短网址/短链 项目

> 七米课堂 微服务实战课程 课件！ 更多Go学习内容👉 liwenzhou.com

说在前面的话

短网址项目是一个非常不错的实战项目。

首先短网址的应用非常广泛，每个人的手机里总会有几条使用了短网址的短信；

其次它的业务逻辑不是很复杂，很容易理解，因此非常适合作为练手项目。

同时，这个项目也可以成为“**能写到简历上的项目**”和“**能在公司内部落地的项目**”。

课程中我们将通过这个项目，详细地介绍从需求评审到项目开发的完整过程。

手把手带你学会如何分析需求、拆解需求以及实现需求。

什么是短网址/短链接?

短链接，通俗来说就是将比较长的一个URL网址，通过程序计算等方式，转换为简短的网址字符串。

https://www.liwenzhou.com/posts/Go/golang-menu 这是一个很长的URL网址。

类似 `q1mi.cn/1ly7vk` 这种形式的就属于短链接。

很多公司有短链接服务：

- 百度：dwz.cn
- 微博：t.cn
- ...

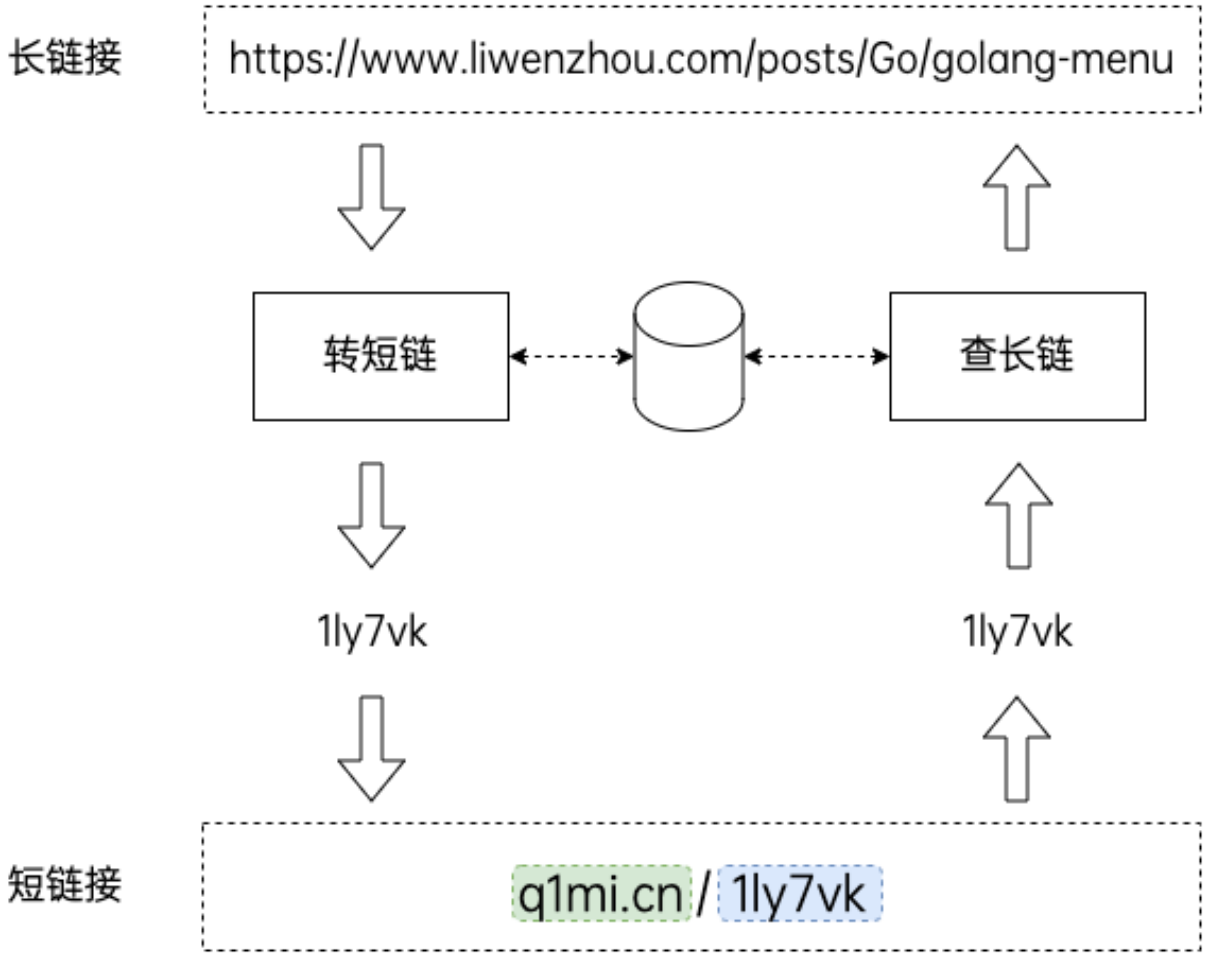
为什么要用短网址/短链接?

公司内部有很多需要发送链接的场景，业务侧的链接通常会比较长，在发送短信、IM工具发送消息、push等场景下长链接有以下劣势：

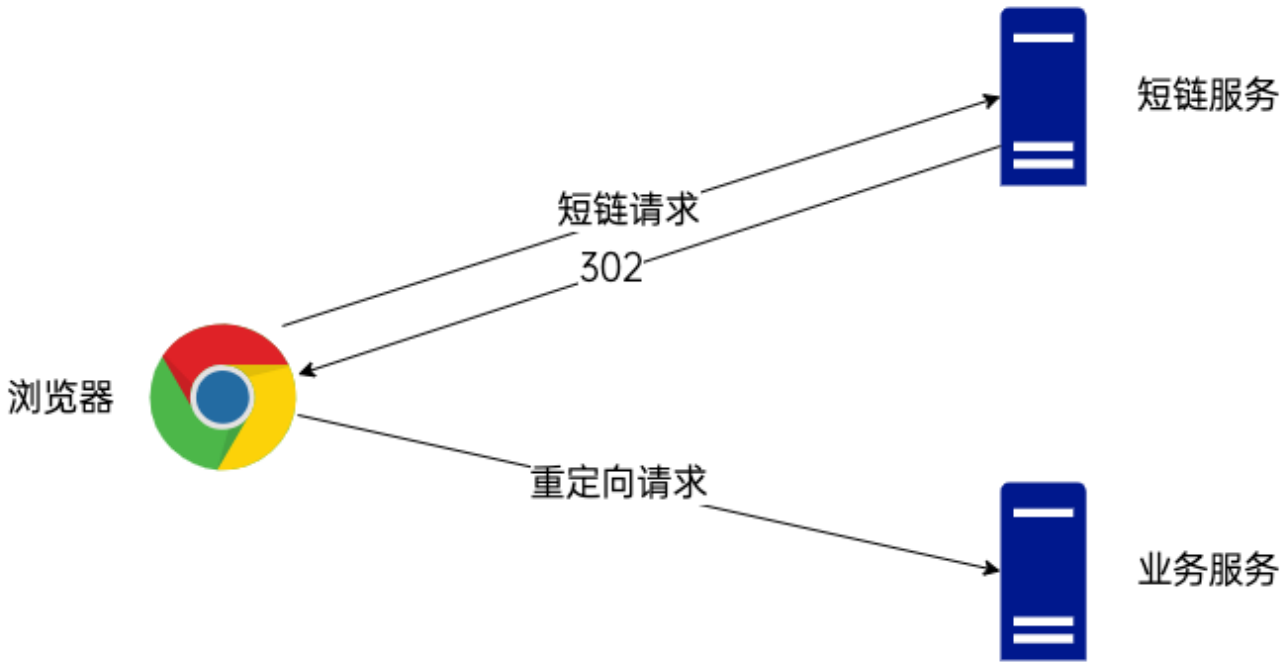
1. 短信内容超长，1条消息被拆分成多条短信发送，浪费钱。
2. 微博等平台有字数限制。
3. 飞书、钉钉等IM工具对长链接（带特殊服务号的）识别有问题。
4. 短链接转成二维码更清晰。

短网址/短链接原理

长链接 → 短链接


长链接
https://www.liwenzhou.com/posts/Go/golang-menu
转短链
查长链
1ly7vk
1ly7vk
短链接
q1mi.cn / 1ly7vk

查看短链接


短链服务
短链请求
302
浏览器
重定向请求
业务服务

代码示例

```go
// shorturllogic.go

func (l *ShorturlLogic) Shorturl(req *types.Request) (resp *types.Response, err error)
{
  // 根据短链接请求的标识符，查找到对应的原始长链接
  // req.ShortURL // 标识符 1ly7vk
  if req.ShortURL == "1ly7vk" {
    return &types.Response{LongURL: "https://www.liwenzhou.com/posts/Go/golang-menu"},
nil
  }
  // 其他查询不到长链接的请求都跳转到 baidu.com
  return &types.Response{LongURL: "https://www.baidu.com"}, nil
}
```

```go
// shorturlhandler.go

func ShorturlHandler(svcCtx *svc.ServiceContext) http.HandlerFunc {
  return func(w http.ResponseWriter, r *http.Request) {
    var req types.Request
    if err := httpx.Parse(r, &req); err != nil {
      httpx.ErrorCtx(r.Context(), w, err)
      return
    }

    l := logic.NewShorturlLogic(r.Context(), svcCtx)
    resp, err := l.Shorturl(&req)
    if err != nil {
      httpx.ErrorCtx(r.Context(), w, err)
    } else {
      // 之前是返回响应数据
      // httpx.OkJsonCtx(r.Context(), w, resp)

      // 返回重定向的HTTP响应
      // w.Header().Set("location", resp.LongURL) // location
      // w.WriteHeader(http.StatusFound)          // status code

      // 另外的一种方式，使用http包里的重定向方法
      http.Redirect(w, r, resp.LongURL, http.StatusFound)
    }
  }
}
```

> 七米课堂 微服务实战课程 课件！ 更多Go学习内容👉 liwenzhou.com

需求介绍

需求背景

为什么要设计短链系统?

公司内部业务需要发送大量的营销短信、通知类短信。

需要一个短链接服务满足各业务线的使用。

- 提供转链接口
- 后续支持提供点击的统计数据报表

需求描述

1. 输入一个长网址得到一个唯一的短网址。
2. 用户点击短网址能够正常跳转到对应的网址。
3. 为了保证业务的延续性，短网址长期有效。

需求分析

产品定位

1. 公司内部业务使用的短网址服务，只接收公司内部的长链转短链需求。（不对外提供短链功能。）
2. 基本在国内使用（点击链接的用户绝大多数为国内用户）
3. 后续可能会要求提供短链的访问数据报表

规模

1. 大致服务于公司内部x条业务线。
2. 大致服务的用户规模有x亿。
3. xx QPS

技术指标

1. 延时x ms内
2. 可靠性99.99%
3. 安全性

> 在公司如果遇到比较复杂的需求，研发给PM反讲一下需求

需求从提出到实现的步骤

需求预沟通 -> 需求评审 -> 技术评审 ->排期开发 -> 联调 -> 测试 -> 上线（小流量->全流量）

需求拆解

根据需求分析，可以将需求拆分为**转链模块**、**存储**和**访问链接模块**。

转链模块

1. 相同的长链要转为同一个短链
2. 生成的短链为尽量短的字符。`q1mi.cn/p6Yo7Z`

作为一个开发想得再多一点，引申出来的需求点或注意事项。

1. 需要避免某些不合适的词（例如`f**k`、`stupid`）
2. 避免生成的短链出现某些特殊含义的词`version`、`health`等
3. 避免循环转链（把已经是短链的再拿来转短链）

存储

1. 保存原始长链接与短链接的对应关系
2. 能够根据短链接查找到原始的长链接

查看链接模块

1. 根据短链查询到长链后返回重定向响应。
2. 后续数据报表需求可能需要采集并统计请求头数据。

系统设计

总体设计方案

通过分析可以得知，这是一个典型的**读多写少**的系统。

并且我们进一步分析这个短链系统区别于其他读多写少的业务场景，它的特点是数据写入后基本不会改变。（好处是不需要考虑数据一致性的问题，可以放心大胆的使用缓存系统来提高读的效率）

短链生成方式

关于生成短链有以下几种方案，

hash

使用hash函数对长链接进行hash，得到hash值作为短链标识符。

优势：简单

缺点：数据量大之后，会出现哈希冲突

扩展：

`MurmurHash`是一种非加密型哈希函数，和其它流行哈希函数相比，对于规律性较强的key随机分布特性表现更良好，在很多开源的软件项目（Redis，Memcached，Cassandra，HBase，Lucene都用它）都有使用。有以下几个特性：

- 随机分布特性表现好
- 算法速度快

golang版本murmurhash实现

发号器/自增序列

每收到一个转链请求，就使用发号器生成递增（1、2、3、4...以此递增）的序号，然后将该序号转为**62进制**，最后拼接到短域名后即得到最终的短链。

例如：

序号`1234567890`转为62进制为`1ly7vk`，再拼接到短域名后`q1mi.cn/1ly7vk`。

> 什么是62进制？👉 使用数字（0-9）和大小写英文字母（a-zA-Z） $$10+26*2=62$$
>
> 为什么要使用62进制？👉 因为字母、数字能组成合法的URL，浏览器能认识

发号器方案的优劣如下

优势

- 生成的id递增
- 理论上容量足够满足现实需求

缺点：

- 高并发下的发号器设计是难点。

发号器实现方式

常见的发号器实现方式有以下几种：

1. 基于uuid实现
    - 优势：不会重复、性能好
    - 劣势：数字太大了，32位16进制数
2. 基于redis实现发号器
    - 优势：高性能
    - 劣势：需搭建高可用架构并考虑持久化
3. 基于雪花算法的分布式ID生成器
    - 优势：高性能、高可用
    - 劣势：实现复杂，依赖时钟
4. 基于MySQL自增主键的发号器
    - 优势：简单、可靠

- 劣势：依赖MySQL，性能会成为瓶颈，但可通过分片扩展可用性

基于MySQL主键实现发号器

这里采用的是基于MySQL数据库主键做发号器的方案。

我们新建一个数据表，这个表结构简单，没有很多其他数据字段。

每有一次转链请求，我们都在这个数据表中插入一个新记录，那么我们可以使用该表的自增 ID 作为生成的号码。

MySQL REPLACE

REPLACE 的工作方式与 INSERT 完全相同，只是如果表中的旧行与新行在PRIMARYKEY 或 UNIQUE 索引具有相同的值，则在插入新行之前删除旧行。

这就让我们能够在数据库中的单行位置进行自动更新，并获得一个新的自动递增的主 ID。

数据表：

```
CREATE TABLE `sequence` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `stub` varchar(1) NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_uniq_stub` (`stub`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COMMENT = '序号表';
```

SQL语句：

```
REPLACE INTO sequence (stub) VALUES ('a');
SELECT LAST_INSERT_ID();
```

分片部署

为了避免单点故障，我们将我们的ID生成器分成奇数和偶数两部分，分别部署在两个MySQL服务器。

两个数据表配置不同的`auto-increment-offset`，`server1`生成1、3、5、7、9...，`server2`生成2、4、6、8...。

```
server1:
auto-increment-increment = 2
auto-increment-offset = 1

server2:
auto-increment-increment = 2
auto-increment-offset = 2
```

数据表

存储长-短链接映射的数据表：

```
CREATE TABLE `short_url_map` (
    `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主键',
    `create_at` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
    `create_by` VARCHAR(64) NOT NULL DEFAULT '' COMMENT '创建者',
    `is_del` tinyint UNSIGNED NOT NULL DEFAULT '0' COMMENT '是否删除：0正常1删除',

    `lurl` varchar(2048) DEFAULT NULL COMMENT '长链接',
    `md5` char(32) DEFAULT NULL COMMENT '长链接MD5',
    `surl` varchar(11) DEFAULT NULL COMMENT '短链接',
    PRIMARY KEY (`id`),
    INDEX(`is_del`),
    UNIQUE(`md5`),
    UNIQUE(`surl`)
)ENGINE=INNODB DEFAULT CHARSET=utf8mb4 COMMENT = '长短链映射表';
```

- `lurl`和`surl`都唯一，长链和短链都不允许重复
- 长链接数据量大不适合建索引，所以这里使用其md5值来做索引。
- `is_del` 软删除标识

> 可以采用读写分离的模式，写主库、读从库。

数据量

> URL网址的最大长度不同浏览器下不同。

1. 1条长链+短链的需要的存储空间假设为`200Bytes/条`
2. 假设每个业务线每秒会写入100条，10条业务线是`1000条/秒`
3. 一主两备共三份数据，索引等冗余系统1.5

$200*1000*3600*24*365*3*1.5 \approx 20T$

数据如何删除

因为本项目不涉及链接配置过期时间，所有的删除均为手动标记删除。

预防攻击的方式

如果是对外服务

- IP限制请求数
- 用户限制转链额度
- 记录已转链的URL缓存防止刷光ID （适用于相同的URL可转为不同的短链的场景）
    - LRU缓存URL
    - 布隆过滤器
- 校验链接是否有效

```
timeout := time.Duration(1 * time.Second)
client := http.Client{
    Timeout: timeout,
}
resp, err := client.Get("http://google.com")
```

如果是内部服务则需要考虑

- 权限认证
- 账户每日限额
- 校验链接是否有效

访问短链方式

基本方案

短链接请求进来后，根据标识符查询MySQL数据库（根据短链查询长链），然后返回重定向响应。

增加缓存

为了提高性能，可以增加Redis缓存

> 甚至可以加本地缓存。程序启动时加载到内存。

缓存相关问题

使用Redis作为缓存，那么就需要考虑几个核心问题。

1. 缓存怎么设置，LRU
    1. Redis集群部署
    2. 根据数据量设置内存大小，内存淘汰策略LRU，移除最近最少使用的key。
2. 如果解决缓存击穿问题？ 👉 引申：什么是缓存雪崩、缓存击穿、缓存穿透
    1. 过期时间设大
    2. 加锁

3. 使用singleflight 合并请求 👉 https://www.liwenzhou.com/posts/Go/singleflight/

3. 如何解决缓存穿透问题?

> 什么是缓存穿透?
>
> 攻击者恶意请求短链服务，短时间大量请求并不存在的短链

1. 布隆过滤器（简单，如果不在那一定不在）
    1. 为什么需要使用布隆过滤器?
        1. 节省空间。并不存储原始数据，只用来判断某个元素是否存在。
    2. 原理
        - 介绍：https://www.cnblogs.com/cpselvis/p/6265825.html
        - 在线可视化：https://www.jasondavies.com/bloomfilter/
    3. 实现
        - Go库：https://github.com/bits-and-blooms/bloom
        - go-zero bloom:https://go-zero.dev/cn/docs/blog/governance/bloom/
    4. 布隆过滤器变种（自己回去查）
        1. 计数布隆过滤器
        2. 。。。
    5. 应用
        1. 防止缓存穿透
        2. 推荐系统去重（文章、视频等推荐去重）
        3. 黑白名单
        4. 垃圾邮件过滤
2. 布谷鸟过滤器（支持删除）
    1. 原理
        - 介绍：https://www.cnblogs.com/zhaodongge/p/15067657.html
        - paper：http://www.linvon.cn/posts/cuckoo/
        - 可视化：http://www.lkozma.net/cuckoo_hashing_visualization/
    2. 实现
        - Go库：https://github.com/seiflotfy/cuckoofilter

> 本项目适合使用布谷鸟过滤器来过滤短链，因为布谷鸟过滤器的缺点是对重复添加相同数据有限制，但是短链接项目不会出现重复添加相同短链。
>
> 但是本项目中查看链接这里没有必要使用布谷鸟过滤器来支持删除短链，因为相对于短链总数来说已删除的短链（is_del=1）很少，即使过滤器放行了被删除的短链影响也不大。

部署

单独部署、与业务隔离

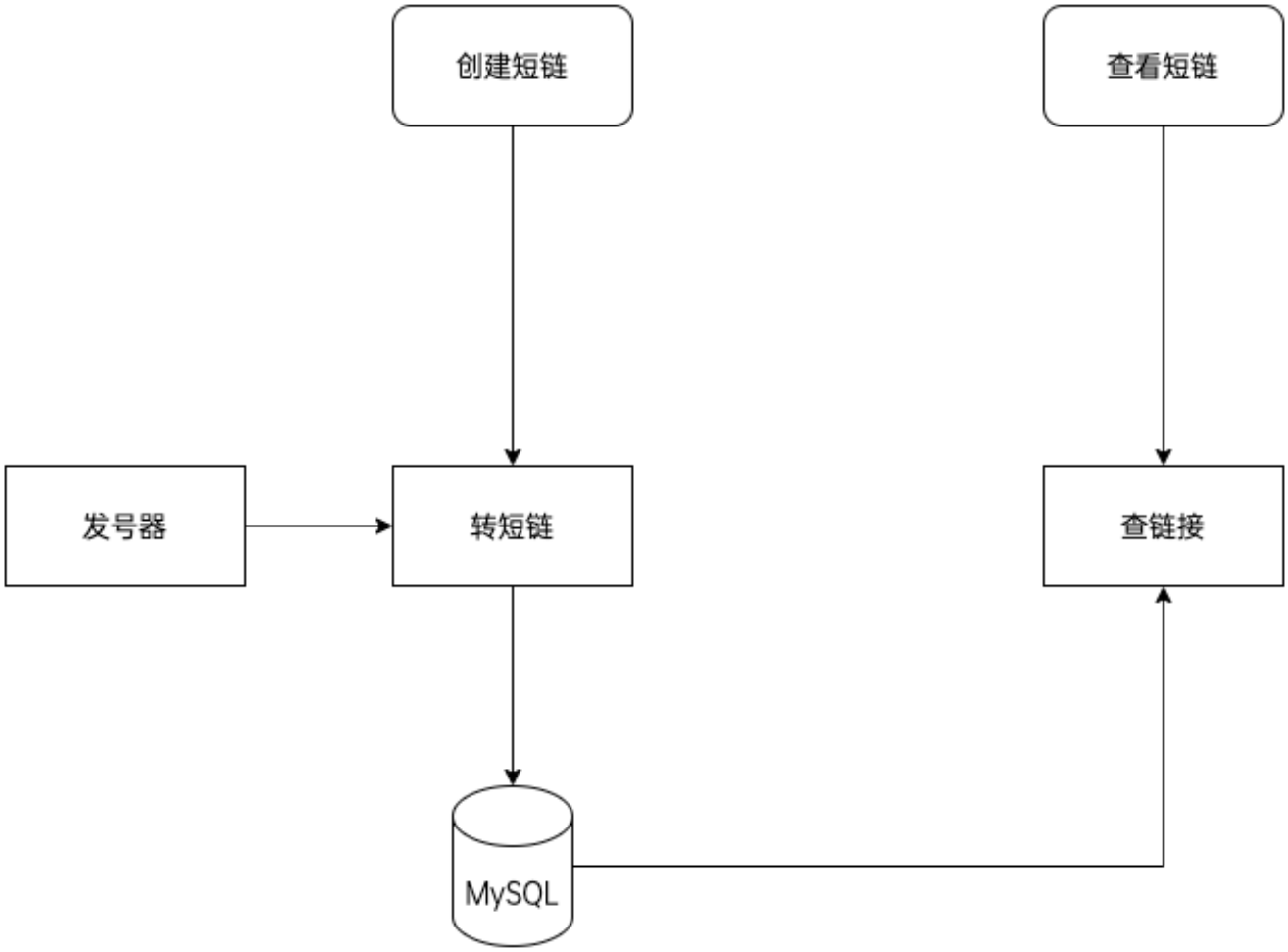
部署该项目的一种推荐方法是在通过`Nginx`代理，即将我们的短链服务部署在`Nginx`后。

通过这种方式，可以通过 `Nginx` 的访问日志（access.log）来统计访问数据。（例如通过EFK采集日志，统计报表）

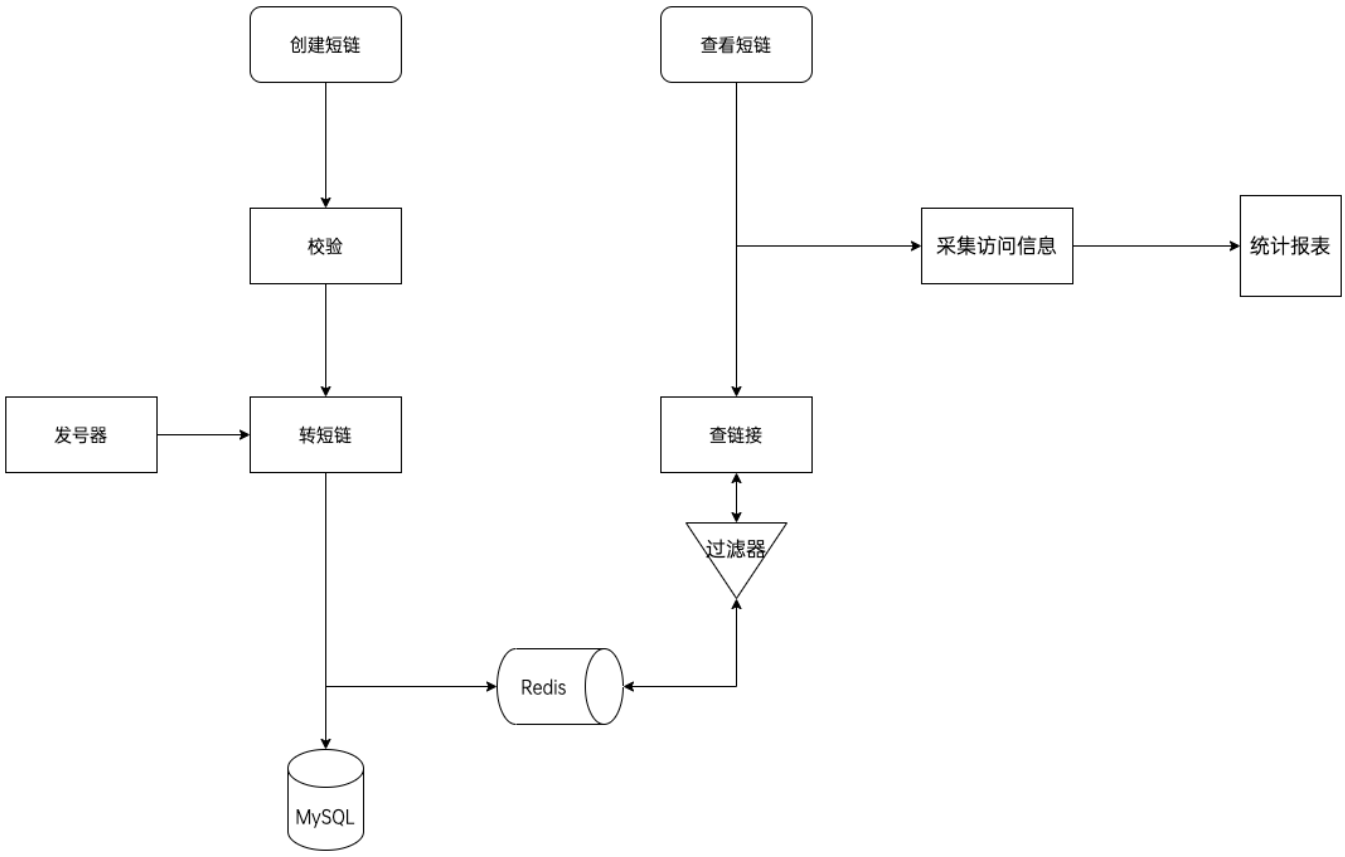
> 七米课堂 微服务实战课程 课件！ 更多Go学习内容👉 liwenzhou.com

项目架构图

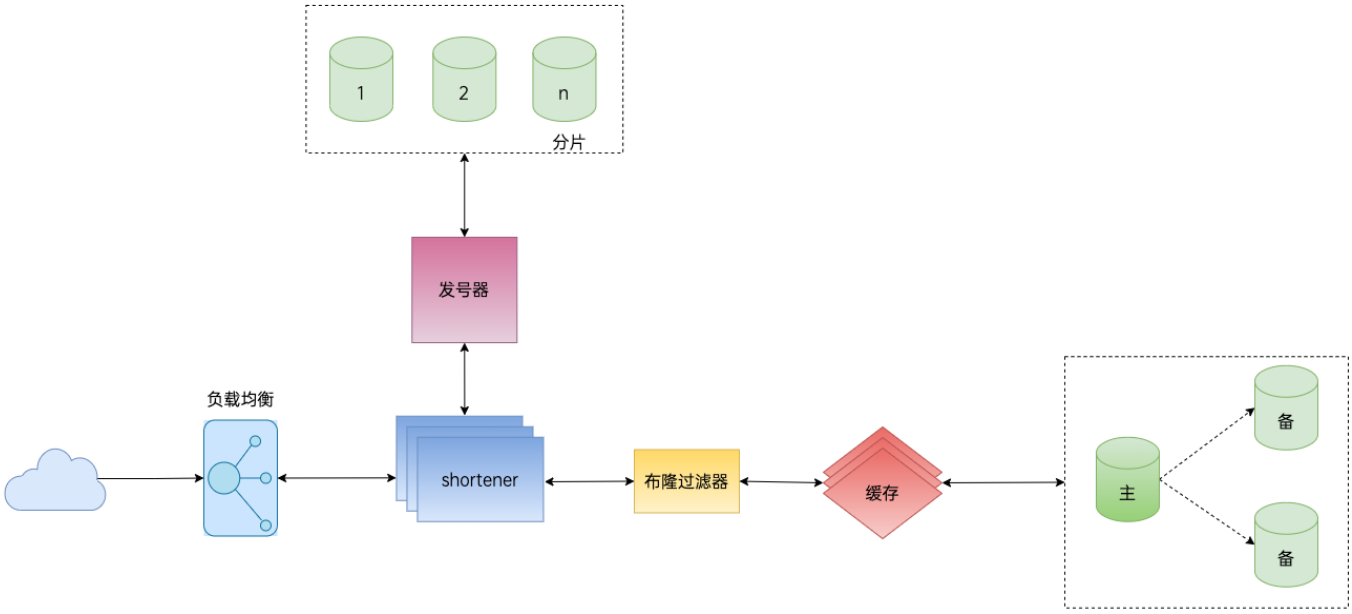
v1版本



v2版本-增加布隆过滤器



项目架构图



- 长链转短链：
  - 单独部署为一个微服务（转链服务）

    - 对其他服务提供转链服务，需要鉴权（接你们公司鉴权）。
        - 通过RESTful API调用我们的转链接口
        - 通过RPC方式调用我们的转链方法（自己实现一个RPC版本的转链）
- 查看短链接：
    - 单独部署为一个服务（查看短链服务）
    - 通过nginx转发查看请求，`/[0-9a-zA-Z]*` --> 转发到我们的查链服务
    - 通过 access.log 收集（EFK）并统计访问数据

扩展

项目如何扩展?

1. 如何支持**自定义短链**?

   维护一个已经使用的序号，后续生成序号时判断是否已经被分配。
2. 如何让短链支持过期时间?

   每个链接映射额外记录一个『过期时间』字段，到期后将该映射记录删除。

   关于删除的策略有以下几种：
    1. 延迟删除：每次请求时判断是否过期，如果过期则删除。
        - 实现简单，性能损失小
        - 存储空间的利用效率低，已经过期得数据可能永远不会被删除
    2. 定时删除：创建记录时根据过期时间设置定时器
        - 过期数据能被及时删除，存储空间的利用率高
        - 占用内存大，性能差
    3. 轮询删除：通过异步脚本在业务低峰期周期性扫表清理过期数据
        - 兼顾效率和磁盘利用率
3. 如何提高吞吐量?

   整个系统分为『生成短链（写）』和『访问短链（读）』两部分
    1. 水平扩展多节点，根据序号分片
4. 延迟优化

   整个系统分为『生成短链（写）』和『访问短链（读）』两部分
    1. 存储层
        1. 数据结构简单可以直接改用kv存储
        2. 对存储节点进行分片
    2. 缓存层
        1. 增加缓存层，本地缓存-->redis缓存
        2. 使用布隆过滤器判断长链接映射是否已存在，判断短链接是否有效
    3. 网络

1. 基于地理位置就近访问数据节点

简历怎么写

[如何打造一份优雅的简历？ - Stefno - 博客园]

黄金法则：清晰！亮点！匹配度！ 👉 让别人看了你的简历就想约你面试！

项目介绍：说清楚你写了个什么项目，能干啥，有啥用，有数据支撑最好！

个人职责：写清楚你在项目里作为什么角色，做了哪些事情，抓住重点写！写到别人的心坎里。

项目收获：这个项目给公司带来啥收益，你从这个项目学到了啥。

短链接项目示例：

项目介绍：一个用于公司内部营销短信和App push的短链接服务，包含转链、存储、链接跳转功能，并可提供短链接点击数据报表功能。支撑了公司内部m条业务线，覆盖了全国n千万用户，从上线至今稳定运行x年。
个人职责：

- 负责项目的整体设计和开发, 负责实现转链和链接跳转模块逻辑。
- 基于MySQL主键实现了高可用的发号器组件。
- 在转链前进行特殊词过滤和防止循环转链的校验处理。
- 查看链接服务采用布隆过滤器防止缓存穿透，使用singleflight防止缓存击穿。
- 查看链接服务单独部署，Nginx转发请求通过EFK采集access日志方式统计短链接的点击数据。

项目收获

- 项目上线后支撑了xx条业务线，凭借这个项目获得当年公司内部的 **最佳新人/最快进步** 奖。
- 熟悉了常用的发号器设计方案，能够结合实际情况选择最适合的方案。
- 熟悉了go-zero框架的使用，对Go语言操作MySQL和Redis都更加熟悉。
- 锻炼了自己的自主学习能力，积累了项目设计和开发的经验。

面试怎么说

1. 说清楚项目背景
    1. 内部业务对外营销需要有一个短链接功能，并且能回收点击数据。
2. 说清楚项目架构
    1. 能够画出实际的架构图，并能够清楚的说出每个组件的功能。
3. 项目是如何部署的?
    1. 转链单独作为一个微服务部署。
    2. 其他项目通过 API接口和RPC方式接入（创建短链接）

3. 查看短链也是单独部署，接前面是nginx，通过nginx的access日志统计点击数据。

4. 说清楚项目实现过程中的重点和难点
    1. 为什么使用302跳转，而不使用301跳转？301与302的区别是什么？
        1. 需要记录访问数据，如果用301永久重定向跳转，下一次访问时浏览器有缓存就不再请求短链服务器了。这样会丢失访问数据。302是临时重定向，每次访问短链都会去请求短链服务器（除非响应中用 Cache-Control 或 Expired 暗示浏览器缓存）,虽然用 302重定向会给 server 增加一点压力，但是能准确记录每一次短链请求数据。
        2. 防浏览器缓存，领导让你把这个短链接ban了，你短链服务端删除了数据，但是浏览器有缓存还是能访问。
    2. 扩展问题：
        1. 常见的HTTP状态码？👉 https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status
        2. 分别介绍下303和307 👉 https://www.jianshu.com/p/70062192f26b
    3. 发号器的设计和实现
        1. 为什么要使用发号器的方案。
        2. 常见的发号器实现方式有哪些。
        3. 更进一步：如何实现高可用的发号器（MySQL主备+分片）
    4. 如何降低查看链接耗时?
        1. 加Redis缓存保存 短链接->长链接
        2. 再进一步：添加本地缓存构成多级缓存
    5. 如何解决缓存击穿问题?
        1. singleflight 合并请求
        2. 引申
            1. singleflight的实现原理
    6. 如何解决缓存穿透问题?
        1. 使用布隆过滤器过滤掉不存在的短链请求。
        2. 引申
            1. 布隆过滤器的实现原理 👉 https://www.cnblogs.com/zhaodongge/p/15017574.html
            2. 布隆过滤器的优点和缺点是什么?
            3. 怎么支持删除短链接?
                1. 使用布谷鸟过滤器
                2. 引申
                    1. 布谷鸟过滤器的实现原理 👉 http://www.linvon.cn/posts/cuckoo/
                    2. 布谷鸟过滤器的优点和缺点
            4. 布隆过滤器和布谷鸟过滤器的区别
                1. 算法：布隆过滤器多个hash函数。布谷鸟过滤器用布谷鸟哈希算法。布隆过滤器的多个哈希函数之间没关系。布谷鸟过滤器的两个哈希函数可互相推导，两者有关系，用到了异或操作。

2. 能否删除：布隆过滤器无法删除元素。布谷鸟过滤器可以删除元素，有误删可能。
3. 空间是否2的指数：布隆过滤器不需要2的指数。布谷鸟过滤器必须是2的指数。
4. 空间利用率：相同误判下，布谷鸟空间节省40%多。
5. 查询性能：布隆过滤器查询性能弱，原因是使用了多个hash函数，内存跨度大，缓存行命中率低。布谷鸟过滤器访问内存次数低，效率相对高。
6. 重复插入相同元素：布隆过滤器天然自带重复过滤。布谷鸟过滤器会发生挤兑循环问题。

5. 说清楚你在项目中的作用
    1. 负责项目的技术方案设计。
    2. 主导项目的开发过程，与同事协作完成需求的开发和上线。

参考资料

https://www.zhihu.com/question/29270034

https://hardcore.feishu.cn/docs/doccnAfY0f35ZgnrFg7jSTQmOOf

https://github.com/mxschmitt/golang-url-shortener

https://github.com/andyxning/shortme

https://code.flickr.net/2010/02/08/ticket-servers-distributed-unique-primary-keys-on-the-cheap/

https://www.geeksforgeeks.org/system-design-url-shortening-service/

https://medium.com/@sandeep4.verma/system-design-scalable-url-shortener-service-like-tinyurl-106f30f23a82

https://www.designgurus.io/blog/url-shortening

> 七米课堂 微服务实战课程 课件！ 更多Go学习内容👉 liwenzhou.com