999，我们通过存储自增拿到9999后，再做一个10进制到62进制的转换，转成62进制数即可。这个10～62进制转换，你完全都可以自己实现。

2. 如何保证同一个长地址，每次转出来都是一样的短地址

上面的发号原理中，是不判断长地址是否已经转过的。也就是说用拿着百度首页地址来转，我给一个http://xx.xx/abc 过一段时间你再来转，我还会给你一个 http://xx.xx/xyz。这看起来挺不好的，但是不好在哪里呢？不好在不是一一对应，而一长对多短。这与我们完美主义的基因不符合，那么除此以外还有什么不对的地方？

有人说它浪费空间，这是对的。同一个长地址，产生多条短地址记录，这明显是浪费空间的。那么我们如何避免空间浪费，有人非常迅速的回答我，建立一个长对短的KV存储即可。嗯，听起来有理，但是。。。这个KV存储本身就是浪费大量空间。所以我们是在用空间换空间，而且貌似是在用大空间换小空间。真的划算吗？这个问题要考虑一下。当然，也不是没有办法解决，我们做不到真正的一一对应，那么打个折扣是不是可以搞定？这个问题的答案太多种，各有各招，我这就不说了。（由于实在太多人纠结这个问题，请见我最下方的更新）

3. 如何保证发号器的大并发高可用

上面设计看起来有一个单点，那就是发号器。如果做成分布式的，那么多节点要保持同步加1，多点同时写入，这个嘛，以CAP理论看，是不可能真正做到的。其实这个问题的解决非常简单，我们可以退一步考虑，我们是否可以实现两个发号器，一个发单号，一个发双号，这样就变单点为多点了？依次类推，我们可以实现1000个逻辑发号器，分别发尾号为0到999的号。每发一个号，每个发号器加1000，而不是加1。这些发号器独立工作，互不干扰即可。而且在实现上，也可以先是逻辑的，真的压力变大了，再拆分成独立的物理机器单元。1000个节点，估计对人类来说应该够用了。如果你真的还想更多，理论上也是可以的。

4. 具体存储如何选择

这个问题就不展开说了，各有各道，主要考察一下对存储的理解。对缓存原理的理解，和对市面上DB、Cache系统可用性，并发能力，一致性等方面的理解。

5. 跳转用301还是302

这也是一个有意思的话题。首先当然考察一个候选人对301和302的理解。浏览器缓存机制的理解。然后是考察他的业务经验。301是永久重定向，302是临时重定向。短地址一经生成就不会变化，所以用301是符合http语义的。同时对服务器压力也会有一定减少。

但是如果使用了301，我们就无法统计到短地址被点击的次数了。而这个点击次数是一个非常有意思的大数据分析数据源。能够分析出的东西非常非常多。所以选择302虽然会增加服务器压力，但是我想是一个更好的选择。

大概就是这样。

------五一假期后更新-------

就回答一点大家最纠结的问题吧，就是如何实现同一个长地址多次转换，出来还是同一个短地址。

我上面其实讲到了，这个方案最简单的是建立一个长对短的hashtable，这样相当于用空间来换空间，同时换取一个设计上的优雅（真正的一对一）。

实际情况是有很多性价比高的打折方案可以用，这个方案设计因人而异了。那我就说一下我的方案吧。

我的方案是：用key-value存储，保存“最近”生成的长对短的一个对应关系。注意是“最近”，也就是说，我并不保存全量的长对短的关系，而只保存最近的。比如采用一小时过期的机制来实现LRU淘汰。

这样的话，长转短的流程变成这样：

1 在这个“最近”表中查看一下，看长地址有没有对应的短地址

1.1 有就直接返回，并且将这个key-value对的过期时间再延长成一小时

1.2 如果没有，就通过发号器生成一个短地址，并且将这个“最近”表中，过期时间为1小时

所以当一个地址被频繁使用，那么它会一直在这个key-value表中，总能返回当初生成那个短地址，不会出现重复的问题。如果它使用并不频繁，那么长对短的key会过期，LRU机制自动就会淘汰掉它。

当然，这不能保证100%的同一个长地址一定能转出同一个短地址，比如你拿一个生僻的url，每间隔1小时来转一次，你会得到不同的短地址。但是这真的有关系吗？