一致性哈希算法与短网址系统
Consistent Hashing & Design Tiny Url

课程版本 v5.1　本节主讲人 东邪

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
微博: http://www.weibo.com/ninechapter
知乎: http://zhuanlan.zhihu.com/jiuzhang
官网: http://www.jiuzhang.com

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像

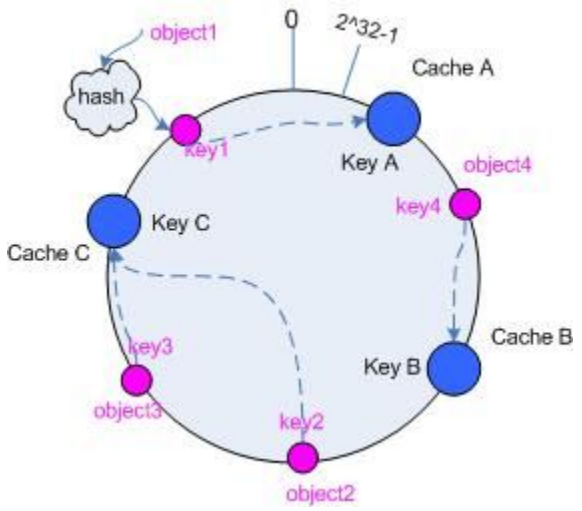
一经发现，将被追究法律责任和赔偿经济损失

今日课程大纲

- Consistent Hashing
  - 简单的Consistent Hashing方法的回顾及缺陷分析
  - 一个更优的 Consistent Hashing 方法
- Replia
  - SQL 通常如何进行备份
  - NoSQL 通常如何进行备份？
- 设计一个短网址系统 Design Tiny Url
  - 4S 分析法
  - No Hire / Weak Hire / Hire / Strong Hire
-
- 附录(供大家自学):
- 当你访问 www.google.com 的时候发生了什么

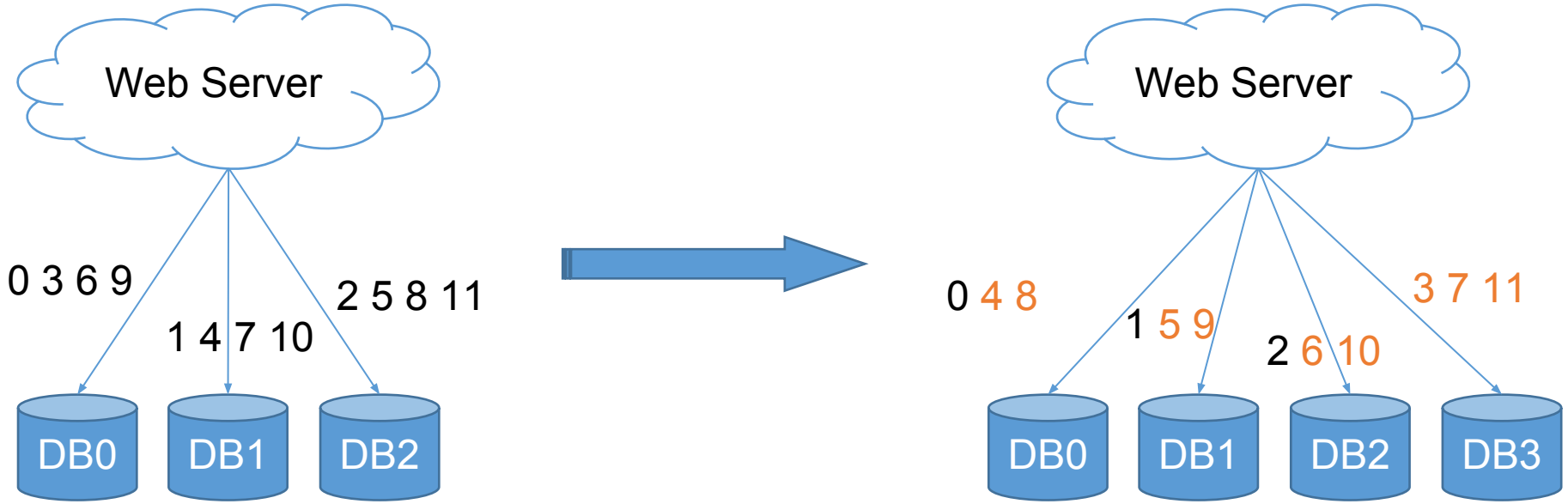
一致性哈希算法
Consistent Hashing

Horizontal Sharding 的秘密武器



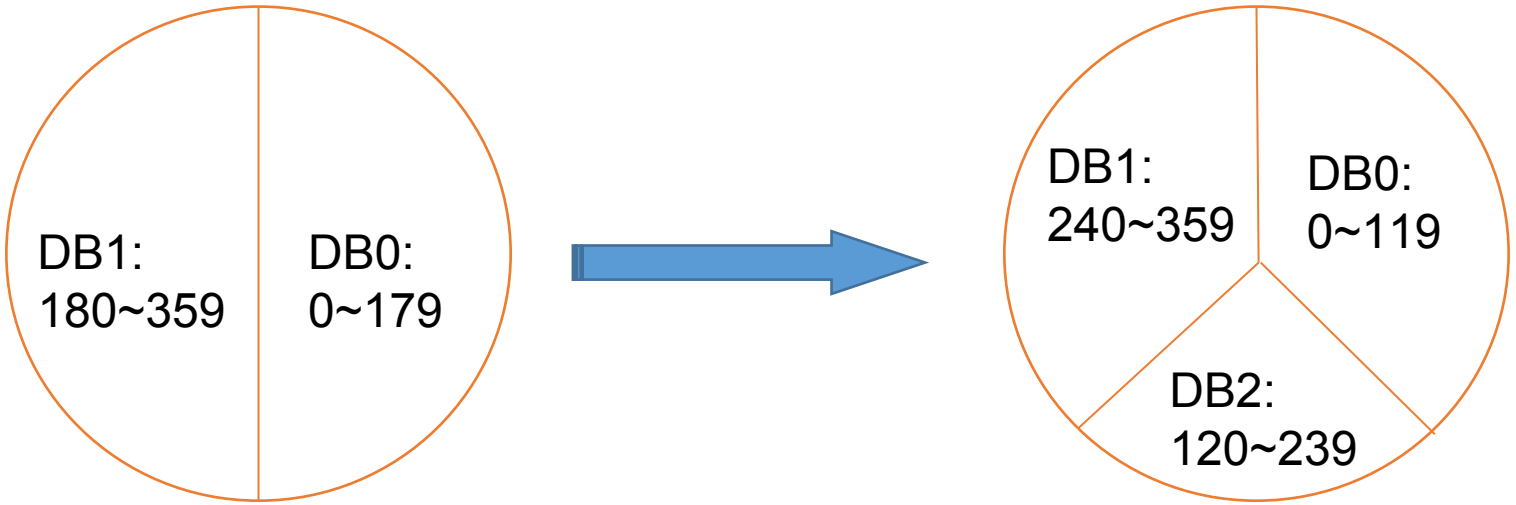
Consistent Hashing

- 我们先来说说为什么要做 “一致性”Hash
  - % n 的方法是一种最简单的 Hash 算法
  - 但是这种方法在 n 变成 n+1 的时候，每个 key % n和 % (n+1) 结果基本上都不一样
  - 所以这个 Hash 算法可以称之为：不一致 hash

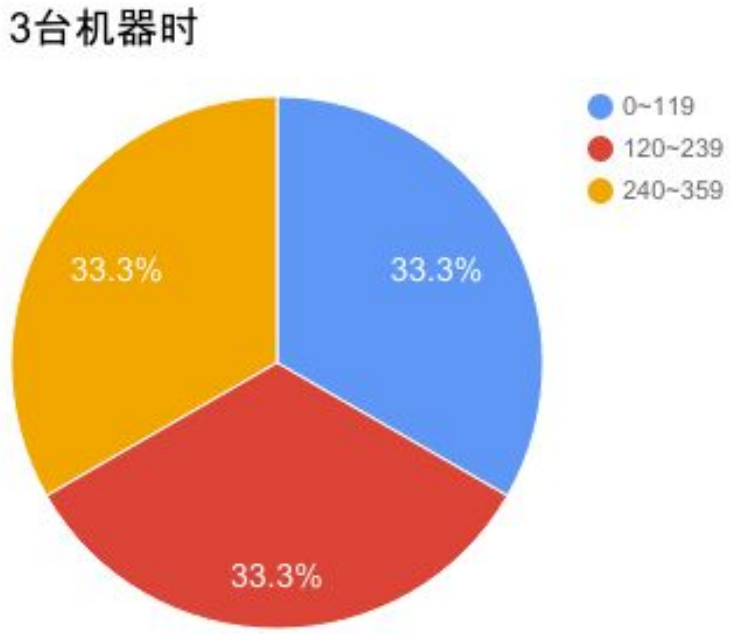


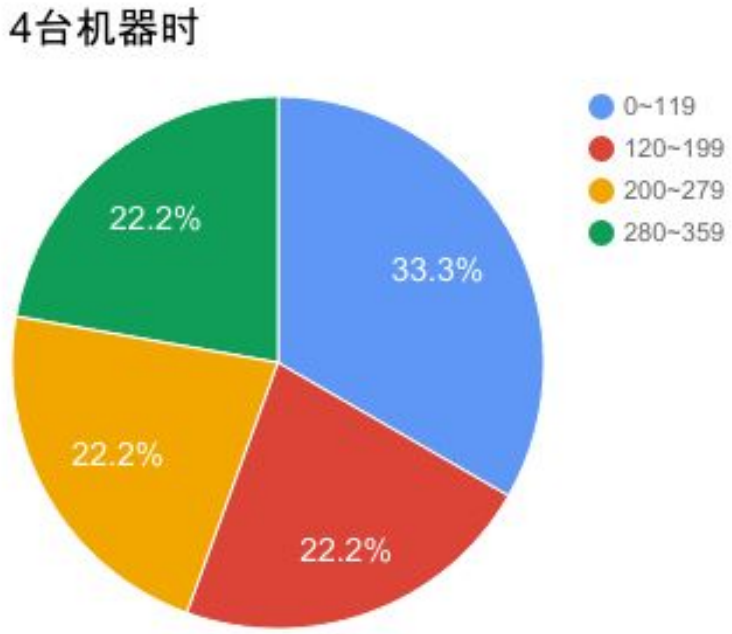
Consistent Hashing

- 一个简单的一致性Hash算法
  - 将 key 模一个很大的数，比如 360
  - 将 360 分配给 n 台机器，每个机器负责一段区间
  - 区间分配信息记录为一张表存在 Web Server 上
  - 新加一台机器的时候，在表中选择一个位置插入，匀走相邻两台机器的一部分数据
- 比如 n 从 2 变化到 3，只有 1/3 的数据移动



3台机器变4台机器的例子

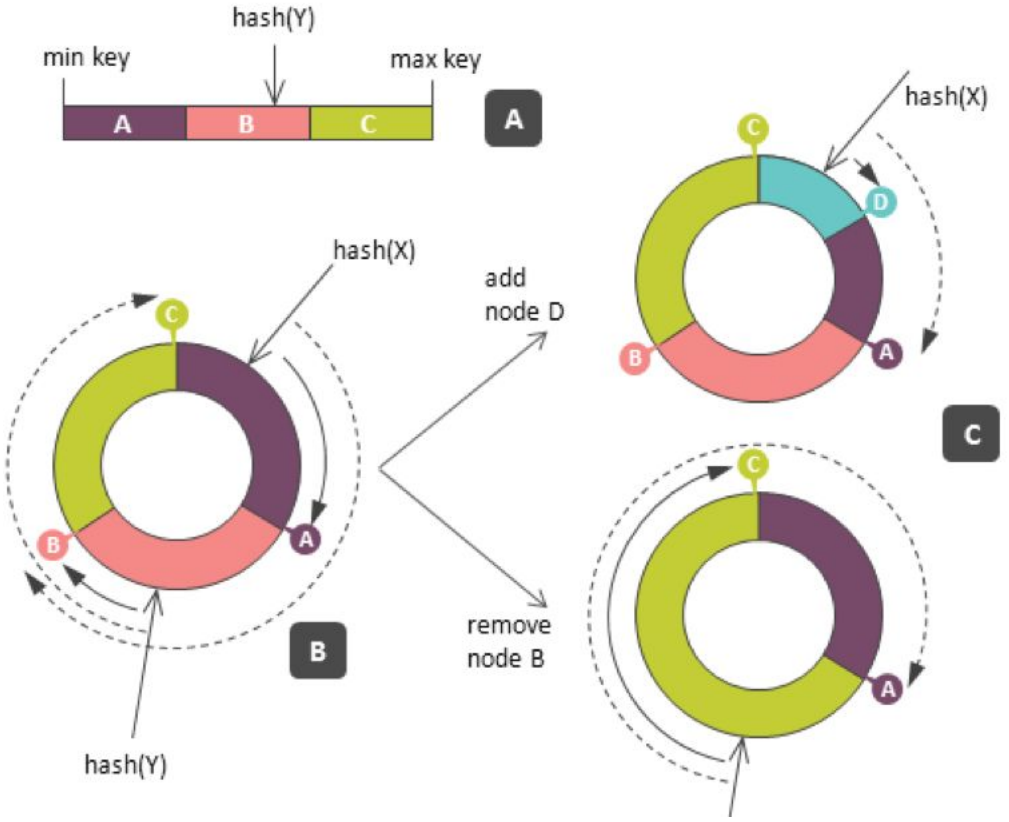




Consistent Hashing

- http://www.lintcode.com/problem/consistent-hashing/
- 每一次加入一台新机器
- 只有1台或者2台数据库的数据会被迁移
  - 因为要占据环上的一段区间

- 这是一个比较简单的 Consistent Hashing 的算法
- 提问:这种简单方法中，有什么缺陷？



缺陷1 数据分布不均匀

因为算法是“将数据最多的相邻两台机器均匀分为三台”

比如，3台机器变4台机器时，无法做到4台机器均匀分布

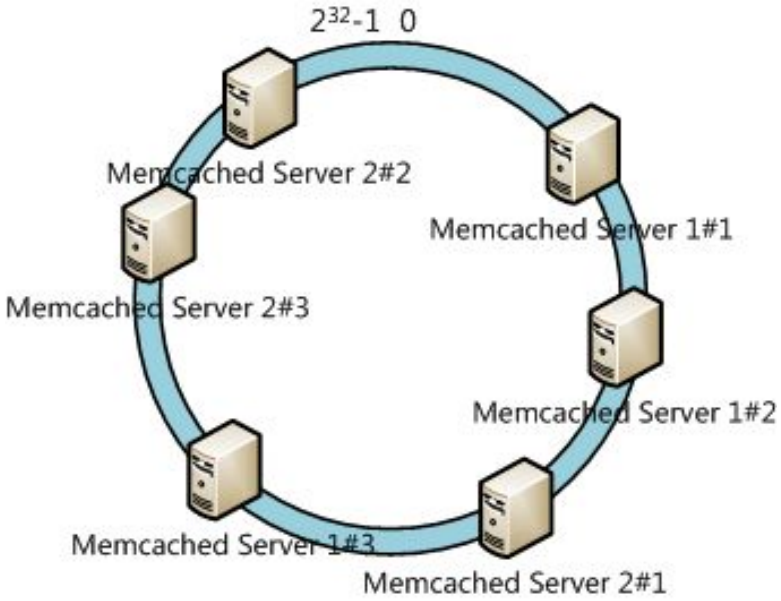
缺陷2 迁移压力大

新机器的数据只从两台老机器上获取

导致这两台老机器负载过大

Consistent Hashing —— 更实用的方法

- 将整个 Hash 区间看做环
- 这个环的大小从 0~359 变为 0~2^{64}-1
- 将机器和数据都看做环上的点
- 引入 Micro shards / Virtual nodes 的概念
  - 一台实体机器对应 1000 个 Micro shards / Virtual nodes
- 每个 virtual node 对应 Hash 环上的一个点
- 每新加入一台机器，就在环上随机撒 1000 个点作为 virtual nodes
- 需要计算某个 key 所在服务器时
  - 计算该key的hash值——得到0~2^{64}-1的一个数，对应环上一个点
  - 顺时针找到第一个virtual node
  - 该virtual node 所在机器就是该key所在的数据库服务器
- 新加入一台机器做数据迁移时
  - 1000 个 virtual nodes 各自向**顺**时针的一个 virtual node 要数据
  - 例子：http://www.jiuzhang.com/qa/2067/



Consistent Hashing

http://www.lintcode.com/problem/consistent-hashing-ii/

Replica 数据备份

问：Backup 和 Replica 有什么区别？

Backup vs Replica

Backup

- 一般是周期性的，比如每天晚上进行一次备份
- 当数据丢失的时候，通常只能恢复到之前的某个时间点
- Backup 不用作在线的数据服务，不分摊读

Replica

- 是实时的，在数据写入的时候，就会以复制品的形式存为多份
- 当数据丢失的时候，可以马上通过其他的复制品恢复
- Replica 用作在线的数据服务，分摊读

思考：既然 Replica 更牛，那么还需要 Backup么？

MySQL Replica

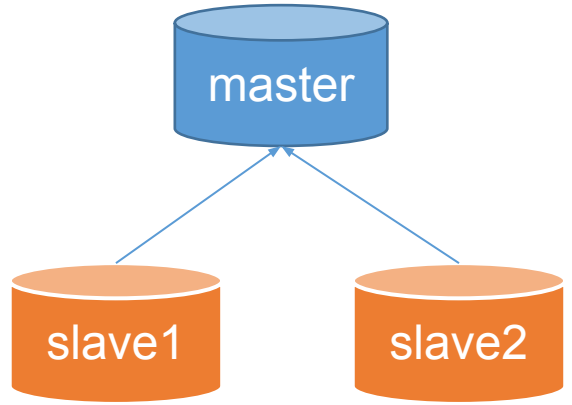
以MySQL为代表SQL型数据库，通常“自带” Master Slave 的 Replica 方法

Master 负责写，Slave 负责读

Slave 从 Master 中同步数据

Master - Slave

- 原理 Write Ahead Log
  - SQL 数据库的任何操作，都会以 Log 的形式做一份记录
  - 比如数据A在B时刻从C改到了D
  - Slave 被激活后，告诉master我在了
  - Master每次有任何操作就通知 slave 来读log
  - 因此Slave上的数据是有“延迟”的
- Master 挂了怎么办？
  - 将一台 Slave 升级 (promote) 为 Master，接受读+写
  - 可能会造成一定程度的数据丢失和不一致



NoSQL Replica

以 Cassandra 为代表的 NoSQL 数据库
通常将数据“顺时针”存储在 Consistent hashing 环上的三个 virtual nodes 中

SQL vs NoSQL in Replica

SQL

“自带” 的 Replica 方式是 Master Slave

“手动” 的 Replica 方式也可以在 Consistent Hashing 环上顺时针存三份

NoSQL

“自带” 的 Replica 方式就是 Consistent Hashing 环上顺时针存三份

“手动” 的 Replica 方式：就不需要手动了，NoSQL就是在 Sharding 和 Replica 上帮你偷懒用的！

设计短网址系统
Design Tiny URL

https://bitly.com/

https://goo.gl/

Google url shortener

Paste your long URL here:

Google

http://www.jiuzhang.com

Shorten URL

进行人机身份验证

reCAPTCHA

隐私权 - 使用条款

All goo.gl URLs and click analytics are public and can be accessed by anyone.

http://goo.gl/2KEEaJ

0 minutes ago - details

http://www.jiuzhang.com/

回顾系统设计的常见误区

流量一定巨大无比

那必须是要用NoSQL了

那必须是分布式系统了

某同学：先来个Load Balancer，后面一堆Web Server，然后memcached，最底层NoSQL，搞定！

系统设计问题的基本步骤

4S 分析法——

1. 提问：分析功能/需求/QPS/存储容量——Scenario
2. 画图：根据分析结果设计“可行解”—— Service+Storage
3. 进化：研究可能遇到的问题，优化系统 —— Scale

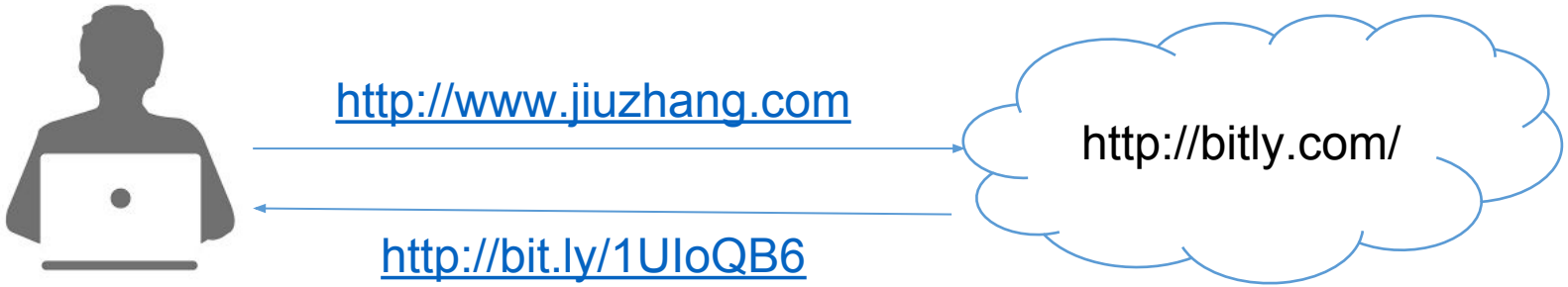
Scenario 场景分析

http://short.url

动起手来试试看

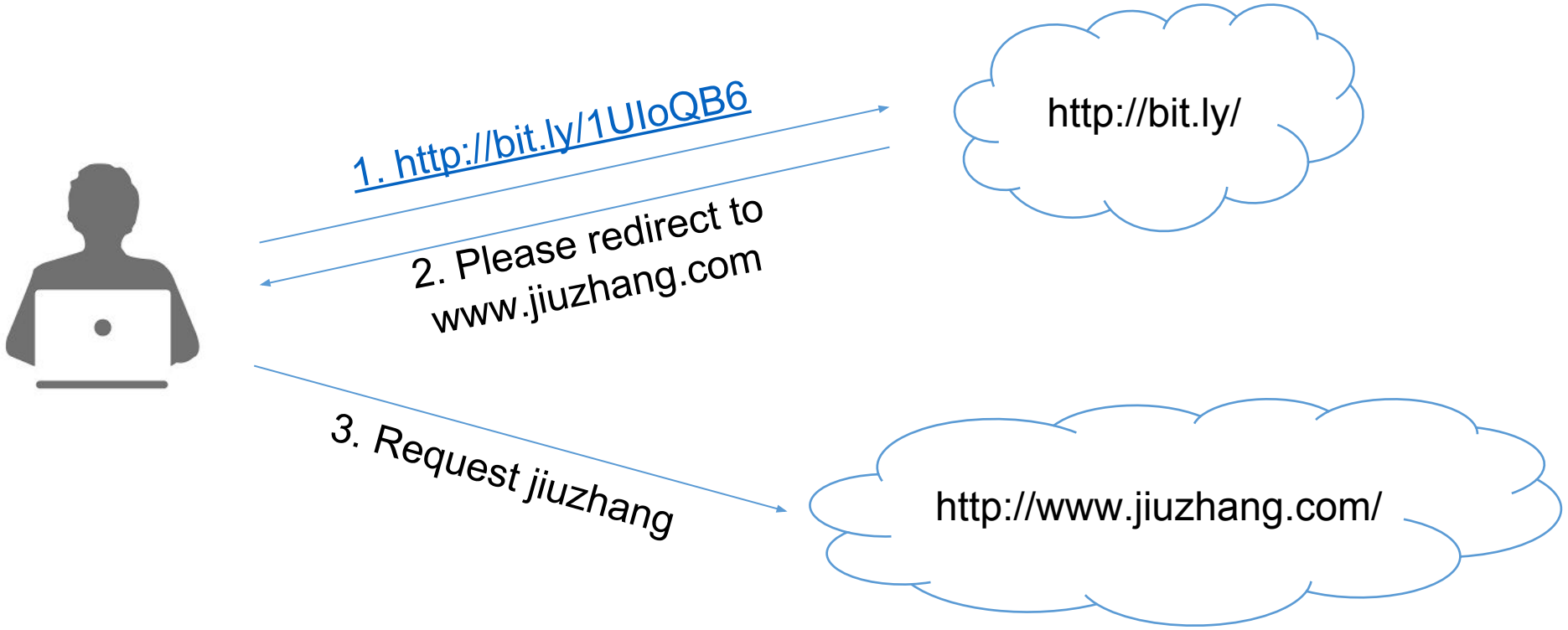
Scenario 场景 —— 我要设计啥

- 根据 Long URL 生成一个 Short URL
  - http://www.jiuzhang.com => http://bit.ly/1UIoQB6



Scenario 场景 —— 我要设计啥

- 根据 Short URL 还原 Long URL，并跳转
  - http://bit.ly/1UIoQB6 => http://www.jiuzhang.com



QPS

动起手来试试看，分析QPS和存储

Scenario 需求 —— QPS + Storage

- 1. 询问面试官微博日活跃用户
  - 约100M
- 2. 推算产生一条Tiny URL的QPS
  - 假设每个用户平均每天发 0.1 条带 URL 的微博
  - Average Write QPS = 100M * 0.1 / 86400 ~ 100
  - Peak Write QPS = 100 * 2 = 200
- 3. 推算点击一条Tiny URL的QPS
  - 假设每个用户平均点1个Tiny URL
  - Average Read QPS = 100M * 1 / 86400 ~ 1k
  - Peak Read QPS = 2k
- 4. 推算每天产生的新的 URL 所占存储
  - 100M * 0.1 ~ 10M 条
  - 每一条 URL 长度平均 100 算，一共1G
  - 1T 的硬盘可以用 3 年

2k QPS
一台 SSD支持 的MySQL完全可以搞定

Service 服务

该系统比较简单，只有一个 Service

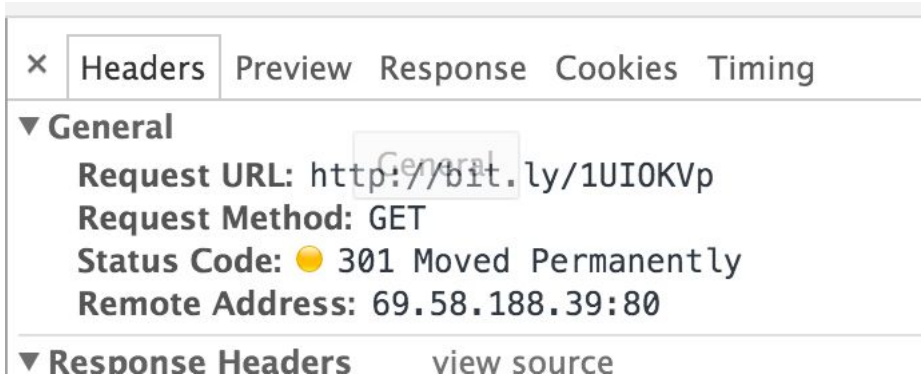
URL Service

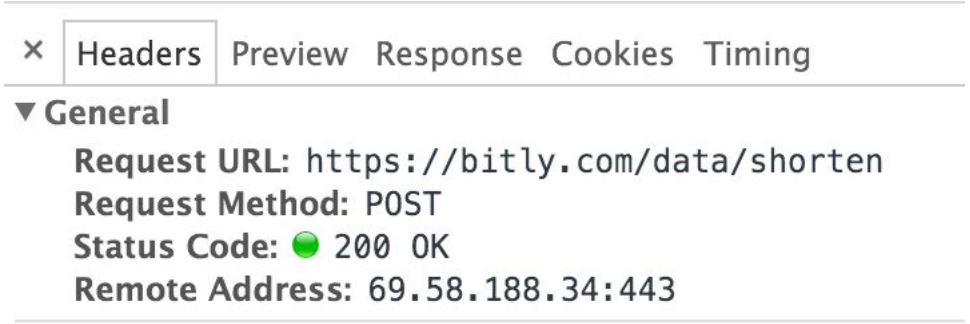
Service 服务 —— 逻辑块聚类与接口设计

- TinyUrl只有一个UrlService
  - 本身就是一个小Application
  - 无需关心其他的

- 函数设计
  - UrlService.encode(long_url)
  - UrlService.decode(short_url)

- 访问端口设计
  - GET /<short_url>
    - return a Http redirect response
  - POST /data/shorten/
    - Data = {url: http://xxxx }
    - Return short url





Tiny URL 里程碑

- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService

Storage 数据存取

数据如何存储与访问

第一步：Select 选存储结构
第二步：Schema 细化数据表

SQL or NoSQL

选什么？标准是什么？

SQL vs NoSQL —— 到底怎么选？

- 是否需要支持 Transaction？
  - NoSQL不支持Transaction
- 是否需要丰富的 SQL Query？
  - NoSQL的SQL Query不是太丰富
  - 也有一些NoSQL的数据库提供简单的SQL Query支持
- 是否想偷懒？
  - 大多数 Web Framework 与 SQL 数据库兼容得很好
  - 用SQL比用NoSQL少写很多代码
- 是否需要Sequential ID？
  - SQL 为你提供了 auto-increment 的 Sequential ID
  - 也就是1,2,3,4,5 …
  - NoSQL的ID并不是 Sequential 的

SQL vs NoSQL —— 到底怎么选？

- 对QPS的要求有多高？
  - NoSQL 的性能更高
- 对Scalability的要求有多高？
  - SQL 需要码农自己写代码来 Scale
    - 还记得Db那节课中怎么做 Sharding，Replica 的么？
  - NoSQL 这些都帮你做了

所以Tiny URL用什么比较合适？

Storage 数据如何存储与访问

- 是否需要支持 Transaction？——不需要。NoSQL +1
- 是否需要丰富的 SQL Query？——不需要。NoSQL +1
- 是否想偷懒？——Tiny URL 需要写的代码并不复杂。NoSQL+1
- 对QPS的要求有多高？—— 经计算，2k QPS并不高，而且2k读可以用Cache，写很少。SQL +1
- 对Scalability的要求有多高？—— 存储和QPS要求都不高，单机都可以搞定。SQL+1
- 是否需要Sequential ID？—— 取决于你的算法是什么

算法是什么？

如何将 Long Url 转换为一个 6位的 Short Url？

http://www.lintcode.com/problem/tiny-url/

算法1 使用哈希函数 Hash Function（不可行）

- 比如取 Long Url 的 MD5 的最后 6 位
  - 只是打个比方，这个方法肯定是有问题的啦
- 优点：快
- 缺点：难以设计一个没有冲突的哈希算法

算法2 随机生成ShortURL + 数据库去重

- 随机一个 6 位的 ShortURL，如果没有被用过，就绑定到该 LongURL
- 伪代码如下：

```
public String longToShort(String url) {
    while (true) {
        String shortURL = randomShortURL();
        if (!database.filter(shortURL=shortURL).exists() {
            database.create(shortURL=shortURL, longURL=url);
            return shortURL;
        }
    }
}
```

- 优点：实现简单
- 缺点：生成短网址的速度随着短网址越来越多变得越来越慢

算法3 进制转换 Base62

- Base62
  - 将 6 位的short url看做一个62进制数(0-9, a-z, A-Z)
  - 每个short url 对应到一个整数
  - 该整数对应数据库表的Primary Key —— Sequential ID
- 6 位可以表示的不同 URL 有多少？
  - 5 位 = 62^5 = 0.9B = 9 亿
  - 6 位 = 62^6 = 57 B = 570 亿
  - 7 位 = 62^7 = 3.5 T = 35000 亿
-
- 优点：效率高
- 缺点：依赖于全局的自增ID

```
int shortURLtoID(String shortURL) {
    int id = 0;
    for (int i = 0; i < shortURL.length(); ++i) {
        id = id * 62 + toBase62(shortURL.charAt(i));
    }
    return id;
}

String idToShortURL(int id) {
    String chars = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String short_url = "";
    while (id > 0) {
        short_url = chars.charAt(id % 62) + short_url;
        id = id / 62;
    }
    while (short_url.length() < 6) {
        short_url = "0" + short_url;
    }
    return short_url;
}
```

随机生成 vs 进制转换

幸福二选一

基于随机生成的方法

需要根据 Long 查询 Short，也需要根据 Short 查询 Long。

如果选择用 SQL 型数据库，表结构如下：

shortKey	longUrl
a0B4Lb	http://www.jiuzhang.com/
Df523P	http://www.lintcode.com/
dao80F	http://www.google.com/
QFD8oq	http://www.facebook.com/

并且需要对 shortKey 和 longURL 分别建**索引(index)**。

- 什么是索引？http://t.cn/R6xgLd8
- 索引的原理？http://t.cn/R6xg4aj

基于随机生成的方法

也可以选用 NoSQL 数据库，但是需要建立两张表（大多数NoSQL数据库不支持二级索引）

以 Cassandra 为例子

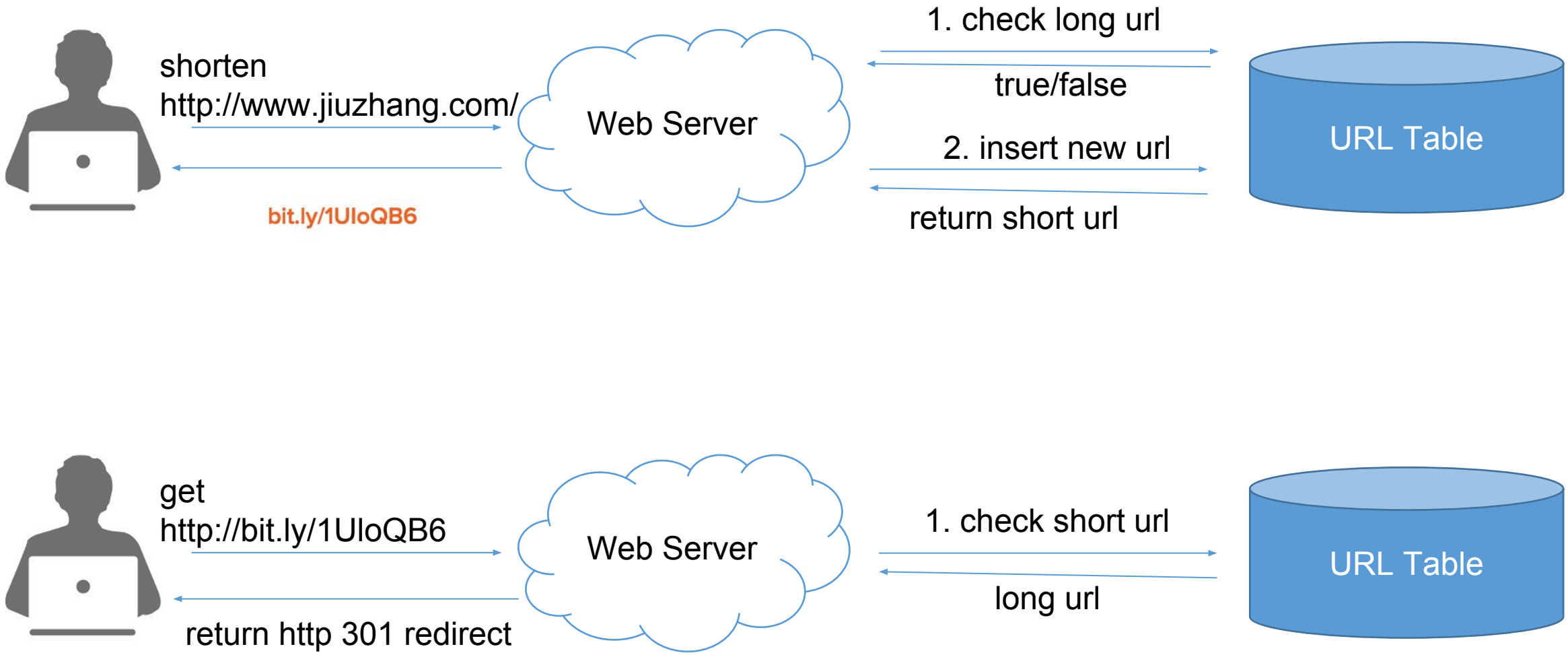
第一张表：根据 Long 查询 Short

row_key=longURL, column_key=ShortURL, value=null or timestamp

第二张表：根据 Short 查询 Long

row_key=shortURL, column_key=LongURL, value=null or timestamp

基于随机生成方法的 Work Solution



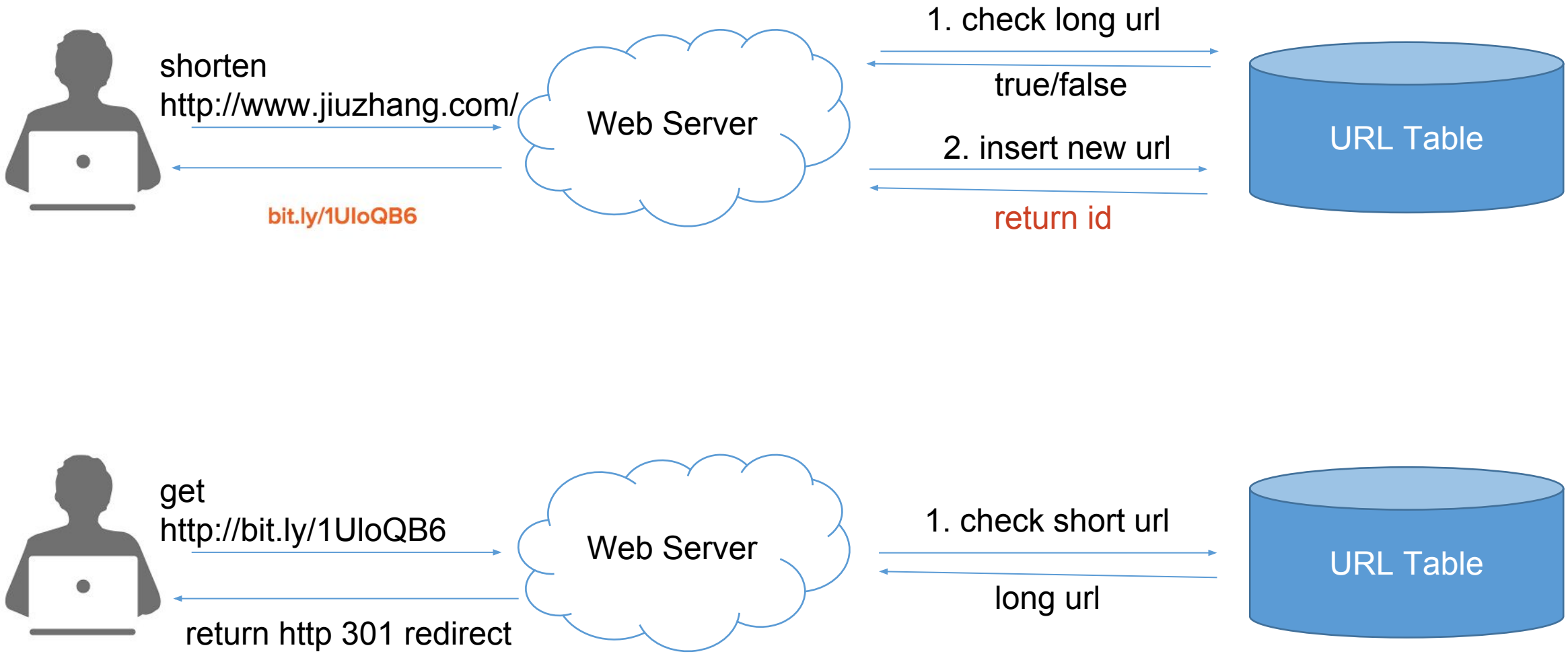
基于进制转换的方法

因为需要用到自增ID(Sequential ID)，因此只能选择使用 SQL 型数据库。

表单结构如下，shortURL 可以不存储在表单里，因为可以根据 id 来进行换算

id	longUrl (index=true)
1	http://www.jiuzhang.com/
2	http://www.lintcode.com/
3	http://www.google.com/
4	http://www.facebook.com/

基于进制转换方法的Work Solution



Tiny URL 里程碑

- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution

先休息5分钟，然后
下半节的内容，你基本Google不到

要开始教大家弹指神通了

期中调查问卷 http://form.mikecrm.com/vmUTqK

You can you up a

你行你上啊！

Scale 进化

Tiny URL 有什么可以优化的地方？

Interviewer: How to reduce response time?

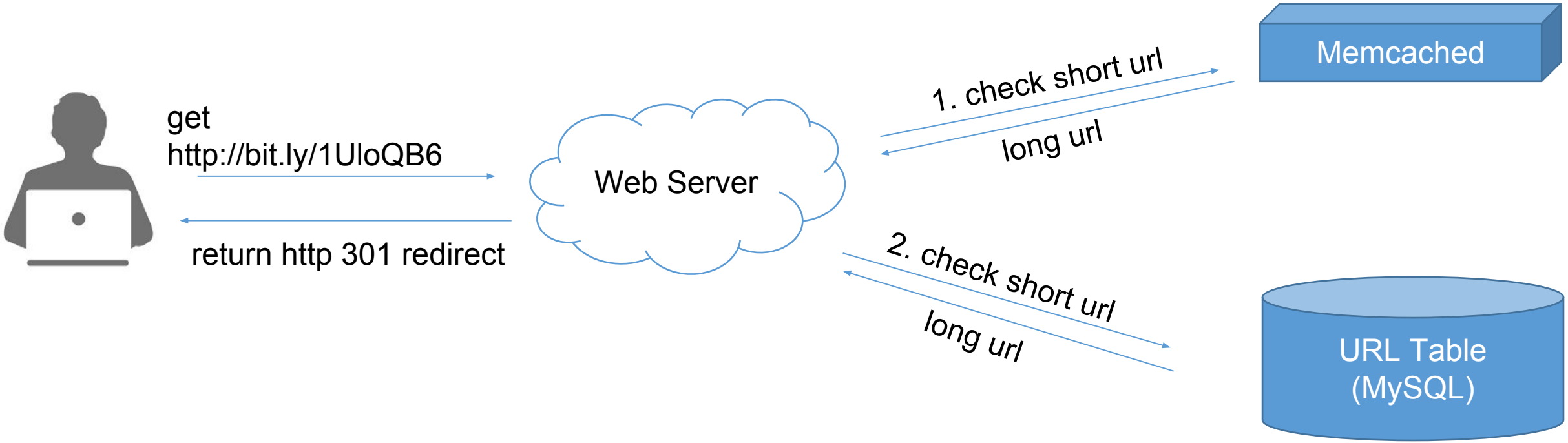
如何提高响应速度？

说说看有哪些地方可以加速？

提高读的速度还是写的速度？

Scale – 如何提速

- 利用缓存提速(Cache Aside)
- 缓存里需要存两类数据:
  - long to short(生成新 short url 时需要)
  - short to long(查询 short url 时需要)
- 小练习:自己画一下生成新URL时的流程图



Tiny URL 里程碑

- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution
- 提高Web服务器与数据服务器之间的访问效率：利用缓存提高读请求的效率

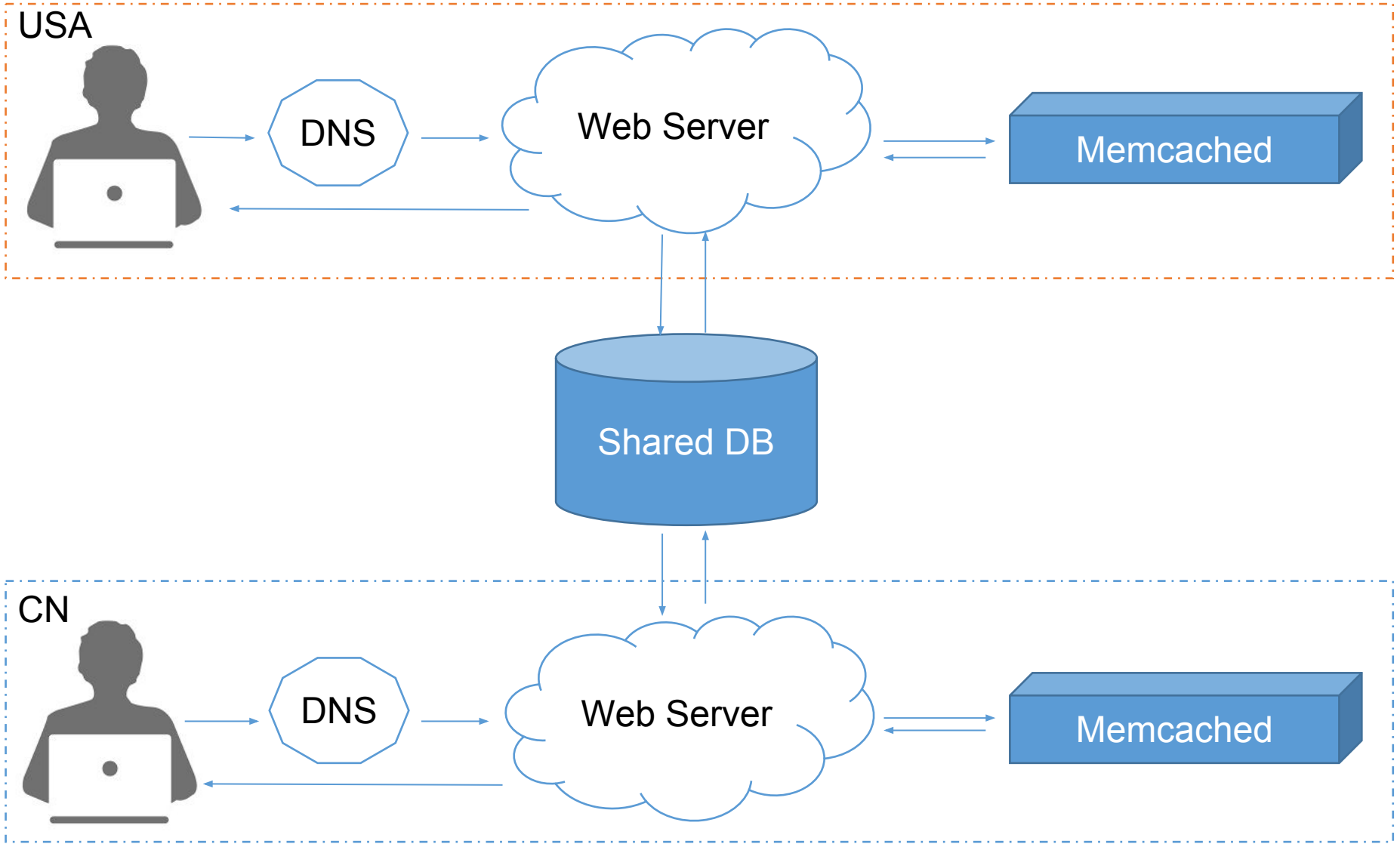
Scale – 如何提速

- 利用地理位置信息提速

- 优化服务器访问速度
  - 不同的地区，使用不同的 Web 服务器
  - 通过DNS解析不同地区的用户到不同的服务器

- 优化数据访问速度
  - 使用Centralized MySQL+Distributed Memcached
  - 一个MySQL配多个Memcached，Memcached跨地区分布

Scale – 如何提速



Tiny URL 里程碑

- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution
- 提高Web服务器与数据服务器之间的访问效率：利用缓存提高读请求的效率
- 提高用户与服务器之间的访问效率：解决了中国用户访问美国服务器慢的问题

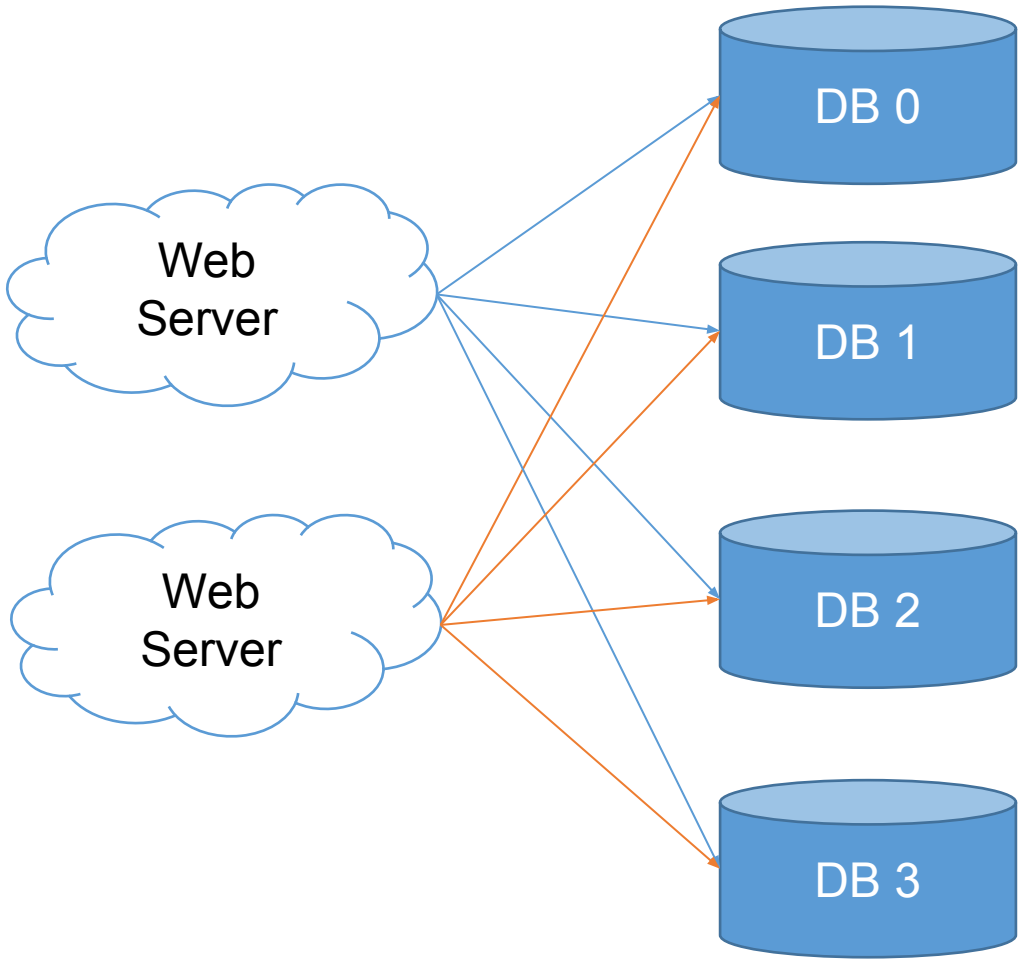
* Interviewer: How to scale?

假如我们一开始估算错了，一台MySQL搞不定了

Scale —— 如何扩展？

- 什么时候需要多台数据库服务器？
  - Cache 资源不够
  - 写操作越来越多
  - 越来越多的请求无法通过 Cache 满足
- 增加多台数据库服务器可以优化什么？
  - 解决“存不下”的问题 —— Storage的角度
  - 解决“忙不过”的问题 —— QPS的角度
  - Tiny URL 主要是什么问题？

如何将数据分配到多台机器？

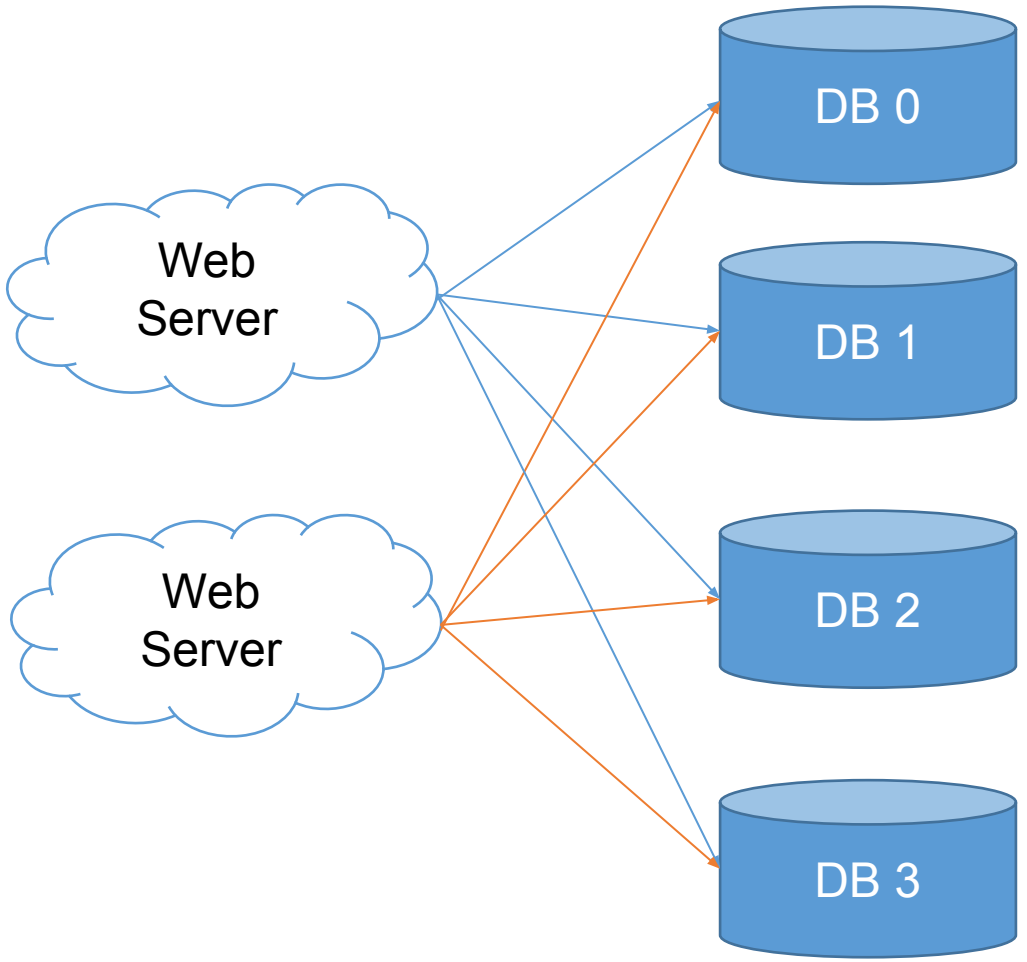


Scale —— 如何扩展？

- Vertical Sharding —— 将多张数据表分别分配给多台机器
  - Tiny URL 并不适用

- Horizontal Sharding
  - 用什么做Sharding Key?
  - 如果用 ID，如何查询 Long Url?
  - 如果用Long Url，如何查询 ID？

用什么做 Sharding Key?



Scale —— 如何扩展？

- 用 Long URL 做 shard key
  - 查询的时候，只能广播给N台数据库查询
  - 并不解决降低每台机器QPS的问题
- 用 ID 做 shard key
  - 按照 ID % N（N为数据服务器个数），来分配存储
  - Short url to long url
    - 将 short url 转换为 ID
    - 根据 ID 找到数据库
    - 在该数据库中查询到 long url
  - Long url to short url
    - 先查询：广播给 N 台数据库，查询是否存在
      - 看起来有点耗，不过也是可行的，因为数据库服务器不会太多
    - 再插入：如果不存在的话，获得下一个自增 ID 的值，插入对应数据库

Scale —— 全局自增ID

- 如何获得在N台服务器中全局共享的一个自增ID是一个难点
- 一种解决办法是，专门用一台数据库来做自增ID服务
  - 该数据库不存储真实数据，也不负责其他查询
  - 为了避免单点失效(Single Point Failure) 可能需要多台数据库
- 另外一种解决办法是用 Zookeeper

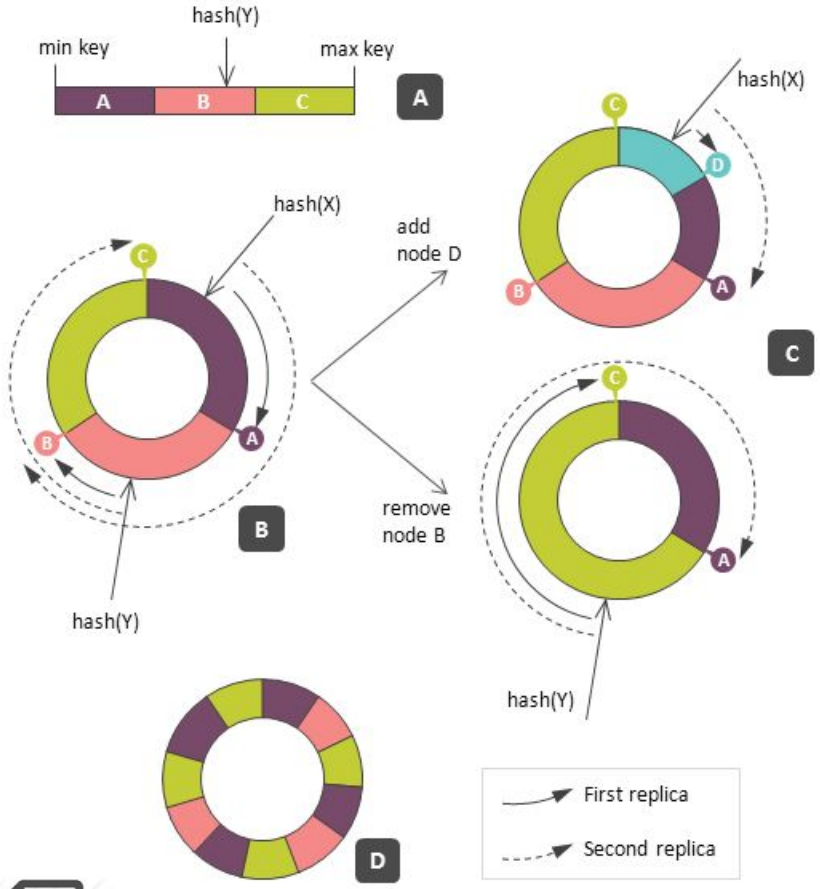
Read more:
http://bit.ly/1RCyPsy

- 使用全局自增ID的方法并不是解决 Tiny URL 的最佳方法
  - 故不展开讨论全局自增ID的细节

卧槽，那到底怎么做？

Scale —— 扩展SHORT KEY

- 如果最开始，short key 为6位，下面为short key增加 1 位前置位
  - AB1234 → **0**AB1234
  - 还有一种做法是把第1位单独留出来做 sharding key，总共还是6位
- 该前置位的值由 Hash(long_url) % 62 得到
- 该前置位则为 sharding key
  - Consistent Hashing
    - 相关练习 http://www.lintcode.com/en/problem/consistent-hashing/
  - 将环分为62个区间
  - 每台机器在环上负责一段区间

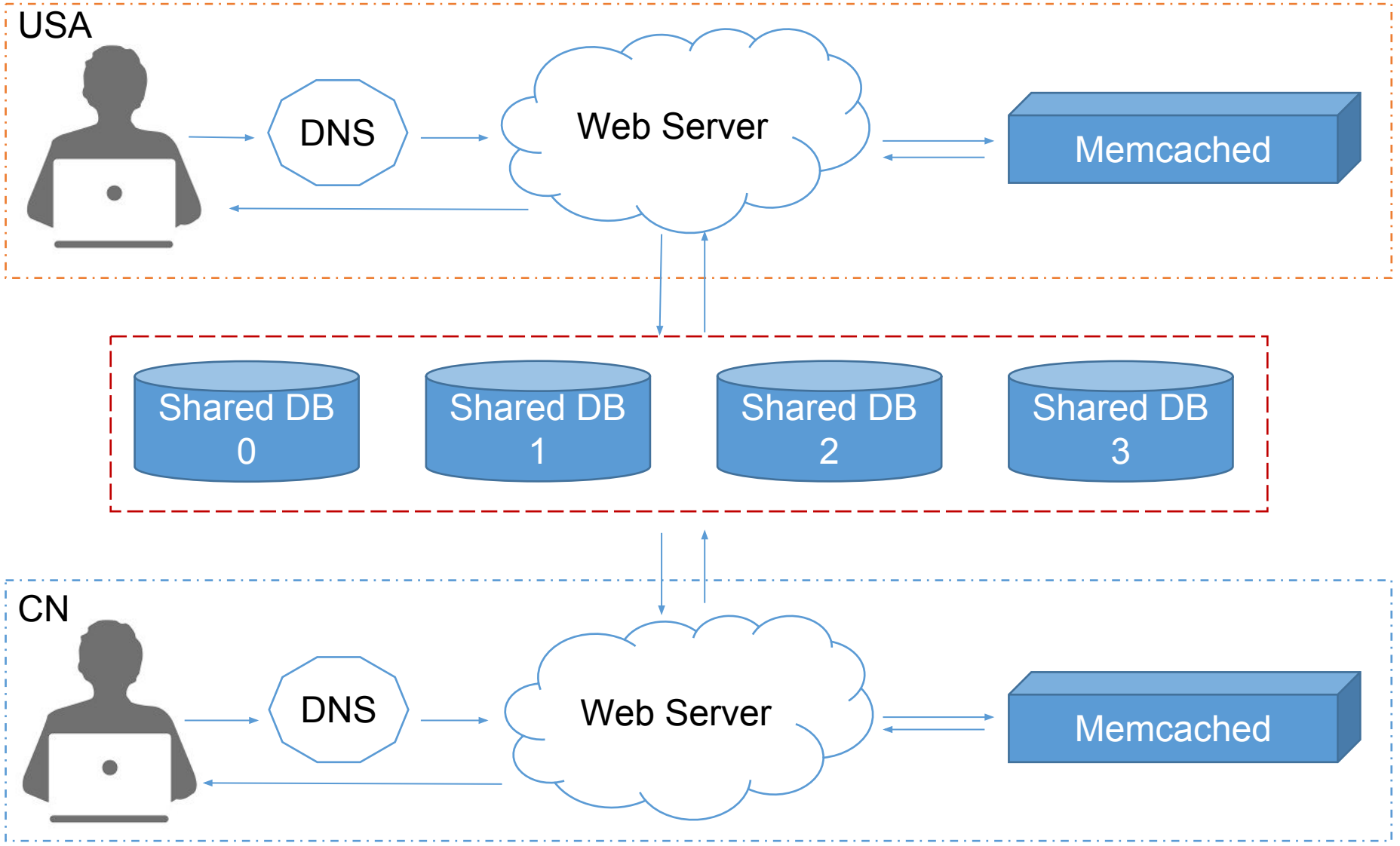


- 这样我们就可以同时通过 short_url 和 long_url 得到 Sharding Key
  - 无需广播
  - 无论是short2long还是long2short
  - 都可以直接找到数据所在服务器

Read more: http://bit.ly/1XU9uZH

Read more: http://bit.ly/1KhqPEr

Scale —— 目前的架构



Consistent Hashing 的 Mapping 表存于每台 Web Server 上，hard code 在代码里

Tiny URL 里程碑

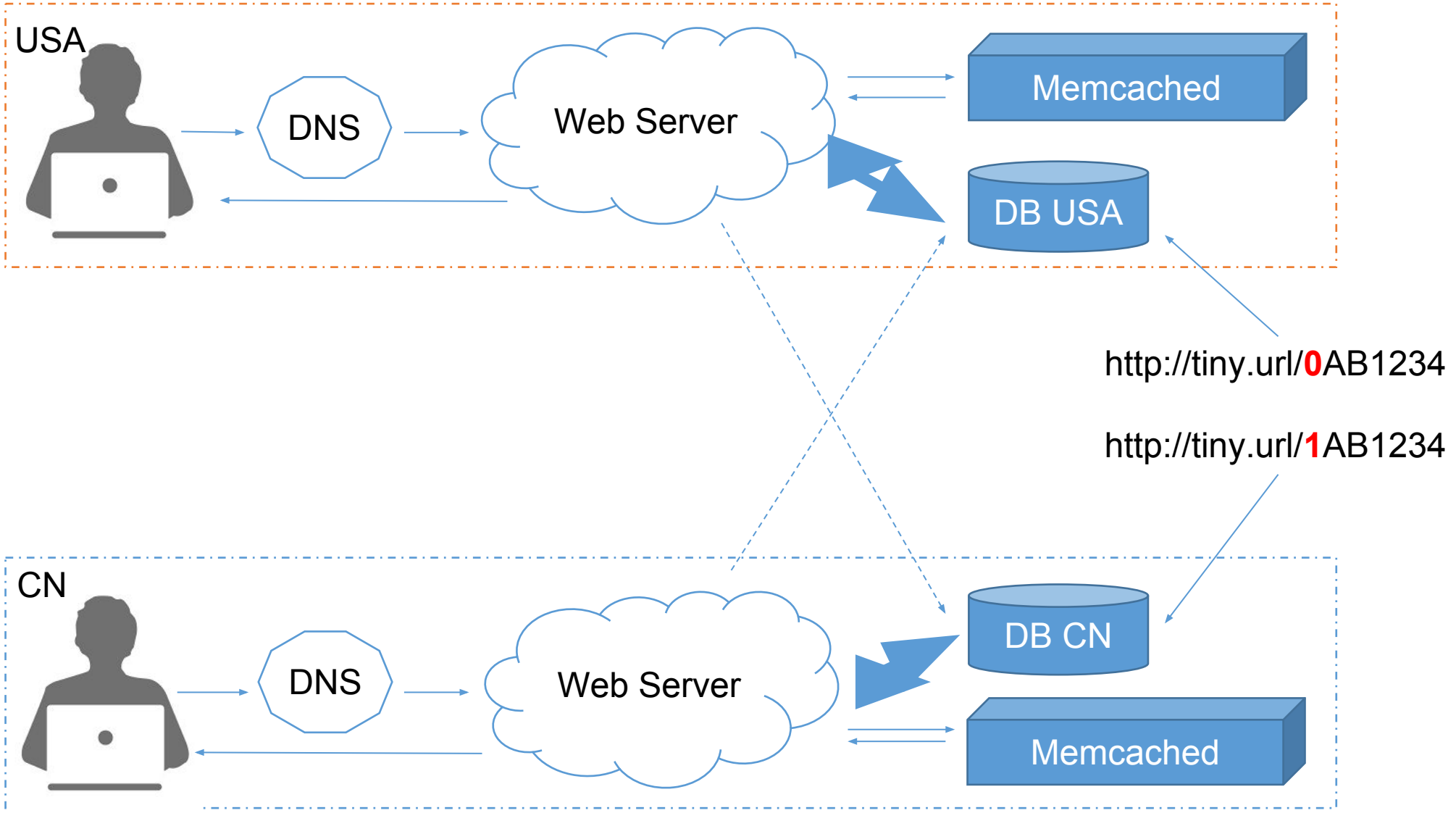
- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution
- 提高Web服务器与数据服务器之间的访问效率：利用缓存提高读请求的效率
- 提高用户与服务器之间的访问效率：解决了中国用户访问美国服务器慢的问题
- 解决流量继续增大一台数据库服务器无法满足的问题：将数据分配到多台服务器

* Interviewer: 还有可以优化的么？

Scale —— Multi Region 的进一步优化

- 上图的架构中，还存在优化空间的地方是 ——
  - 网站服务器 (Web Server) 与 数据库服务器 (Database) 之间的通信
  - 中心化的服务器集群（Centralized DB set）与 跨地域的 Web Server 之间通信较慢
    - 比如中国的服务器需要访问美国的数据库
- 那么何不让中国的服务器访问中国的数据库？
  - 如果数据是重复写到中国的数据库，那么如何解决一致性问题？
    - 很难解决
- 想一想用户习惯
  - 中国的用户访问时，会被DNS分配中国的服务器
  - 中国的用户访问的网站一般都是中国的网站
  - 所以我们可以按照网站的**地域信息**进行 Sharding
    - 如何获得网站的地域信息？只需要将用户比较常访问的网站弄一张表就好了
  - 中国的用户访问美国的网站怎么办？
    - 那就让中国的服务器访问美国的数据好了，反正也不会慢多少
    - 中国访问中国是主流需求，优化系统就是要优化主要的需求

Scale —— 最终的架构



Tiny URL 里程碑

- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution
- 提高Web服务器与数据服务器之间的访问效率：利用缓存提高读请求的效率
- 提高用户与服务器之间的访问效率：解决了中国用户访问美国服务器慢的问题
- 提高QPS，将数据分配到多台服务器：解决流量继续增大一台数据库服务器无法满足的问题
- 提高中国的Web服务器与美国的数据库服务器通信较慢的问题：按照网站地域信息进行Sharding

* Interviewer: How to provide custom url?

http://tiny.url/google/ => http://www.google.com

http://tiny.url/systemdesign/ => http://www.jiuzhang.com/course/2/

Scale —— 自定义的短链接

- 新建一张表存储自定义URL
  - CustomURLTable
- 查询长链接
  - 先查询CustomURLTable
  - 再查询URLTable
- 根据长链接创建普通短链接
  - 先查询CustomURLTable是否存在
  - 再在URLTable中查询和插入
- 创新自定义短链接
  - 查询是否已经在URLTable中存在
  - 再在CustomURLTable中查询和插入

custom_url	long_url (index=true)
gg	http://www.google.com/
fb	http://www.facebook.com/
jz	http://www.jiuzhang.com/

- 错误的想法：
  - 在URLTable中加一个column
  - 大部分数据这一项都会是空

- 课后练习：http://www.lintcode.com/problem/tiny-url-ii/

Tiny URL 设计总结

- 场景分析：要做什么功能
- 需求分析：QPS和Storage

Scenario: 各种问面试官问题，搞清楚要干嘛

- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution

Service + Storage:
根据问到的内容进行分析
得到一个基本可以work的方案

- 提高Web服务器与数据服务器之间的访问效率
  - 利用缓存提高读请求的效率
- 提高用户与服务器之间的访问效率
  - 解决了中国用户访问美国服务器慢的问题
- 提高QPS，将数据分配到多台服务器
  - 解决流量继续增大一台数据库服务器无法满足的问题
- 提高中国的Web服务器与美国的数据库服务器通信较慢的问题
  - 按照网站地域信息进行Sharding
- 提供 Custom URL 的功能

Scale

follow up

重要的事情说N遍

系统设计没有标准答案，言之有理即可

设计一个可行解比抛出Fancy的概念更有用

课后作业

思考题：

- 请基于随机生成的算法，设计一整套系统
- 假如同一个 Long URL 可以对应到不用的 Short URL，请根据这个特性优化你的系统架构。
- 是否需要长久保存 ShortURL？好处是什么，坏处是什么？
- 如果一个 short URL 被疯狂分享和点击，会出现什么情况？以及如何避免这种情况带来的麻烦？

编程题：

www.lintcode.com/problem/tiny-url

www.lintcode.com/problem/tiny-url-ii

www.lintcode.com/problem/consistent-hashing/

今天我们学会了什么？

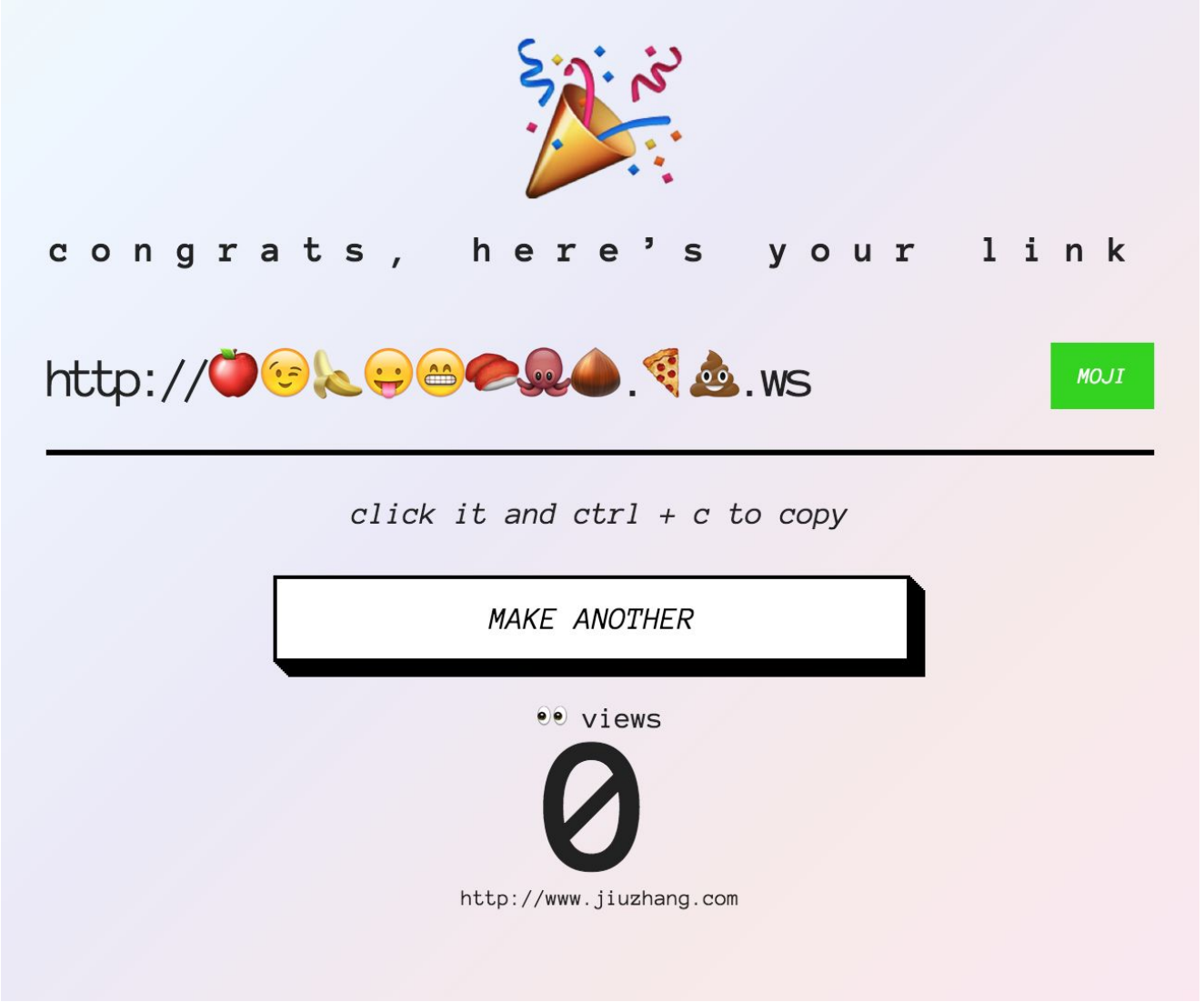
- 两个系统设计的 General Questions
  - How to sharding?
  - How to replica?
- 一个超高频的系统设计面试题
  - Design Tiny URL
- 从题目之中我们学会的：
  - SQL 和 NoSQL 的选择标准
  - 读比较多的话，可以用 Memcached 来优化
  - 写比较多的话，可以拆分数据库
    - Vertical Sharding / Horizontal Sharding
    - Consistent Hashing
  - Multi Region
    - 知道了不同区域之间的访问速度很慢
    - 知道了用户跨区访问比服务器跨区访问更慢

附录:扩展阅读

- Tiny URL 相关（只作为参考，有一些答案并不完全正确，以课堂讲述内容为准）
  - 知乎 http://bit.ly/22FHP5o
  - The System Design Process http://bit.ly/1B6HOEc
- 用Django搭建一个网站 http://bit.ly/21LApIb
  - 选做:搭建一个tiny url的网站
- Global Sequential ID
  - http://bit.ly/1RCyPsy
- Consistent Hashing
  - http://bit.ly/1XU9uZH
  - http://bit.ly/1KhqPEr

Linkmoji —— emoji url

- http://www.xn--vi8hiv.ws



附录

以下内容供大家自学

为什么了解网站系统如此重要？

System Design 几乎都是 Backend Design

Backend Design 几乎都是 Web Backend Design

面试官：当你访问www.google.com的时候发生了什么？

网站系统的基本概念

- DNS
- HTTP
- Domain
- IP Address
- URL
- Web Server（硬件）
- HTTP Server（软件）
- Web Application（软件）

当你访问 www.google.com 的时候发生了什么

- 你在浏览器输入 www.google.com
- 你首先访问的是离你最近的 DNS 服务器
  - Domain Name Service
  - DNS 服务器记录了 www.google.com 这个域名的 IP 地址是什么
- 你的浏览器然后向该 IP 发送 http/https 请求
  - 每台服务器/计算机联网都需要一个 IP 地址
  - 通过 IP 地址就能找到该 服务器/计算机

IP 地址

```
PING www.google.com (173.194.202.104) 56(84) bytes of data.
64 bytes from pf-in-f104.1e100.net (173.194.202.104): icmp_seq=1 ttl=63 time=32.9 ms
64 bytes from pf-in-f104.1e100.net (173.194.202.104): icmp_seq=2 ttl=63 time=33.9 ms
64 bytes from pf-in-f104.1e100.net (173.194.202.104): icmp_seq=3 ttl=63 time=39.3 ms
64 bytes from pf-in-f104.1e100.net (173.194.202.104): icmp_seq=4 ttl=63 time=34.8 ms
```

当你访问 www.google.com 的时候发生了什么

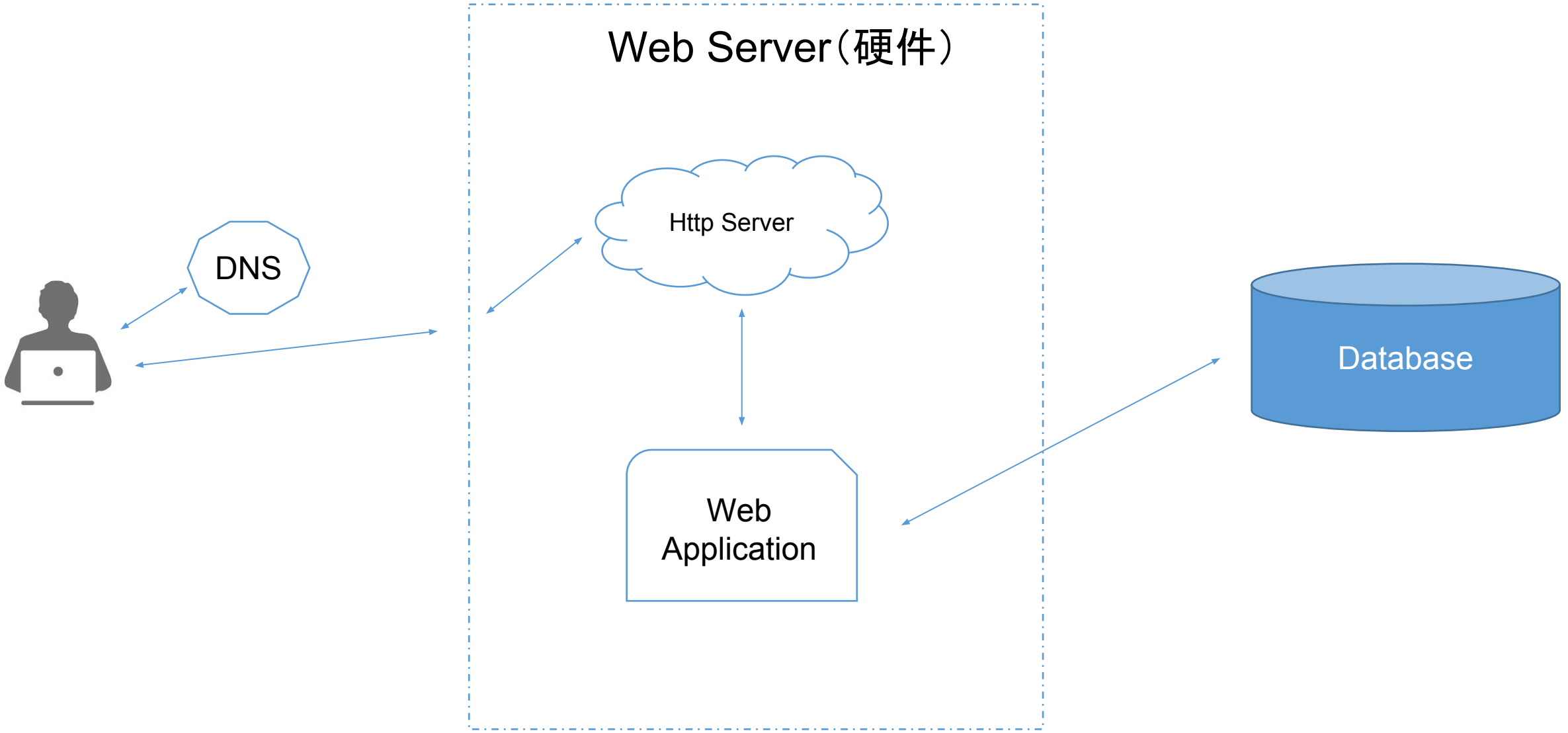
- 服务器(Web Server)收到请求，将请求递交给正在 80 端口监听的HTTP Server
- 比较常用的 HTTP Server 有 Apache, Unicorn, Gunicorn, Uwsgi
- HTTP Server 将请求转发给 Web Application
  - 最火的三大Web Application Framework: Django, Ruby on Rails, NodeJS
- Web Application 处理请求
  - 根据当前路径 “/”找到对应的逻辑处理模块
  - 根据用户请求参数(GET+POST)决定如何获取/存放数据
  - 从数据存储服务（数据库或者文件系统）中读取数据
  - 组织数据成一张 html 网页作为返回结果
- 浏览器得到结果，展示给用户

当你访问 www.google.com 的时候发生了什么

动手用Django搭建一个网站（约1天）

Read more: http://bit.ly/21LApIb

简单的网站系统架构原理图



复杂的网站系统架构原理图