

短域名方案及实现

1、需求

实现短域名服务（细节可以百度/谷歌）

撰写两个 API 接口：

- 短域名存储接口：接受长域名信息，返回短域名信息
- 短域名读取接口：接受短域名信息，返回长域名信息。

限制：

- 短域名长度最大为 8 个字符
- 采用SpringBoot，集成Swagger API文档；
- JUnit编写单元测试，使用Jacoco生成测试报告(测试报告提交截图)；
- 映射数据存储在JVM内存即可，防止内存溢出；

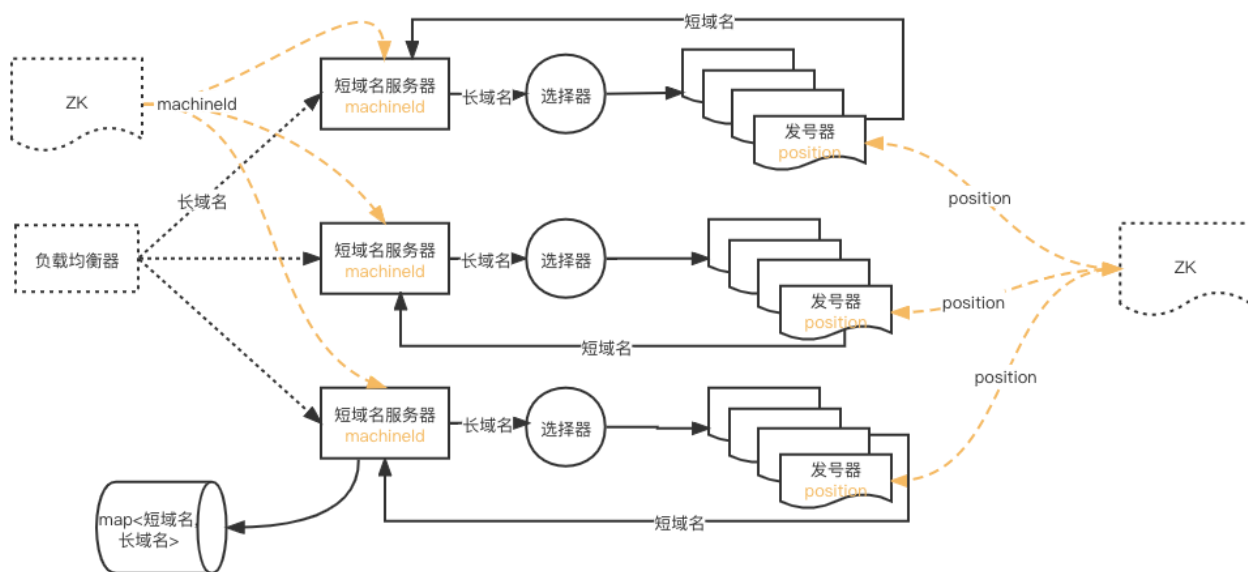
通过互联网了解到短域名的知识：

<https://leetcode-cn.com/circle/discuss/EkCOT9/> <<https://leetcode-cn.com/circle/discuss/EkCOT9/>>

<https://segmentfault.com/a/1190000012088345>
<<https://segmentfault.com/a/1190000012088345>>

<https://juejin.cn/post/6844904152338808845>
<<https://juejin.cn/post/6844904152338808845>>

2、系统架构



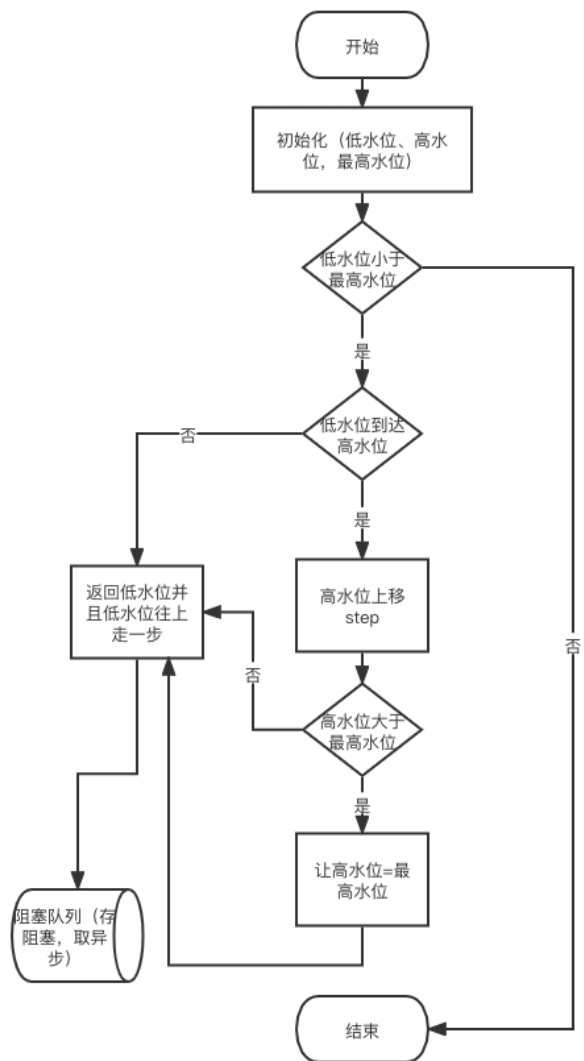
解释：

- 1、为了提高短域名的服务，采用分布式的架构
- 2、每台服务从ZK获取自己唯一的ID,最前面可用Nginx等负载均衡器进行分流
- 3、每个服务里面有N个发号器，发号器的实现可以有多种实现。发号器实质是一个码号产生器，可用UUID、HASH、自增等编码方式，发号器的号码生成规则可以使用单一或组合
- 4、选择器从N个发号器中选择一个作为T，由T获取一个或多个编码返回
- 5、最后由存储服务将长、短对应关系进行存储（本文简单使用guava缓存存储），提供后续的查询
- 6、对于8位短链符号，机器编号占N位，发号器编号暂M位，号码占用K位,满足 $M+N+K=8$

3、具体实现&核心算法

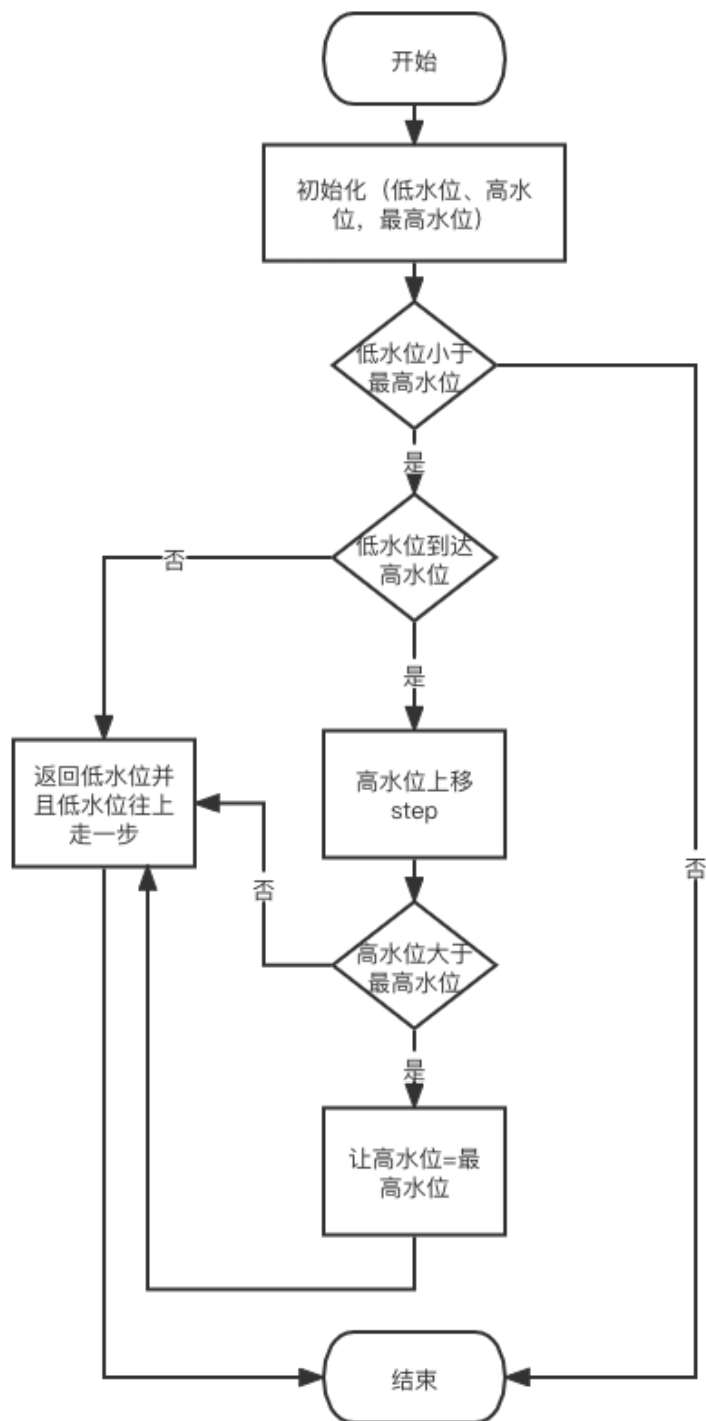
3.1 自增发号器实现(有缓存)

```
1 public CacheQueueNumberGenerator(Long id, Long highMaxConfig, Long stepConfig, Long lowMaxConfig, Long lowConfig) {
2
3     this.id = id;
4     this.highMax = highMaxConfig;
5     this.step = stepConfig;
6     //todo 判断是否有持久化的high,如果没有都初始化成0
7     Long highConfig = 0L;
8     this.high = highConfig;
9     this.low = highConfig;
10
11     queue = new ArrayBlockingQueue<>(queueSizeConfig);
12
13     //后台线程
14     Thread thread = new Thread() -> {
15         while (true) {
16             if (this.low >= this.highMax) {
17                 log.error("已用尽号码,停止后台线程,id={},low={},high={},step={},highMax={}",
18                     id, low, high, step, highMax);
19                 return;
20             }
21             //如果低水位达到高水位,高水位上移
22             if (low >= high) {
23                 high += step;
24                 //高水位不可越过最高水位
25                 if (high >= highMax) {
26                     high = highMax;
27                 }
28                 //todo 持久化记录high
29             }
30             try {
31                 //采用阻塞方式放入元素
32                 queue.put(low);
33             } catch (InterruptedException e) {
34                 log.info("[CacheQueueNumberGenerator] daemon thread interrupted", e);
35             }
36             //低水位移动
37             low++;
38             //todo 可以加提前预警, 比如low>highMax*0.8
39         }
40     };
41     thread.setDaemon(true);
42     thread.start();
43 }
```



3.2自增发号器实现(无缓存)

```
1 public synchronized Long generateCode() {
2     if (low >= highMax) {
3         log.error("已用尽号码,停止服务,low={},high={},step={},highMax={}", low, high, step);
4         return null;
5     }
6     //如果低水位达到高水位,高水位上移
7     if (low >= high) {
8         high += step;
9         //高水位不可越过最高水位
10        if (high >= highMax) {
11            high = highMax;
12        }
13        //todo 持久化记录high
14    }
15    Long temp = low;
16    //低水位移动
17    low++;
18    //todo 可以加提前预警, 比如low>highMax*0.8
19    return temp;
20 }
```



3.3 选择器实现(随机)

原理：将所有发号器放入数组array，使用随机数生成器在数组array范围内生成一个随机下标即可

```

1 public NumberGenerator selectOneRandom(List<NumberGenerator> numberGeneratorList) {
2     //加读锁
3     try {
4         rwl.readLock().lock();
5         if (numberGeneratorList == null || numberGeneratorList.size() <= 0) {
6             return null;
7         }
8         int rnd = new SecureRandom().nextInt(numberGeneratorList.size());
9         return numberGeneratorList.get(rnd);
10    } finally {
11        rwl.readLock().unlock();
12    }
13
14 }

```

3.4 选择器实现(权重)

原理：将所有发号器放入数组array，权重大的放入多次，选择步骤同3.3同

```

1 public NumberGenerator selectOneWeight(List<NumberGenerator> numberGeneratorList) {
2     //加读锁
3     try {
4         rwl.readLock().lock();
5         if (numberGeneratorList == null || numberGeneratorList.size() <= 0) {
6             return null;
7         }
8         int[] weight = {0, 0, 0, 0, 1, 2, 3, 4, 5, 5};
9         int rnd = new SecureRandom().nextInt(weight.length);
10        return numberGeneratorList.get(weight[rnd]);
11    } finally {
12        rwl.readLock().unlock();
13    }
14
15 }

```

4、单元&功能测试

配置说明 (application.properties) :

#每位编号的进制

app.config.redix = 62

#短链接总长度

app.config.totalBit = 8

#机器占位

app.config.machineBit = 1

#计数器占位

app.config.counterBit = 1

#缓存过期时长

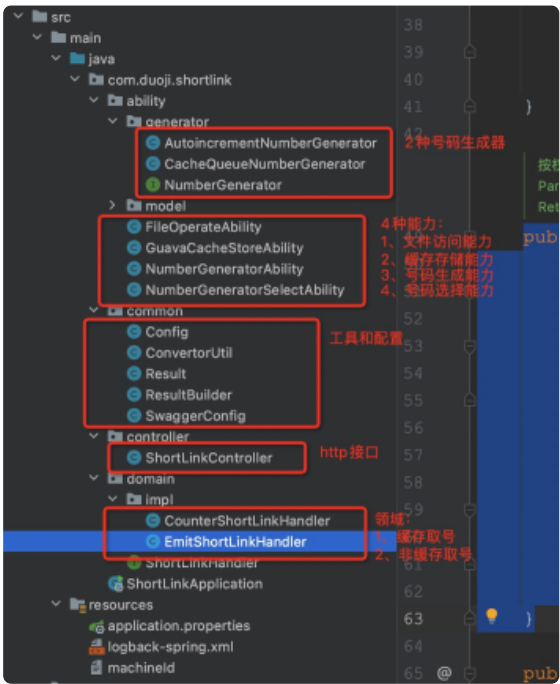
app.config.expireSec = 3600

发号器选择 (emit、counter)

app.model = emit

特变注意,还需要新建一个machineId文件,里面是当前微服务的ID。这里可以加拓展, 用 zooKeeper记录机器ID每台服务启动时去zookeeper获取

4.1 项目目录



4.2 单元测试完成度

localhost:63342/short-link/target/site/jacoco/index.html?_ijt=84q7gp02ernvoPl8tblkeu3le

应用 办公网站

short-link

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
com.duojishortlinkability	<div></div>	92%	<div></div>	92%	3 33	6 85	1 20	0 4
com.duojishortlink	<div></div>	46%	<div></div>	n/a	1 3	3 5	1 3	0 1
com.duojishortlinkcommon	<div></div>	97%	<div></div>	100%	2 20	2 41	2 15	0 3
com.duojishortlinkdomainimpl	<div></div>	96%	<div></div>	66%	4 11	2 35	0 5	0 1
com.duojishortlinkabilitygenerator	<div></div>	98%	<div></div>	91%	1 15	2 47	0 9	0 2
Total	55 of 1,092	94%	7 of 60	88%	11 82	15 213	4 52	0 11

4.3 功能测试

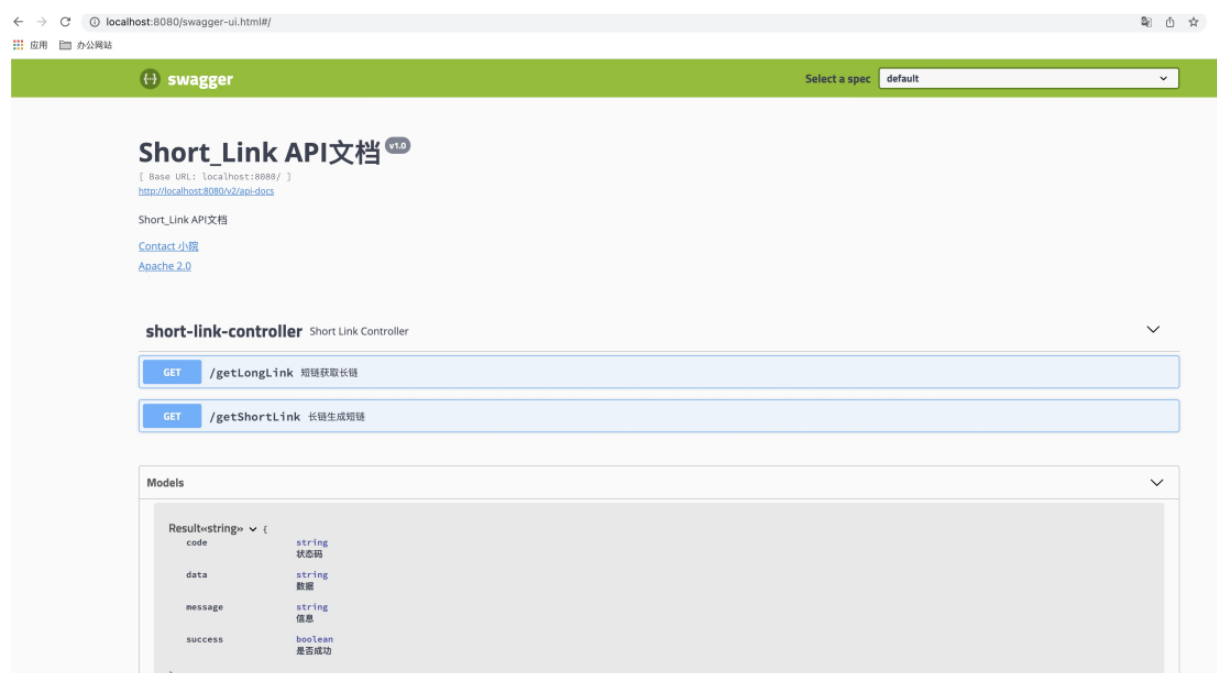
1、获取短链



2、通过短链获取长链



4.4 swagger API



5、性能测试

测试工具: jmeter

机器配置 (办公电脑,开了其他软件,如idea) :

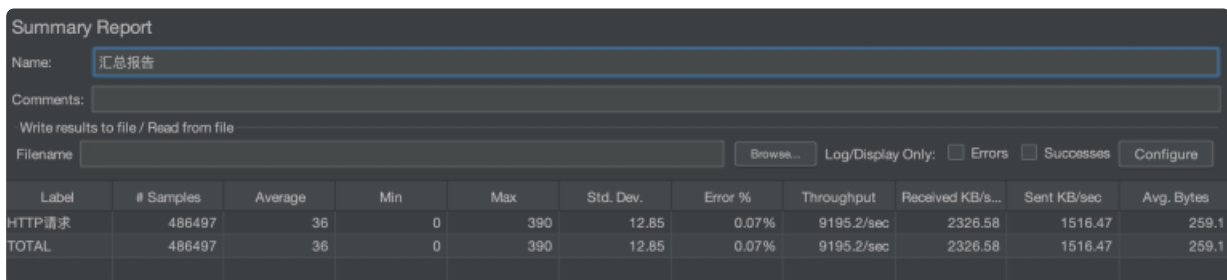
处理器: 2.6 GHz 六核Intel Core i7

内存: 16 GB 2667 MHz DDR4

JVM配置(保守配置, 用2G堆内存, 1.6G年轻代):

- 版本: JDK8
- JVM设置: -Xms2048M -Xmx2048M -Xmn1640M -Xss1M -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=5 -XX:PretenureSizeThreshold=1M -XX:+UseParNewGC -XX:+UseConcMarkSweepGC
- 62个缓存队列

测试结果:



Summary Report

Name: 汇总报告

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
HTTP请求	486497	36	0	390	12.85	0.07%	9195.2/sec	2326.58	1516.47	259.1
TOTAL	486497	36	0	390	12.85	0.07%	9195.2/sec	2326.58	1516.47	259.1

平均响应时间: 36ms,最大响应时间390ms

QPS: 9195

发现: 性能瓶颈在tomcat上, 后端开了62个缓存队列, 只有0号队列在返回数据, 也就是说所有队列都返回的情况下, 理论上可达到 $9k \times 62 = 54w$ QPS

6、展望

1. 机器ID获取可以改进, 使用zookeeper注册并动态获取。(架构图中虚线部分)
2. 由于没有持久化, 每次重启服务, 编码又从0开始计数。这里可以在生成器中, 每次加步长的时候持久化高水位, 之后再启动直接读上次的高水位。(架构图中虚线部分)
3. 由于没有持久化, 使用缓存记录长、短的对对应关系, 后续可用分库分表记录这种对应关系, 并加上布隆过滤器防止同链接刷单。
4. 单个发号器的码号达到最高水位80%时, 可以加上报警功能, 提前感知。
5. 此系统可拓展性比较强, 可以用来当唯一编号获取系统, 比如为订单ID的获取。
6. 性能瓶颈不在发号器上, 在tomcat上, 可以继续优化tomcat

7、作者信息

基本信息

姓名：肖源 地址：杭州市五常街道福鼎家园 电话：13657235345 Email：
1214118459@qq.com

教育背景

2014.9--2017.6 武汉理工大学 计算机学院（硕士） 计算机科学与技术

2010.9--2014.6 武汉理工大学 计算机学院（本科） 物联网工程

工作经历

2020.8.28-至今 工作单位: 阿里巴巴-同城-物流 职位：高级JAVA开发工程师

主要工作：参与近端履约的揽、收、分系统的开发和架构工作，主要使用HSF、分库分表、Tair、Schedulex等技术。

2017.7.3-2020.8.26 工作单位:武汉斗鱼 职位：大数据开发工程师

主要工作：参与广告RTB实时竞价系统、用户行为收集系统、主播排名系统、任务调度平台的开发，主要使用Hive、Storm、Kafka、Redis、SpringBoot等技术。

2016.6.27-2016.9.2 工作单位：阿里巴巴-阿里健康 职位：java开发实习生

主要工作：熟悉阿里内部RPC中间件hsf、消息队列notify、分布式数据库中间件tddl等，参与阿里健康APP4.0的后台日志处理、打赏功能开发、天猫项目共建。