**短地址服务分析设计**

# 分析研究

## 长转短发号器

由于长地址理论上的无限性，以及长对短哈希冲突的可能，决定了不能采用长短一 一哈希对应的方式。所以应该保证每一个长地址都对应一个唯一的短地址，否则逆 向转换时无法确定。所以最佳方式是用一个发号器对每一个长地址赋予一个永不重 复的token。

潜在方案设计如下，经过多方面思考最终选择第3种方案：

1. mysql自增主键方式：

理论上可行，但是频繁访问数据库性能较差，而且不能满足分布式高并发场景， 基本放弃。

1. redis原子自增函数方式：

由于redis的AOF和RDB持久化机制可保证当前值永不丢失，所以redis分布 式集群环境可以保证永远递增的token，并且存取性能较高，能够满足绝很多 分布式高并发场景。

但是它的缺点是RDB+AOF持久化机制在保证永不丢失的情况下就可能存在 宕机重启后的重复数据问题和启动瞬间的不可用问题。

所以这种方案在一般场景下可以使用，但是对性能和可靠性极度苛刻的环境需 要优化很多细节。

1. Twitter的Snowflake雪花算法：

它能保证高并发高可用的分布式环境下产生永不重复的递增编号，java进程内 产生，没有网络消耗，性能极高，天生支持分布式分段发号，具备较强的扩充 弹性，而且没有redis的那些缺点，但它又有自己的缺点：

1. 严格依赖机器时钟，存在时间回拨产生的重复数据问题。
2. 分布式环境中需要区别机器和服务id，涉及到比较复杂的机器和服务管理分配问题（比如zookeeper）。
3. workerId和serviceId存在一定的限制，如果加大它的阈值就会导致诸如token数值太大、序列号范围缩减等问题，不可兼得。
4. 产生的数据一般数值较大，存在号段浪费问题，不容易控制较小的短地址生成。
5. 生成的序号具有一定的递增规律，容易被黑客灌水和暴击，需要采用一种算法进行跨段或混淆。
6. Butterfly蝴蝶算法：

解决了上述方案存在的所有问题，而且性能超高，能够满足大部分的复杂 场景，但缺点是这种技术还未经过大量生产验证，目前还处于初级发展阶 段有待考核。

## 长转短存储

地址访问一般都在应用前端，需满足极高的并发性能，所以传统的DB存储方式基 本放弃，可以选用redis分布式集群进行存储，再配合Jedis客户端工具查询，性能 较高。

由于长地址海量存在，为了节约存储空间，不能采用简单粗暴的全量存储，应该采 用一种过期机制或淘汰机制，这些都是KV存储中的可优化点。

## 短转长

访问短地址时需要验证是否有效，从redis集群中获取对应的长地址，而且后端还 要有相应的历史记忆、自动预热和智能排序，优化存储，提高性能。这个功能应该 放在应用网关中，比如Nginx或GateWay，统一处理和转发。

# 设计实现

## SnowflakeGenerator发号器（雪花算法改进版）

1. 自定义“时间戳”、“serviced”、“workderId”、“序列号”的位数，这也就决定了各自的最大值，自定义具体的起始“时间戳”、“workerId”和“serviceId”，这些自定义用来实现分布式并发执行和控制生成数据的位数。
2. 定义一个ConcurrentHashMap<String,SnowflakeGenerator>类型的对象Map，相同serviceId对应的发号器不用重复生成。
3. 定义getInstance发号器对象生成器，供外部直接使用。
4. 核心方法nextId进行发号，同一时间戳内的序列号递增，跨时间戳的序列号归零，然后进行位移运算得到最终的token。

## SnowflakeConfig发号配置器

接收配置文件中“时间戳”、“serviced”、“workderId”、“序列号”的位数，做为构 建和选取发号器的参数。

## ShortUrlService长转短服务

1. 自动注入SnowflakeConfig发号配置器
2. 接收具体的“workerId”和“serviceId”属性参数，与SnowflakeConfig共同确定对应的发号器。
3. getSerialGenerator方法通过各个参数获取对应的发号器。
4. long2short长地址转短地址，先将长地址进行Md5并转16进制，从存储中获取对应的短地址，如果存在则返回，否则就调用getSerialGenerator获取对应的发号器并生成新token存储。

其中，先取再存的操作并不是原子操作，所以需要加锁，并且写入时判断是否 已存在。另外，因为10进制的token和62进制的shortUrl存在双向唯一对应 关系，所以存储token而不是shortUrl，节省存储空间。

## ShortUrlService短转长服务

先将62进制短地址转换为10进制的token，再从存储中获取对应的长地址。

## ShortUrlController控制器

1. long2short

调用服务直接获取长地址对应的token，再转换为短地址。

1. example

客户端访问短地址时被过滤器拦截，将短地址转换为token再获取对应的长地 址，然后自动跳转过去。

## ShortUrlFilter过滤器

doFilterInternal过滤所有的请求并进行验证和处理，如果是合法有效短地址请求则 将62进制的短地址转换为10进制的token再获取对应长地址，并自动跳转过去。

# 改进优化

## 发号器

发号器是重中之重，所有功能的成败好坏全在它，Twitter的Snowflake雪花算法固 然很好，但也有若干缺点，所以如果一个正式线上项目，应该对雪花算法进行改进 优化，或者选用更加强大高效的Butterfly蝴蝶算法。

## token递增

发号器递增有优点也有缺点，在保证唯一的前提下，尽量消除规律性，防范黑客， 保护安全。

## 号段浪费

目前雪花算法生成的短地址位数并不是从0开始，存在很大的号段浪费，这可能是 这个算法固有的弊端，即便采用更先进的Butterfly蝴蝶算法也存在类似问题，可 以采用一种二次发号的算法，对短地址幂等性缩减。

## 长度控制

目前雪花算法和其它高效的算法生成的短地址都很难精确控制生成的长度，或者性 能高效或者长度不可控，很难兼得，还需进一步研究。

## 发号配置器

发号配置器应该全网通用，所以可以将其放入通用配置中心，而每个机器每个服务 在具体的配置文件中单独配置，这些共性和个性应该区分清楚。

## KV存储

长短地址KV存储也是个重要问题，应该支持增量或部分存储，后端应该采用一种 过期机制或淘汰机制，还要有相应的历史记忆、自动预热和智能排序，优化存储和 查询。或者采用spark等实时数据智能分析和推荐技术。

## 失效KV

应该对已失效并重建的KV进行处理，将旧的短地址转换为新的短地址，保证实时 应用。

## 地址转发

短转长的逻辑应该放在应用网关中，比如Nginx或GateWay，统一处理和转发，从 源头进行url过滤筛选和forward，并且对无效或非法url进行鉴别和记录。

# Swagger API文档



# Jacoco单元测试覆盖率报告

