0天来算，短 URL 总条数为 21 亿+。

```
计算公式：300W*30天*12月*2年=21亿+
```

数据库存储预估

按照我的模型设计，单条数据 0.6k 左右，总容量约为 1201 GB。ps：存储是没有算触达记录的（访问记录）

```
计算公式：21亿*0.6k=1201GB
```

短 URL CODE 预估

短 URL 的 CODE 用的 base62（其实也可以 base64，后面我会讲），长度选择 6 个字符（也可以是7个），我希望整个长度越短越好，6 个字符生成的量要少一些。

CODE 长度是 6 个字符：每一个字符都可以是 62 个字符中的任何一个的组合方式，所以 CODE 数量是 62 的 6 次方大约是 568 亿。

CODE 长度是 7 个字符：62 的 7 次方大约是 3.52 万亿。

按照数据行数的预估，2 年的数据是 21 亿，所以，采用 6 个字符是完全够用的。

QPS 预估

全平台有 21 亿短 URL，2 年内，每个 URL 平均被访问 300 次，每秒平均

QPS 为 9988 qps/s，高峰期 qps 会更高一些，预估翻 1.5 倍，qps 为 14982 qps/s。

```
计算公式：
1、总请求总量：21 亿 * 300 约等于 6300 亿
2、年转换为妙：730 天 × 24 小时/天 × 3600 秒/小时 = 63072000 秒
3、平均QPS：总请求量/时间=6300亿/63072000妙=9988 qps/s
4、业务高峰期QPS：平均QPS*1.5=9988*1.5=14982 qps/s
```

带宽预估

一次 http 请求占用带宽有下面几种因素决定。

1. 请求头部大小：包括请求行、各种头部字段（如User-Agent、Accept、Cookie等）、空行等。
2. 请求体大小：如果是POST请求，并且包含了请求体数据，那么请求体的大小也需要考虑在内。
3. 响应头部大小：与请求头部类似，响应头部也包括状态行、各种头部字段等。
4. 响应体大小：实际的响应数据大小，例如HTML页面、图片、JSON数据等。

长 URL 大小预估 200 字节 (B) 左右；加上请求头、请求体，响应头等，预估 800 字节 (B)左右，单次请求小于 1 kb。QPS 每秒 9988 ，平均每秒带宽为 9.5 Mb，业务高峰期每秒带宽为 14.2 MB。

```
计算公式：
平均带宽：1000 字节 (B) * 9988=9.5 兆字节 (MB)
业务高峰期带宽：平均带宽*1.5=9.5*1.5=14.25兆字节 (MB)
```

Redis 存储预估

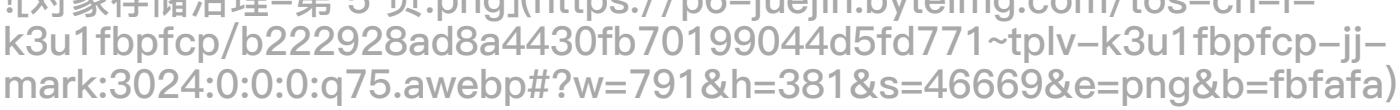
平均 qps 预估是 9988 qps/s，业务高峰期翻1.5倍 14982 qps/s ，不可能把所有流量都打到数据库，redis 至少抗 90% 流量，数据库才比较安全。但是我们不可能把所有的短链都缓存起来（1000gb+，redis 也扛不住啊）。

如果有线上/业界数据，根据这些数据来定。根据我的经验，刚生成的短 URL 才是热点链接，所以，决定保留 7 天数据。另外，缓存 miss 从数据库加载也会被保存在 redis 中。

7 天短 URL 生成量为 2100W，再加历史短 URL 缓存预热 500 W，所以，redis 总缓存数量 2600W，redis 只保留 CODE 和长 URL，预估 800 字节 (B)，总容量为 19.3 千兆字节 (GB)

```
计算公式
1、7 天短 URL 生成量 ： 300W*7=2100W
2、缓存预热：缓存 miss，从数据库加载预估=500W
3、redis容量消耗：2600W*800 字节 (B)=19.3 千兆字节 (GB)
```

架构图

短 URL 架构图比较简单


关键路径概述

1. 前端输入长 URL，短链服务生成 CODE，mapping 长 URL，数据落库，同时返回完整短 URL；
2. 前端访问短 URL，短链服务通过CODE 查询长 URL，302/301 重定向长 URL 访问目标服务即可。

架构拆解

CODE 算法

我只讲 base62 算法，其它算法网上一大堆。标准的 base64 应该有64 个字符。但，网上算法都是 base62，但是没有给具体原因，随手百度下 base64 标准表编码。

索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1

>
> a. tidb 提供了定期删除数据的能力，我看腾讯云的数据库也支持了。[使用 TTL (Time to Live) 定期删除过期数据 | PingCAP 文档中心](http://cxyroad.com/ "https://docs.pingcap.com/zh/tidb/stable/time-to-live")
> ![image.png](https://p1-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/e94539f393e442ec89c07ad1f23997dc~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1090&h=390&s=105998&e=png&b=ffffff)

OK，删除数据完美解决。如果有备份数据需求提前想办法备份咯。或者 binglog 事先同步好。

核心代码

```
import (
"crypto/md5"
"encoding/base64"
"github.com/bwmarrin/snowflake"
"io"
"strings"
)

type Shortener interface {
Generate(url string) (string, error)
}

var base64Factory = []string{
"0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k",
"l", "m", "n", "o", "p", "q", "r", "s", "t", "u",
"v", "w", "x", "y", "z", "A",
"B", "C", "E", "F", "D", "G", "H", "I", "J", "K", "L", "M",
```

```
”N”, ”O”, ”P”, ”Q”, ”R”, ”S”, ”T”, ”U”, ”V”, ”W”, ”X”, ”Y”, ”Z”, ”-”, ”_”,
}

const (
size = 6
)

type snowflakeGenerate struct {
}

func (s *snowflakeGenerate) Generate(url string) (string, error) {
node, err := snowflake.NewNode(1)
if err != nil {
return ””, err
}

id := node.Generate().Int64() // todo 高并发情况下可能会有冲突
l := int64(len(base64Factory))
sb := strings.Builder{}
for ; id/l > 0; id /= l {
index := id % l
sb.WriteString(base64Factory[index])
}

return sb.String()[:size], nil // 截断字符串
}

type md5Generate struct {
}

func (s *md5Generate) Generate(url string) (string, error) {
hash := md5.New()
_, err := io.WriteString(hash, url)
if err != nil {
return ””, err
}

b := hash.Sum(nil)
es := base64.RawURLEncoding.EncodeToString(b) // url base64编码，已经过滤转义字符

return es[:size], nil // 截断字符串
}
```

测试代码

```
func TestSnowflakeGenerate(t *testing.T) {
s := &snowflakeGenerate{}
code, err := s.Generate("")
if err != nil {
panic(err)
}
fmt.Println(code)
}

func TestMD5Generate(t *testing.T) {
s := &md5Generate{}
code, err := s.Generate("https://docs.pingcap.com/zh/tidb/stable/time-to-live")
if err != nil {
panic(err)
}
fmt.Println(code)
}
```

结尾

短 URL 系统，存储 21 亿+数据、高 qps 但是技术方案看起来并不算复杂。只要保证 CODE 生成和长 URL 的映射正确；

数据量其实都不算大，现在分布式数据库存储这点量是完全没问题的；

提升 QPS 可以考虑 Redis 存储，极端情况还可以上内存缓存，能有效提升系统的 QPS；

另外，短 URL 服务都是多 POD，高可用这块完全没问题。

写作不易各位看官动动发财小手点点赞。

祝：大家周末愉快。

原文链接: https://juejin.cn/post/7367635686936559667