Please visit website: http://cxyroad.com

如何设计一套高性能的短链系统?

你好，我是猿java

如上图，对于这种客评短信，相信大家并不陌生，通过点击短信里“蓝色字体”，就能跳转到一个网页。其实，背后的秘密就是一套完整的短链系统，今天我们就来看看腾讯的后端女生是如何设计的?

上图中那串蓝色字符，有个专业的术语叫做“短链”，它可以是一个链接地址，也可以设计成二维码。

为什么要用短链?

存在既合理，这里列举 3个主要原因。

1.相对安全

短链不容易暴露访问参数，生成方式可以完全迎合短信平台的规则，能够有效地规避关键词、域名屏蔽等风险，而原始 URL地址，很可能因为包含特殊字符被短信系统误判，导致链接无法跳转。

2.美观

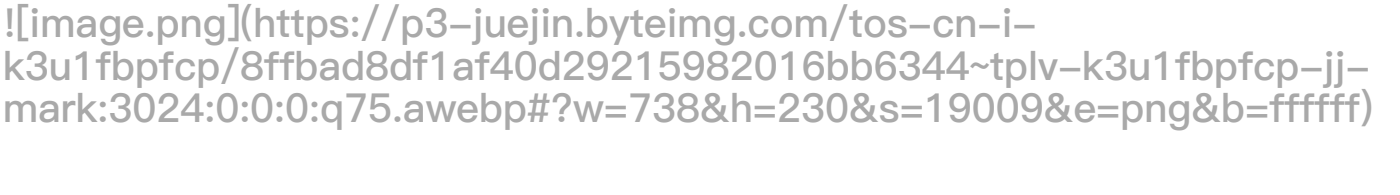
对于精简的文字，似乎更符合美学观念，不太让人产生反感。

3.平台限制

短信发送平台有字数限制，在整条短信字数不变的前提下，把链接缩短，其他部分的文字描述就能增加，这样似乎更能达到该短信的实际目的（比如，营销）。

短链的组成

如下图，短链的组成通常包含两个部分：域名 + 随机码



短链的域名最好和其他业务域名分开，而且要尽量简短，可以不具备业务含义（比如：xyz.com），因为短链大部分是用于营销，可能会被三方平台屏蔽。

短链的随机码需要全局唯一，建议 10位以下。

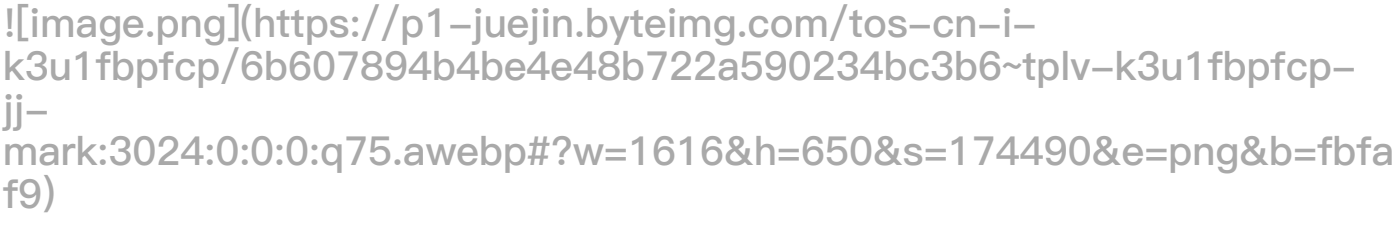
短链跳转的原理

首先，我们先看一个短链跳转的简单例子，如下代码，定义了一个 302重定向的代码示例：

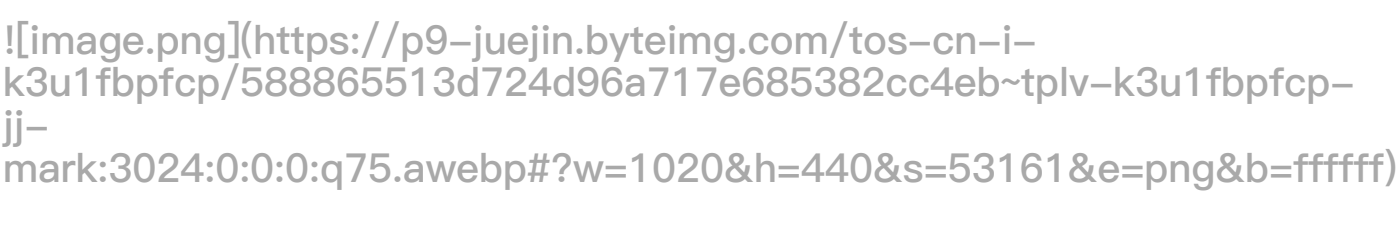
```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.servlet.view.RedirectView;

@Controller
public class RedirectController {

  @GetMapping("/{shortCode}")
  public RedirectView redirect(@PathVariable String shortCode) {
    String destUrl = "https://yuanjava.com";
```

```
    // destUrl = getDestUrlByShortCode(shortCode); //真实的业务逻辑
    return new RedirectView(destUrl);
  }
}
```

接着，在浏览器访问短链”[http://127.0.0.1:8080/s2TYdWd”](http://cxyroad.com/”http://127.0.0.1:8080/s2TYdWd%E2%80%9D”) 后，请求会被重定向到[yuanjava.com](http://cxyroad.com/ ”https://yuanjava.com”)，下图为浏览器控制台信息：



从上图，我们看到了 302状态码并且请求被 Location到另外一个 URL，整个交互流程图如下：



是不是有一种偷梁换柱的感觉？？？

最后，总结下短链跳转的核心思想：

生成随机码，将随机码和目标 URL（长链）的映射关系存入数据库；

用域名+随机码生成短链，并推送给目标用户；

当用户点击短链后，请求会先到达短链系统，短链系统根据随机码查找出对应的目标 URL，接着将请求 302重定向到目标 URL（长链）；

关于重定向有 301 和 302两种，如何选择?

* 302，代表临时重定向：每次请求短链，请求都会先到达短链系统，然后重定向到目标 URL（长链），这样，方便短链系统做一些统计点击数等操作；通常采用 302
* 301，代表永久重定向：第一次请求拿到目标长链接后，下次再次请求短链，请求不会到达短链系统，而是直接跳转到浏览器缓存的目标 URL（长链），短链系统只能统计到第一次访问的数据；一般不采用 301。

如何生成短链?
=======

从短链组成章节可以知道`短链=域名+随机码`，因此，如何生成短链的问题转换成了如何生成一个随机码，而且这个随机码需要全局唯一。通常有 3种做法：

Base62

Base62 表示法是一种基数为62的数制系统，包含26个英文大写字母（A-Z），26个英文小写字母（a-z）和10个数字（0-9）。这样，共有62个字符可以用来表示数值。 如下代码：

```
import java.security.SecureRandom;

public class RandomCodeGenerator {
  private static final String CHAR_62 =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
  private static final SecureRandom random = new SecureRandom();

  public static String generateRandomCode(int length) {
    StringBuilder sb = new StringBuilder(length);
    for (int i = 0; i < length; i++) {
      int rndCharAt = random.nextInt(CHAR_62.length());
      char rndChar = CHAR_62.charAt(rndCharAt);
      sb.append(rndChar);
    }
    return sb.toString();
  }
}
```

```

对于 Base62算法，如果是生成 6位随机数有 62^6 - 1 = 56800235583, 568亿多，如果是生成 7位随机数有 62^7 - 1 = 3521614606208，合计3.5万亿多，足够使用。

Hash算法
------

Hash算法算法是我们最容易想到的办法，比如 MD5, SHA-1, SHA-256, MurmurHash, 但是这种算法生成的 Hash算法值还是比较长，常用的做法是把这个 Hash算法值进行 62/64进行压缩。

如下代码，通过 Google的 MurmurHash算法把长链 Hash成一个 32位的 10进制正数，然后再转换成62进制（压缩），这样就可以得到一个 6位随机数，

```
import com.google.common.hash.HashFunction;
import com.google.common.hash.Hashing;
import java.nio.charset.StandardCharsets;

public class MurmurHashToBase62 {

    private static final String BASE62 =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    public static String toBase62(int value) {
        StringBuilder sb = new StringBuilder();
        while (value > 0) {
            sb.insert(0, BASE62.charAt(value % 62));
            value /= 62;
        }
        return sb.toString();
    }
    public static void main(String[] args) {
        // 长链
        String input = "https://yuanjava.cnposts/short-link-system/design?code=xsd&page=1";
        // 长链利用 MurmurHash算法生成 32位 10进制数
        HashFunction hashFunction = Hashing.murmur3_32();
        int hash = hashFunction.hashString(input,
```
```

```
StandardCharsets.UTF_8).asInt();
        if (hash < 0) {
            hash = hash & 0x7fffffff; // Convert to positive by dropping the
sign bit
        }
        // 将 32位 10进制数 转换成 62进制
        String base62Hash = toBase62(hash);
        System.out.println("base62Hash:" + base62Hash);
    }
}
```

全局唯一 ID

比如，很多大中型公司都会有自己全局唯一 ID 的生成服务器，可以使用这些服务器生成的 ID来保证全局唯一，也可以使用雪花算法生成全局唯一的ID，再经过 62/64进制压缩。

如何解决冲突
===

对于上述3种方法的前 2种：base62 或者 hash，因为都是哈希函数，所以，不可避免地会产生哈希冲突（尽管概率很低），该怎么解决呢?

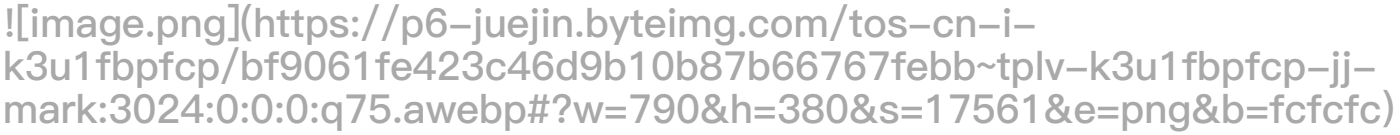
要解决冲突，首先要检测冲突，通常来说有 3种检测方法。

利用数据库锁

如下，这里以 MySQL数据库为例（也可以保存在 Redis中），表结构如下：

```
CREATE TABLE `short_url_map` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`long_url` varchar(160) DEFAULT NULL COMMENT '长链',
`short_url` varchar(10) DEFAULT NULL COMMENT '短链',
`gmt_create` int(11) DEFAULT NULL COMMENT '创建时间',
  PRIMARY KEY (`id`),
```

```
  UNIQUE INDEX 'short_url' ('short_url')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```

首先创建一张长链和短链的关系映射表，然后通过给 short\_url字段添加唯一锁，这样，当数据插入时，如果存在 Hash冲突（short\_url值相等），数据库就会抛错，插入失败，因此，可以在业务代码里捕获对应的错误，这样就能检测出冲突。

也可以先用 short\_url去查询，如果能查到数据，说明 short\_url存在 Hash冲突了。

对于这种通过查询数据库或者依赖于数据库唯一锁的机制，因为都涉及DB操作，所以对数据库是一个开销，如果流量比较大的话，需要保证数据库的性能。

布隆过滤器过滤器
---

在 DB操作的上游增加一个布隆过滤器，在长链生成短链后， 先用短链在布隆过滤器中进行查找，如果存在就代表冲突了，如果不存在，说明 DB里不存在此短链，可以插入。 对于布隆过滤器的选择，单机可以采用 Google的布隆过滤器，分布式可以使用 RedisBloom。

整体流程可以抽象成下图：




检测出了冲突，需要如何解决冲突?

再 Hash，可以在长链后面拼接一个 UUID之类的随机字符串，然后再次进行 Hash，用得出的新值再进行上述检测，这样 Hash冲突的概率又大大大的降低了。

表设计
===
```

在整个短链系统中，最核心的表就是 长链和短链的映射关系表，表设计如下：

```
CREATE TABLE `short_url_map` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`long_url` varchar(160) DEFAULT NULL COMMENT '长链',
`short_url` varchar(10) DEFAULT NULL COMMENT '短链',
`gmt_create` int(11) DEFAULT NULL COMMENT '创建时间',
  PRIMARY KEY (`id`),
  UNIQUE INDEX 'short_url' ('short_url')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

需要对短链字段`short_url`添加一个唯一索引，这样的话，一方面可以保证short_url全局唯一，一方面可以通过索引加快以下查询语句的速度：

```
select * from short_url_map  where short_url = ?
```

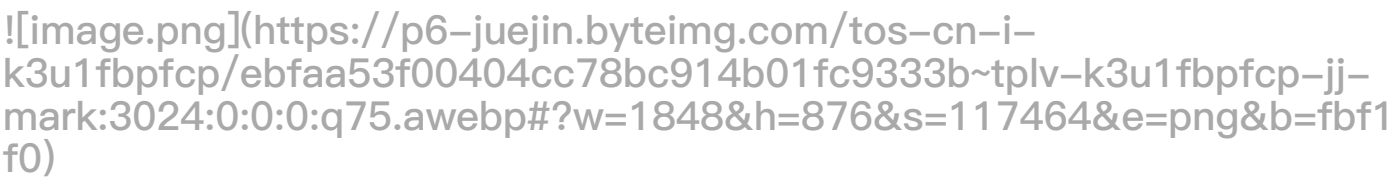
高并发场景
======

设计思路

在流量不大的情况，上述方法怎么折腾似乎都没有问题，但是，为了架构的健壮性，很多时候需要考虑高并发，大流量的场景，因此架构需要支持水平扩展，比如：

* 采用微服务
* 功能模块分离，比如，短链生成服务和长链查询服务分离
* 功能模块需要支持水平扩容，比如：短链生成服务和长链查询服务能支持动态扩容
* 缓解数据库压力，比如，分区，分库分表，主从，读写分离等机制
* 服务的限流，自保机制
* 完善的监控和预警机制

这里给出一套比较完整的设计思路图：



分库分表

关于短链和长链映射关系表的分库分表是一个重点，这里需要详细分析。

是否需要分库分表

在做技术架构时，很忌讳过度设计，因此，对于高并发场景，是否需要分库分表，分多少个库，分多少个表，分库分表键如何选择等问题都应该根据具体业务数据量进行评估。

分库分表键需要如何选择

如果需要分库分表，库和表的 PartitionKey 该如何选择?

方法一：短链码进行 hash取模

如下算法，确认库和表的路由规则：

```
    库ID = 短链的 hash值 % 库数量
    表ID = 短链的 hash值 / 库数量  % 表数量
```

该方法需要根据业务的数据量以及库表设计需要支持几年的数据总量来评估出库的数量和表的数量，另外，因为短链数据绝大多数都是一次性的，所以可以对存量数据进行归档，这样可以解决数据过多需要扩容的问题。

该方案的优缺点：

优点：

1. 分库分表方式清晰易懂

缺点：

1. 扩容比较困难，扩容时需要迁移大量的数据；
2. 最开始时就需要把库和表全部创建好，对于前期数据量不多的时候，是一种浪费；

那么，有没有一种好的方式，可以支持动态扩容而且尽量不牵涉到数据的迁移呢？这里我们就要看第二种方案：

方法二：支持动态扩容

通过方法一，我们可以知道，库和表是动态计算出来的，能不能我们固定设置库和表的标号呢？基于这个想法，我们设计了如下的方案，在随机码的前面增加一位代表库的标号，在随机码的后面增加一位代表表的标号，如下图：

这样数据库可以支持62个，每个库的表可以支持62张表，按照每张表 2000万条数据，支持的总数据 = 62 * 62 * 2000w = 768.8亿，如果还不够用的话，那可以在随机码的前后各增加两位来表示库和表，这样就足够了。

实现细节

预先配置分库分表中库和表的标号，比如：库标号 [0,1,2]，表标号 [0,1,2,3]，通过上面的方法获取到一个随机码之后，然后从库标号 [0,1,2]随机获取一个标号，拼接在随机码的前面作为库标识，从表标号 [0,1,2,3]随机获取一个标号，拼接在随机码的后面作为表，然后在做分库分表陆路由的时候，分别截取第一位和最后一位作为库和表的路由编号。

注意，这里是随机获取，也可以使用轮询算法获取库标号和表标号。

扩容

假如，需要对库标号 [0,1,2]，表标号 [0,1,2,3]进行扩容，只需要将标号添加进去，比如：库标号[0,1,2,3]，表标号 [0,1,2,3,4,5]，这样原始的数据不需要进行迁移就完成了库容操作。

该方案的优缺点

优点：

1. 支持动态扩容
2. 动态扩容时不需要迁移数据

缺点

1. 需要在随机码前后增加库和表的标识，增加了短链的长度
2. 库标识和表标识添加的算法，直接影响数据的离散性

更多关于分库分表的细节，可以参考文章 [大厂实例分享：每日 3000万订单，如何分库分表？](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzIwNDAyOTI2Nw==&mid=2247484608&idx=1&sn=c2deb84f4dabf5838fafe77ab274dfba&chksm=96c728fca1b0a1ea307ac9aefae146d39e6367732537d460d4a1581696bf05a7729201604ea4&token=1153290687&lang=zh_CN#rd”)

总结
==

本文通过一个客服评价的短信开始，分析了短链的构成，短链跳转的原理，同时也给出了业内的一些实现算法，以及一些架构上的建议。

对于业务体量小的公司，可以根据成本来搭建服务（单机或者少量服务器做负载），对于业务体量比较大的公司，更多需要考虑到高并发的场景，如何保证

服务的稳定性，如何支持水平扩展，当服务出现问题时如何具备一套完善的监控和预警服务器。

其实，很多系统都是在一次又一次的业务流量挑战下成长起来的，我们需要不断打磨自己宏观看架构，微观看代码的能力，这样自己也就跟着业务，系统一起成长起来了。

原创好文
====

* [腾讯女后端设计了一套短链系统，当场就想给她offer！](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzIwNDAyOTI2Nw==&mid=2247495742&idx=1&sn=8ce136a30392b97b86556ef363b7b267&chksm=96c4dc02a1b35514d90cd30e648e1f9a155d43735e716f6ba9861f618de1ea2552bc2d3d1e1d&token=1175502933&lang=zh_CN#rd”)
* [肝了一周，彻底弄懂了 CMS收集器原理，这个轮子造的真值！](http://cxyroad.com/”https://link.juejin.cn?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247494932%26idx%3D1%26sn%3D1e0a7a74e947dd03967c36cc3270f8a1%26chksm%3D96c4c128a1b3483e78405fa2b1cf2f2f901b91593c56fc4d6ee086eb1b7c8a27e4d9be1fbf7d%26token%3D2016670857%26lang%3Dzh_CN%23rd”)
* [9款常见的 JVM垃圾回收器](http://cxyroad.com/”https://link.juejin.cn?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247491611%26idx%3D1%26sn%3De3ea710a6e0bad4035254f731847b8fc%26chksm%3D96c4cc27a1b34531653701a7e96b62b80bbc53bea087f84acbd3f165a7cd71cd78a159b91d7f%26token%3D1912578422%26lang%3Dzh_CN%23rd”)
* [美团一面：Git 是如何工作的？（推荐阅读）](http://cxyroad.com/”https://link.juejin.cn?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247491037%26idx%3D1%26sn%3D2351e0e54dcbf16a2575e353ed41734d%26chksm%3D96c731e1a1b0b8f73027d20dc84e60760a195d46e106fcbddf98a945d125c453990f805a2769%26token%3D1593775808%26lang%3Dzh_CN%23rd”)
* [阿里 P7二面：Redis 执行 Lua，能保证原子性吗？](http://cxyroad.com/”https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247485640%26idx%3D1%26sn%3D68b6a1bf4873a394b7a903d6a15ba697%26chksm%3D96c724f4a1b0ade21c40da5b73dbf36851de848ed7b3d0af203ef7fead81cb850e3242351402%26token%3D1151064718%26lang%3Dzh_CN%23rd”)

* [当下环境，程序员需要修炼的 3项技能](http://cxyroad.com/”https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247485548%26idx%3D1%26sn%3D9fbabe3280e88810859975f44140e15d%26chksm%3D96c72450a1b0ad46a0741e0f4369424fda76d8906750ef87e58839ce97ffd0381c84c4492f6d%26token%3D595465147%26lang%3Dzh_CN%23rd”)
* [AI是打工人的下一个就业风口吗?](http://cxyroad.com/”https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247485571%26idx%3D1%26sn%3De00066ca6bd2f5df2bcf0fe2b5a1f340%26chksm%3D96c724bfa1b0ada9ae2b3116cf0805f48edd2aef81c9e4ebfca386a8014cc81d800cc21283fe%26token%3D595465147%26lang%3Dzh_CN%23rd”)
* [和斯坦福博士写代码的一个月](http://cxyroad.com/”https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247485310%26idx%3D1%26sn%3D98111850b5aab60f129b2414f355fef6%26chksm%3D96c72b42a1b0a254ae6a6bc4ffafbc9a11b6b93a22044fd51ce647b9aef618b89e710d1eb59d%26token%3D595465147%26lang%3Dzh_CN%23rd”)
* [肝了一周，这下彻底把 MySQL的锁搞懂了](http://cxyroad.com/”https://link.juejin.cn/?target=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMzIwNDAyOTI2Nw%3D%3D%26mid%3D2247484561%26idx%3D1%26sn%3D2d021a42fd74163367b121eb0b98b53f%26chksm%3D96c728ada1b0a1bb7660c3918a684b288405da848bf9602dfaaaa8c6554d6c3148d864ee4f75%26token%3D595465147%26lang%3Dzh_CN%23rd”)
 原文链接: https://juejin.cn/post/7350585600858898484