需求分析

这里的短域名服务，我理解为是将长URL地址（域名[服务名[请求参数]]），经过转码，变为长度最大为8个字符的字符串，并且能够反向获取原始URL地址。

关键点一：如何判定接收的URL地址已经在数据库存在记录？

计算URL字符串哈希值，用哈希值在数据库查询，建立索引，没有数据表示不存在，如果有数据，用URL对结果集进行过滤匹配，如果没有匹配数据说明都是哈希冲突的记录，针对哈希冲突的数据，在进行转码字符串时用哈希加序号的方式生成短链接key。

关键点二：如何生成编码，如何转成字符串？

考虑第一点时，计算了URL哈希值，这里可以继续用这个哈希值作为编码，然后转成62进制字符串，6位的62进制字符串可以表示568亿条数据。

针对哈希冲突的数据，可以在6位62进制字符串基础上增加1-2位计数字符。

关键点三：反向获取原始URL地址时，快速判定是否存在以及缓存处理

接收短链接key参数，先查询缓存中短链接key对应的原始URL有没有数据，如果没有再查询数据库，然后缓存结果。扩展方案可以在查询之前先用布隆过滤器检查，判定不存在的直接缓存空结果。

系统设计

采用前后端分离模式开发，前端基于Vue框架做一个API调用客户端，后端Spring Boot应用。

数据库

使用MySql数据库，sql脚本如下：

```
CREATE TABLE `short_url` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `short_key` varchar(10) DEFAULT NULL COMMENT '短链接key，不含域名，建唯一索引',
  `long_url` varchar(500) DEFAULT NULL COMMENT '原始url',
  `long_hash_code` bigint(20) DEFAULT NULL COMMENT '原始url的哈希值',
  `seq` int(11) DEFAULT NULL COMMENT '原始url的哈希值冲突后，计算短链接key的计数序号',
  `create_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `index_short_url_short_key` (`short_key`),
  KEY `index_short_url_long_hash_code` (`long_hash_code`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

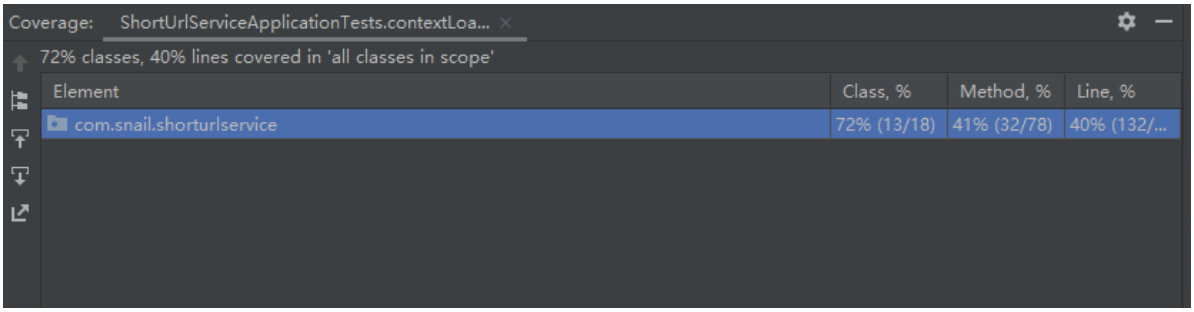
缓存

Redis缓存，短链接字符串做key，原始url做value。过期删除策略代码里使用的时缓存60秒，每次访问有效数据key时延长60秒。淘汰策略使用volatile-lfu，从所有配置了过期时间的键中驱逐使用频率最少的键。

算法

转码算法使用32位的murmur3非加密哈希算法计算出哈希值，然后将哈希值转为6位62进制的字符串表示。针对哈希冲突数据，用计数序号转换为62进制字符串后追加到6位哈希字符串后边，组成最终的短链接key。

单元测试覆盖率



扩展方案

算法优化

增加缓存，将最近提交频率高的URL缓存起来，减少生成短链接时查询数据库的压力。增加布隆过滤器，对不存在的数据进行快速响应。

分布式锁

随数据量扩大，多线程或多服务出现同一个哈希冲突的数据时，可以考虑使用分布式锁，避免出现数据入库失败，返回给调用方错误信息。

数据库扩展

分库分表，可以拿哈希值字段作为分库分表的字段。

定期清理过期数据

增加统计分析功能，统计数据的访问频率，定时任务对长期不被访问的数据进行删除淘汰。