

网站系统设计 Web System

课程版本 v4.1 本节主讲人 东邪

不允许录像与传播录像，否则将追究法律责任和经济赔偿



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>

- 通过一道面试题了解网站相关的基本概念
 - What happened if you visit www.google.com?
- 设计一个短网址系统 Design Tiny Url
 - SNAKE 分析法
 - No Hire / Weak Hire / Hire / Strong Hire
- 访问限制 Design Rate limiter
- 访问统计 Design Data dog

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？



- 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



URL

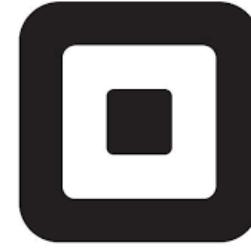
域名 Domain

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
```

- 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

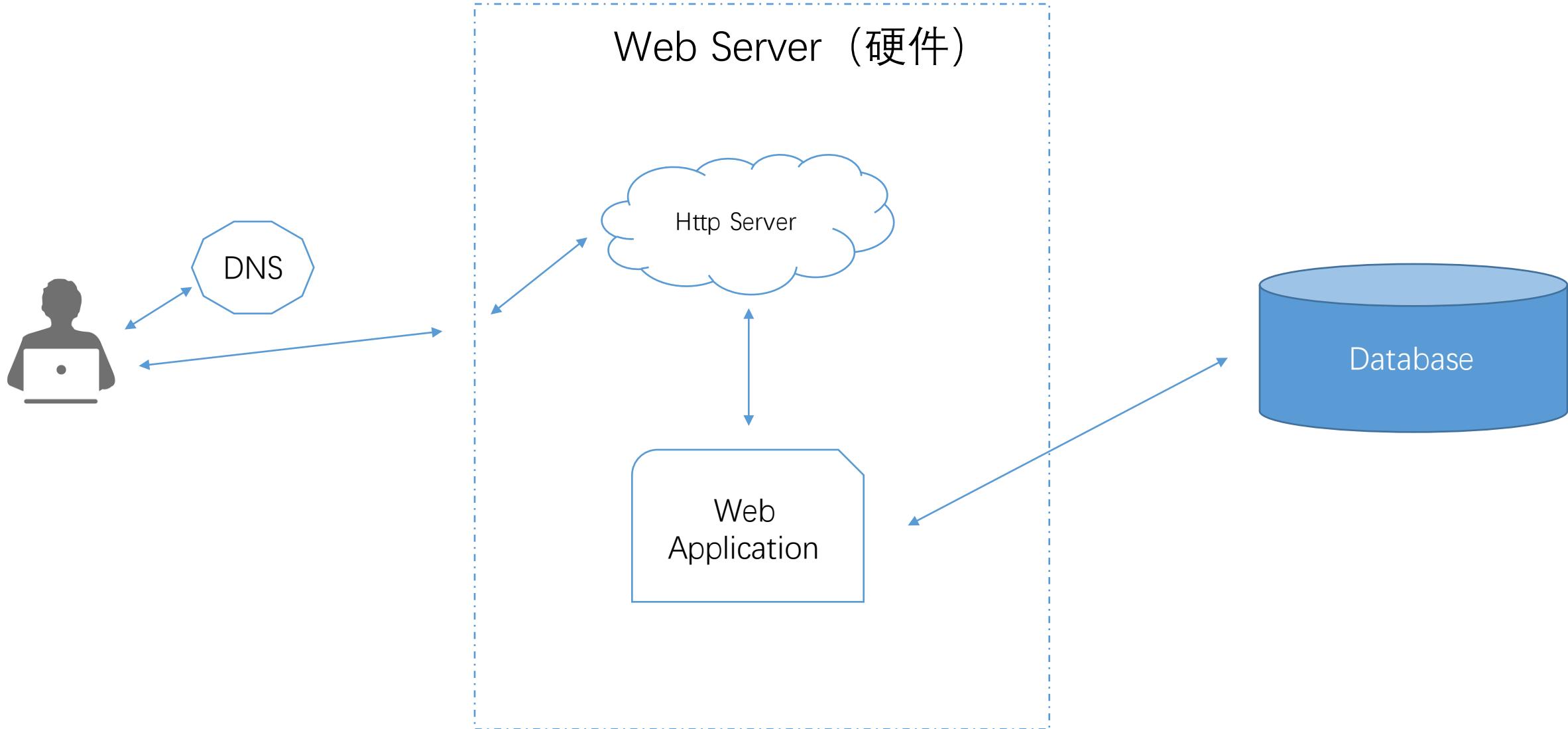


动手用Django搭建一个网站（约1天）

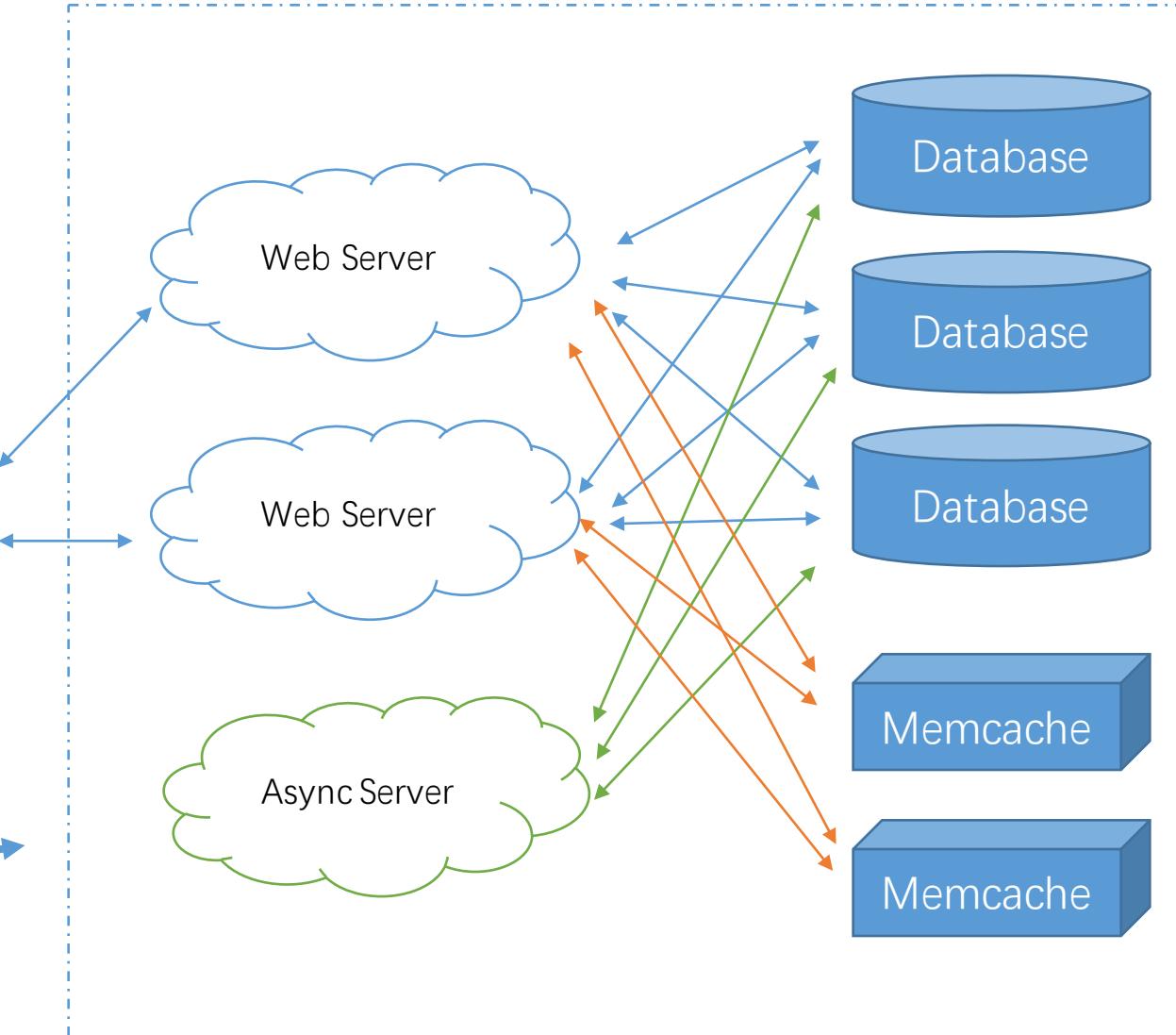
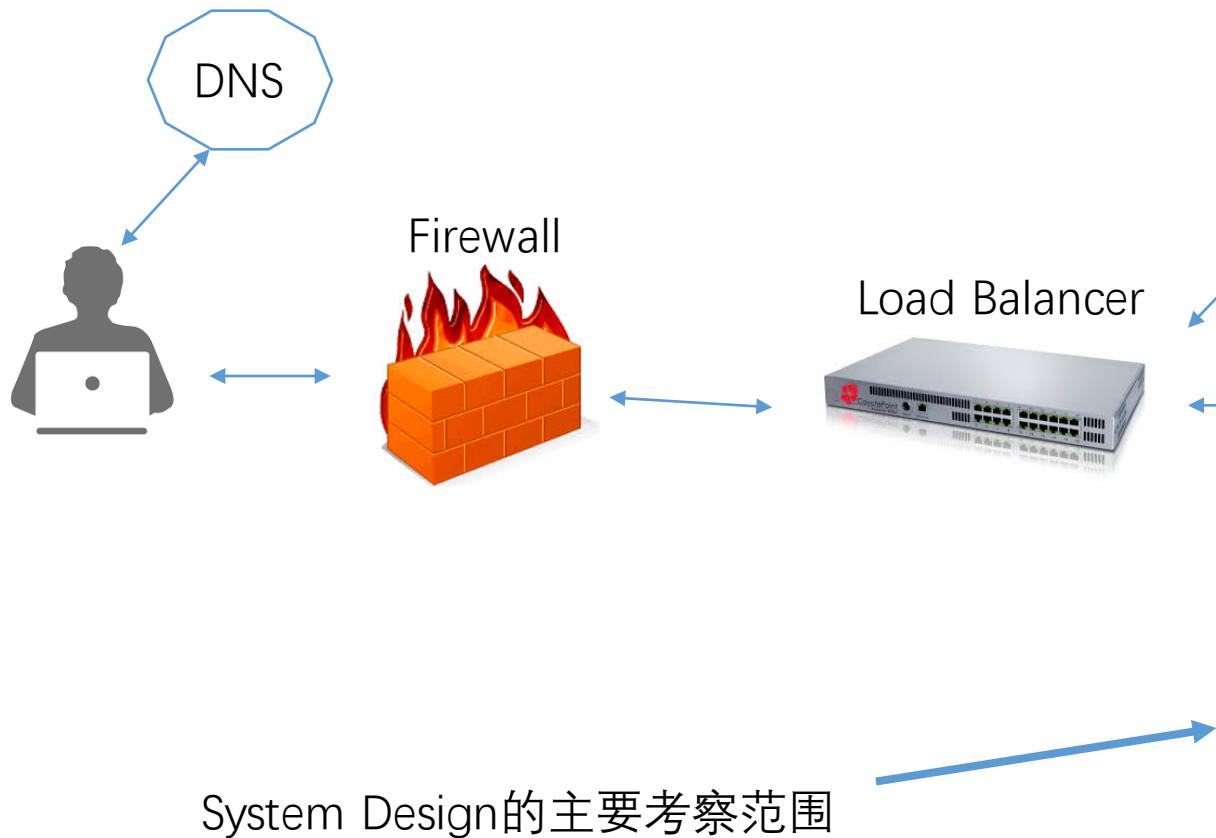


Read more: <http://bit.ly/21LApIb>





复杂的网站系统架构原理图



设计短网址系统

Design Tiny URL

<https://bitly.com/>

<https://goo.gl/>



Google url shortener

The screenshot shows the Google URL Shortener interface. On the left, a user has pasted the URL <http://www.jiuzhang.com> into the input field and clicked the "Shorten URL" button. A reCAPTCHA verification step is shown, indicating successful verification. On the right, the shortened URL <http://goo.gl/2KEEaJ> is displayed along with its original long URL (<http://www.jiuzhang.com>) and a timestamp of "0 minutes ago - details". Below the shortened URL, there is a preview of a webpage titled "九章算法，帮助更多中国人找到好工作" (Jiuzhang Algorithm, helping more Chinese people find good jobs).

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

SNAKE 分析法——

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

Scenario 场景分析

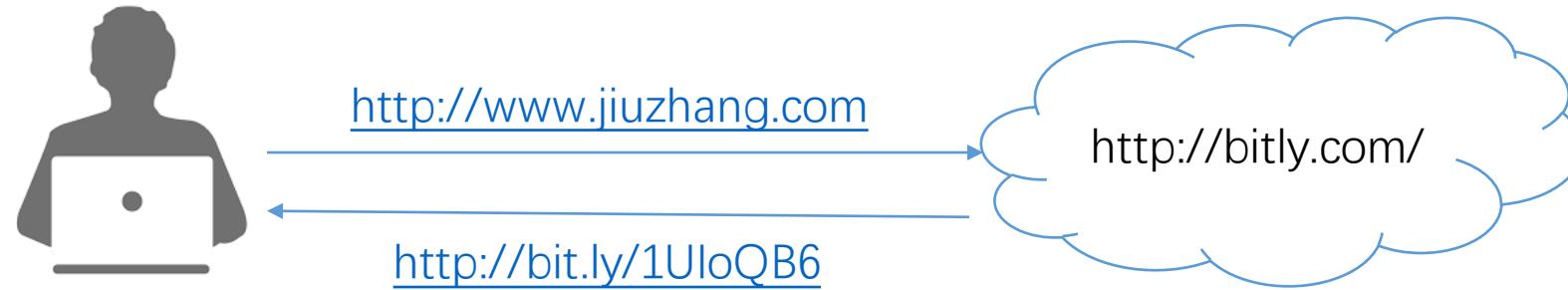
http://ooooooooong.url

http://short.url

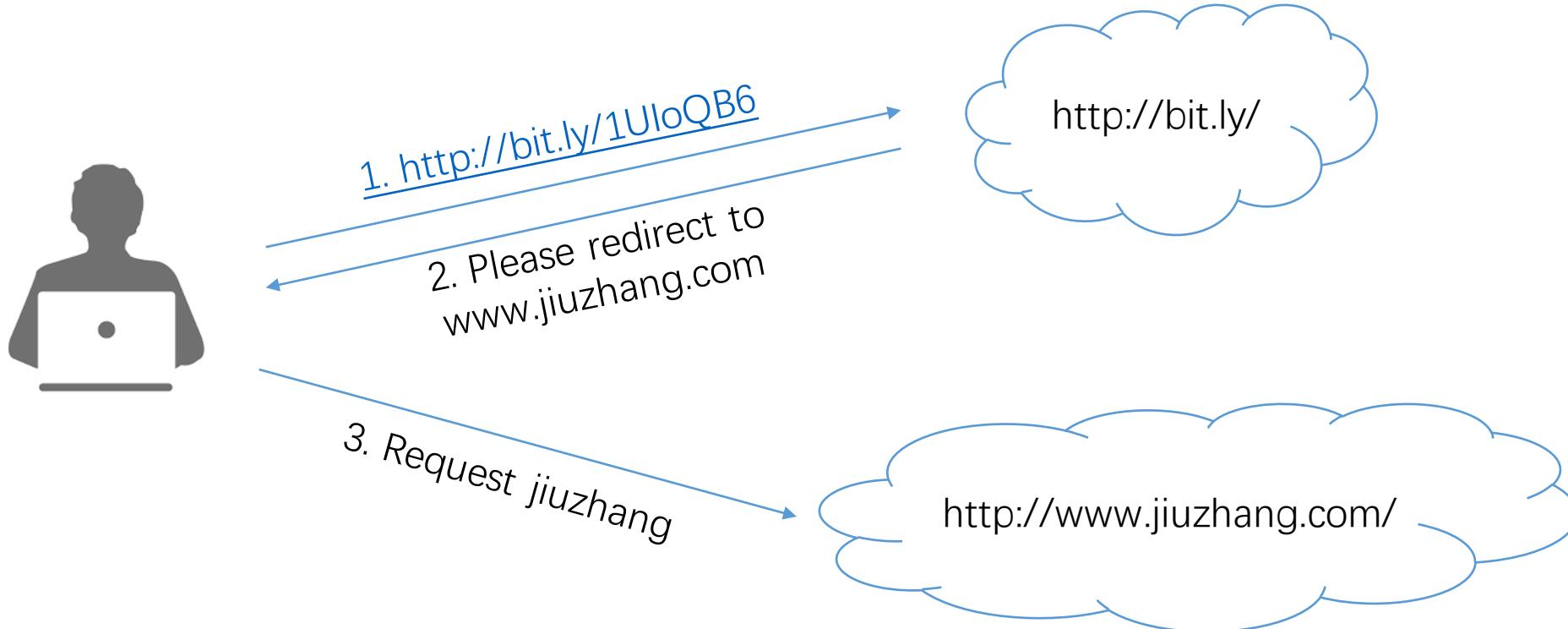


动起手来试试看

- 根据 Long URL 生成一个 Short URL
 - <http://www.jiuzhang.com> => <http://bit.ly/1Ul0QB6>



- 根据 Short URL 还原 Long URL，并跳转
 - <http://bit.ly/1UloQB6> => <http://www.jiuzhang.com>



Needs 需求分析



动起手来试试看，分析QPS和存储

- 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 = $100M * 0.1 / 86400 \sim 100$
 - 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 = $100M * 1 / 86400 \sim 1k$
 - Peak Read QPS = 2k
- 4. 推算每天产生的新的 URL 所占存储
 - $100M * 0.1 \sim 10M$ 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

Application 应用与服务

该系统比较简单，只有一个Application

URL Service

- 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

The image displays three separate browser developer tool windows, each showing an HTTP request and its corresponding response.

- Request URL:** <http://bit.ly/1UI0KVp>
- Request Method:** GET
- Status Code:** 301 Moved Permanently
- Remote Address:** 69.58.188.39:80

Response Headers

- Request URL:** <https://bitly.com/data/shorten>
- Request Method:** POST
- Status Code:** 200 OK
- Remote Address:** 69.58.188.34:443

Response

```
1 {"status_code": 200, "data": {"url": "http://bit.ly/1UI0KVp",}}
```

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

Kilobyte 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

- 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 or NoSQL



选什么？标准是什么？

- 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？

- 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1 : hash function

- 优点, 根据 Long Url 直接生成
- 缺点, 难以设计一个没有冲突的 hash function
 - 克服冲突方法 : 加上 timestamp 重试, 但冲突过多之后, 效率会下降

```
1 while (true) {  
2     short_url = hashfunc(long_url + current_timestamp());  
3     if (!db.exists(short_url=short_url)) {  
4         break;  
5     }  
6 }
```

- 算法2：进制转换 (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
- 课后练习：
- <http://www.lintcode.com/problem/tiny-url/>

```
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;  
}
```

既然要用Sequential ID 那么我们就用SQL吧

下一步：设计表结构 Table Design

Columns stores a specific data type

Row →
Or record

Emp No	Name	Age	Department	Salary
001	Alex S	26	Store	5000
002	Golith K	32	Marketing	5600
003	Rabin R	31	Marketing	5600
004	Jons	26	Security	5100

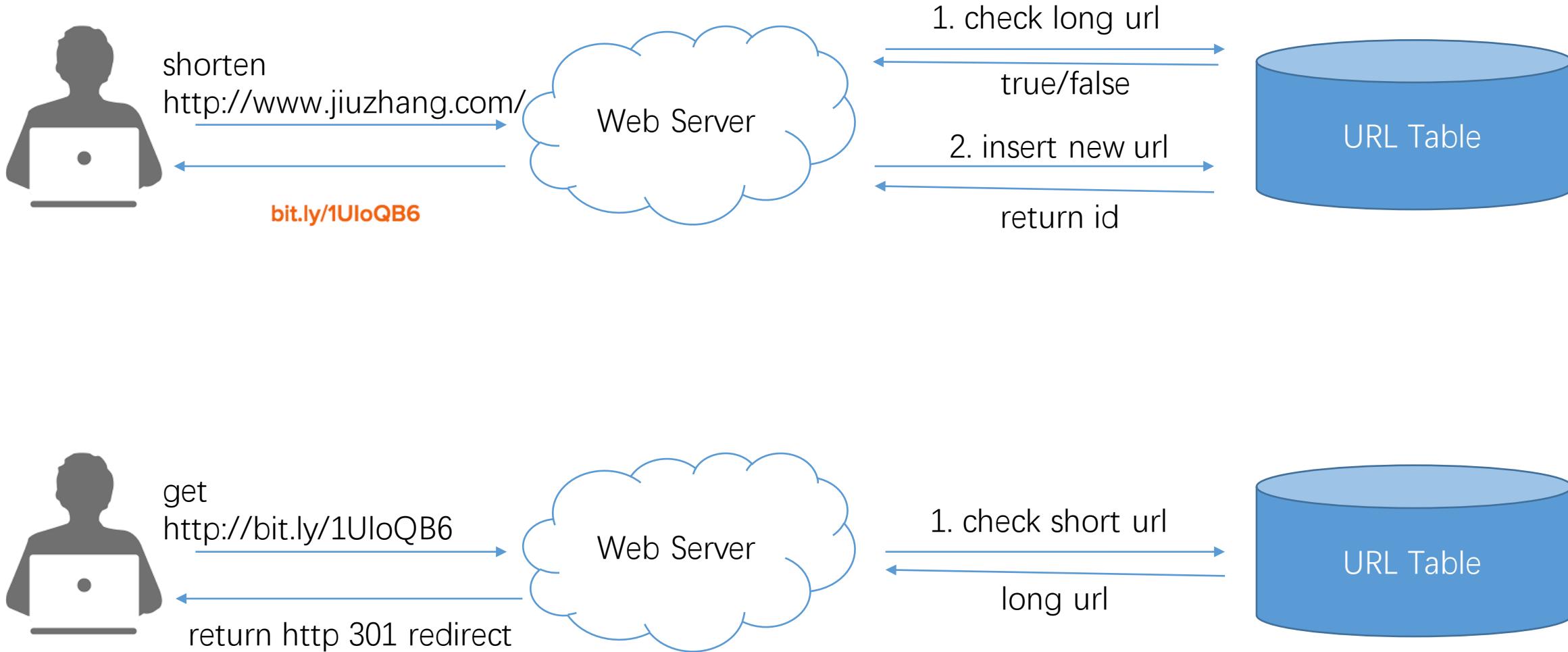


动动手？

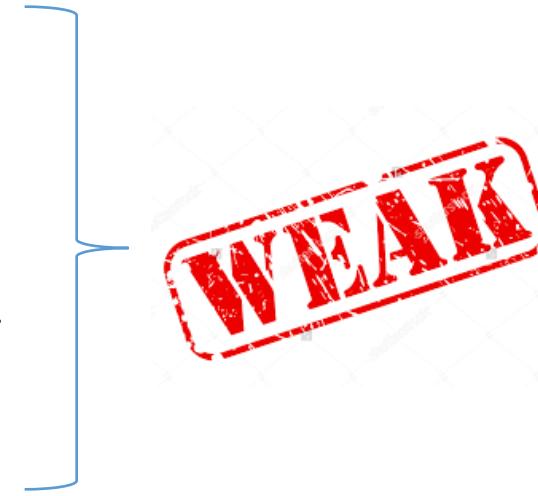
id	url (index=true)
1	http://www.jiuzhang.com/
2	http://www.lintcode.com/
3	http://www.google.com/
4	http://www.facebook.com/

short url 无需存储在table中，因为可以通过Id计算得到

- 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



- 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



Evolve 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？

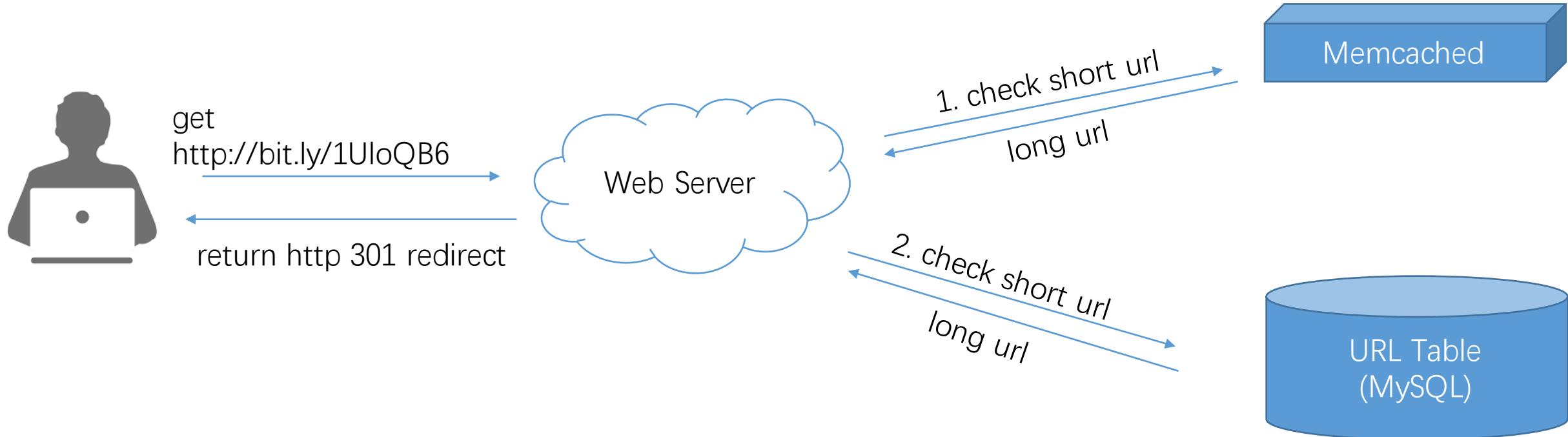


说说看有哪些地方可以加速？



提高读的速度还是写的速度？

- 利用缓存提速



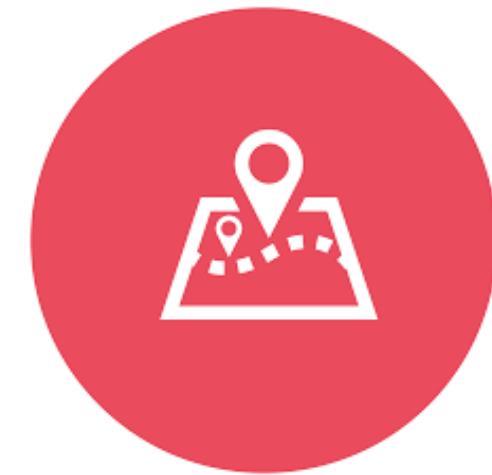
- 场景分析：要做什么功能
- 需求分析：QPS和Storage
- 应用服务：UrlService
- 数据分析：选SQL还是NoSQL
- 数据分析：细化数据库表
- 得到一个Work Solution
- 提高Web服务器与数据服务器之间的访问效率：利用缓存提高读请求的效率

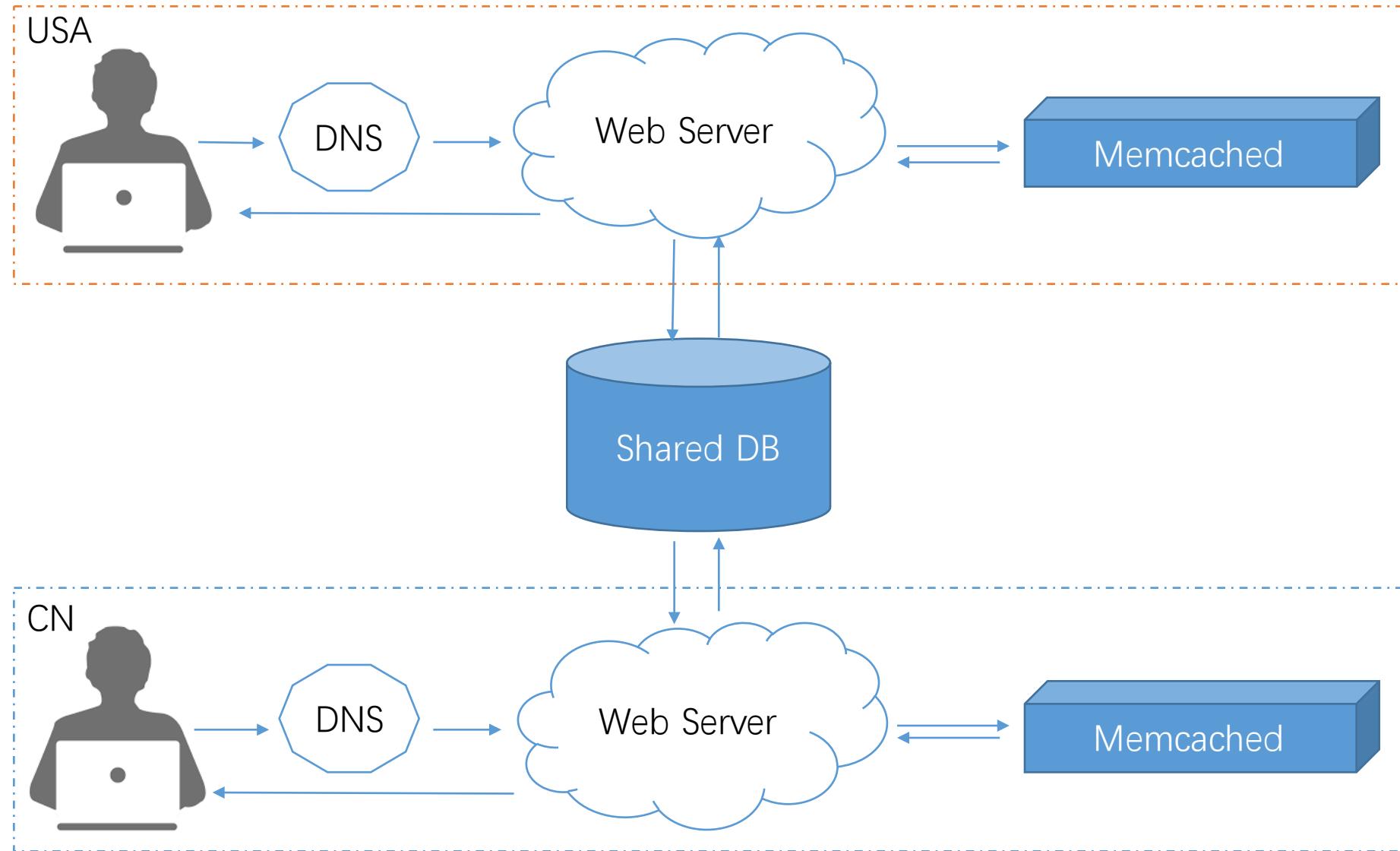
先休息5分钟，然后
下半节的内容，你基本Google不到
要开始教大家弹指神通了



You can you up a
你行你上啊！

- 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





- 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?

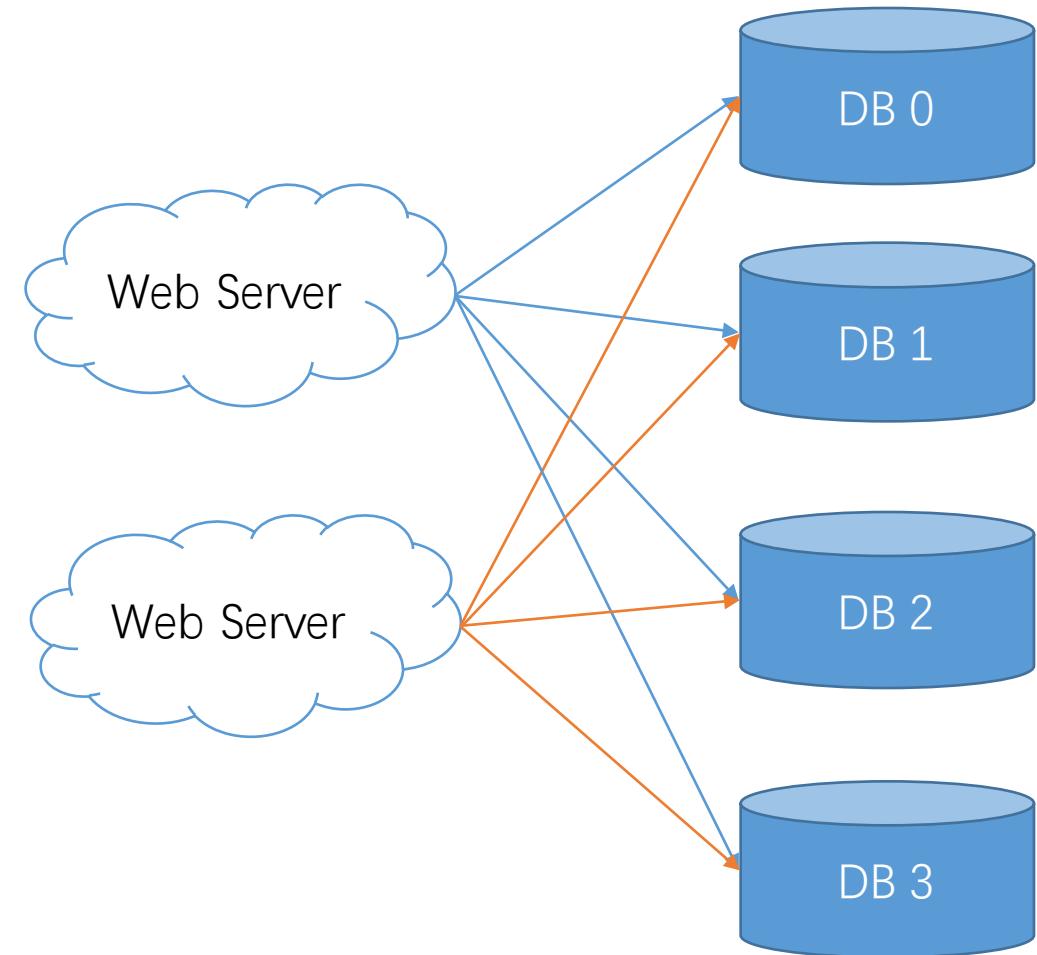
假如我们一开始估算错了，一台MySQL搞不定了



- 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？



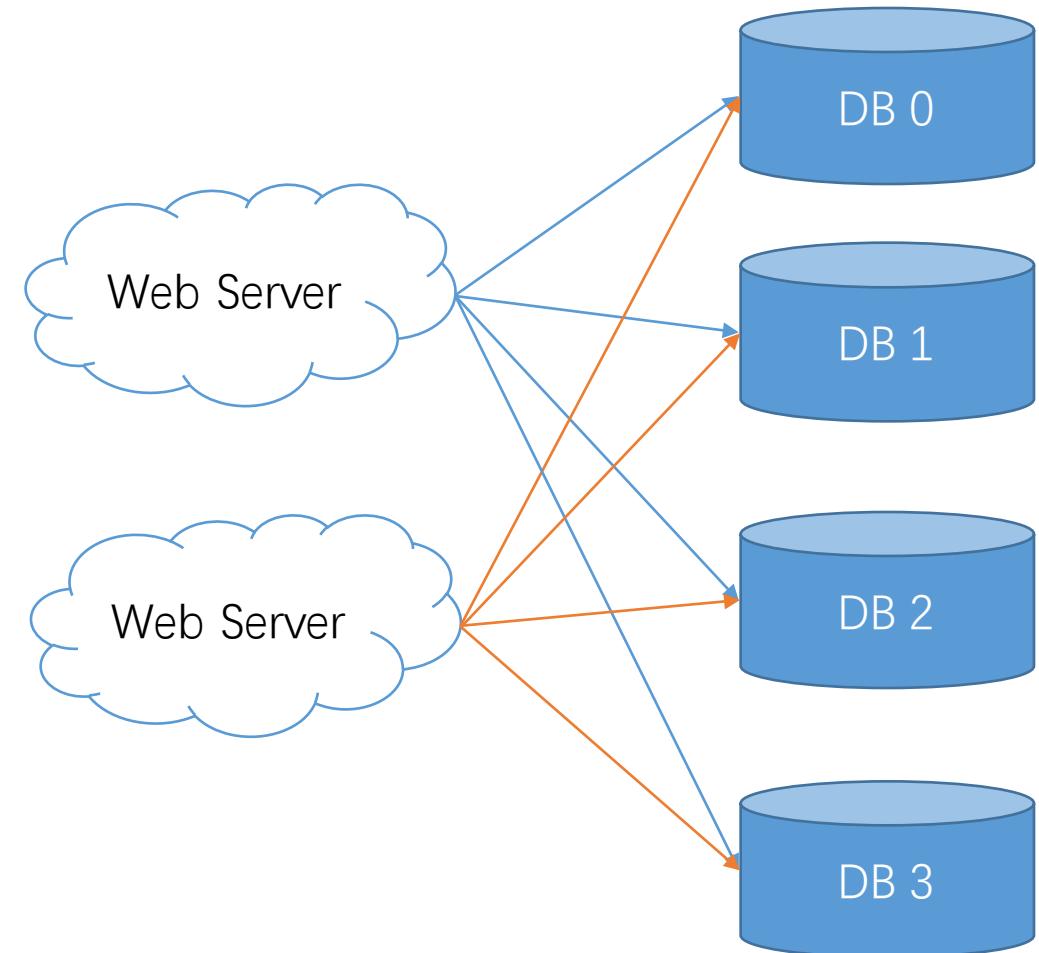
如何将数据分配到多台机器？



- Vertical Sharding —— 将多张数据表分别分配给多台机器
 - Tiny URL 并不适用
 - + Custom URL 也只有两张表
- 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?



- 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

- 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
- 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



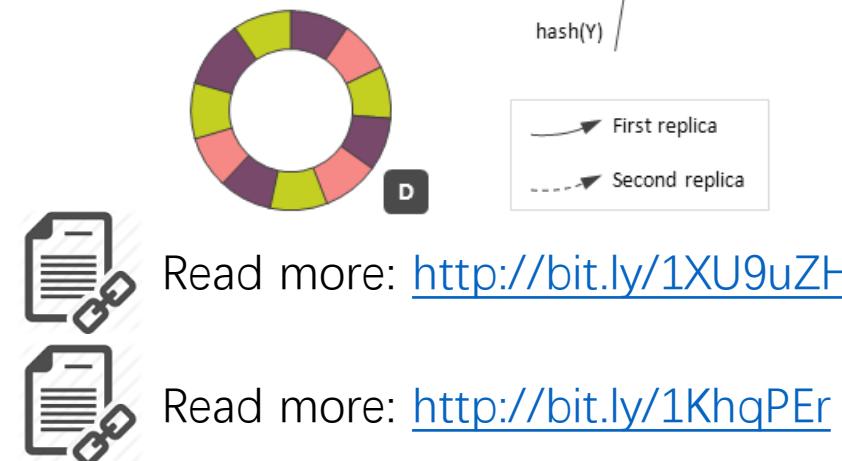
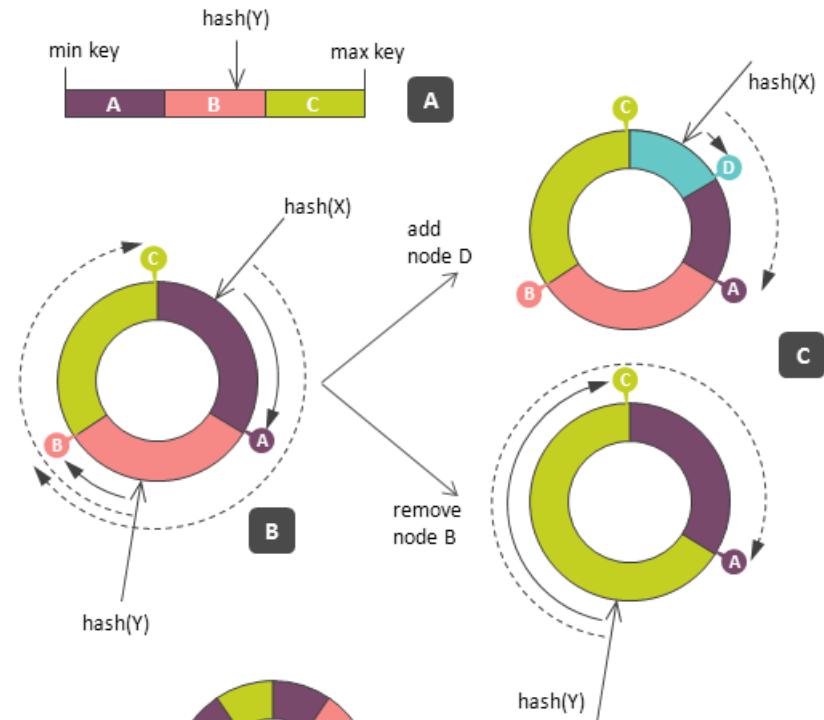
Read more: <http://bit.ly/1RCyPsy>

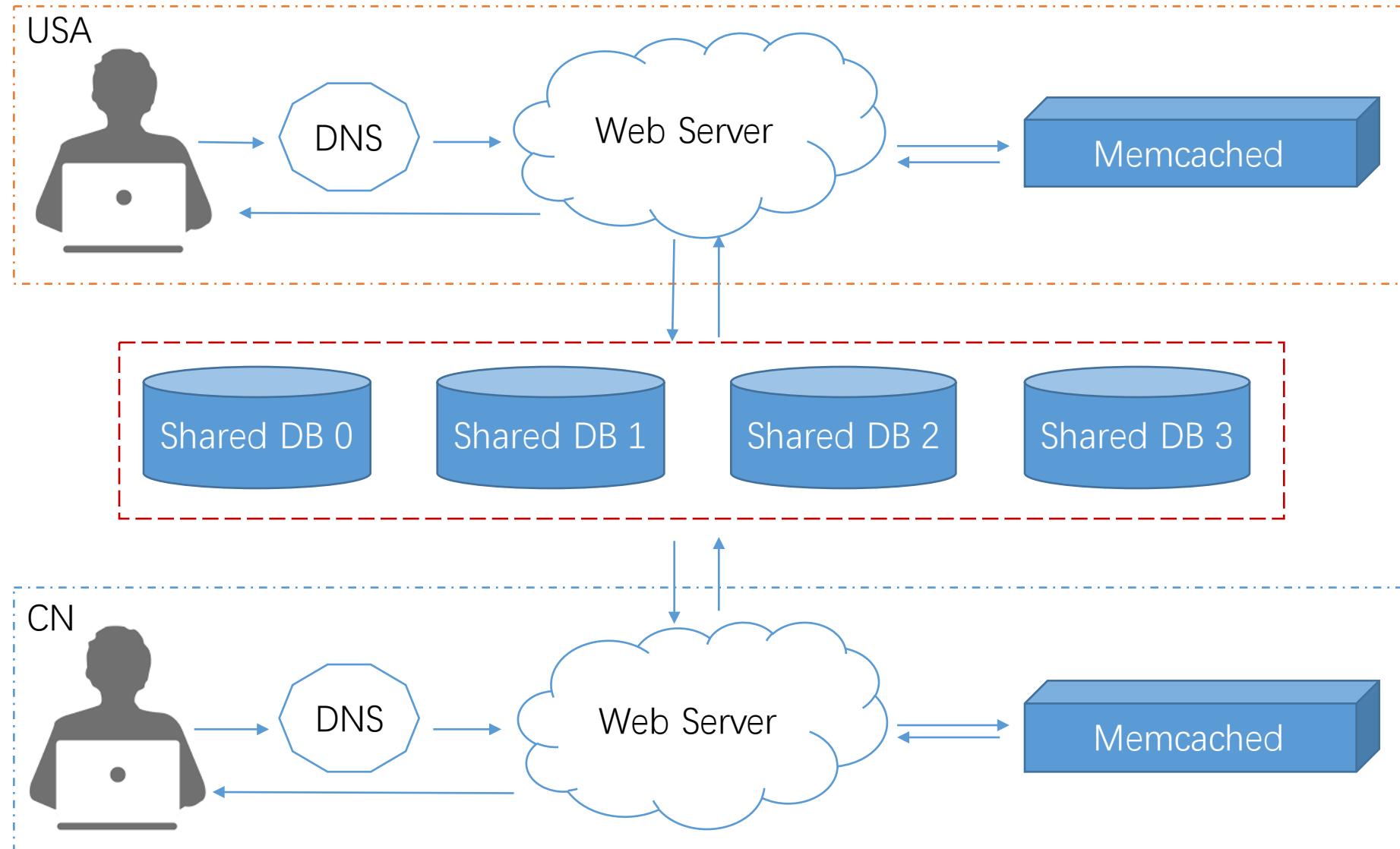


卧槽，那到底怎么做？



- 如果最开始，short key 为6位，下面为short key增加 1 位前置位
 - $AB1234 \rightarrow 0AB1234$
 - 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 $\text{Hash}(\text{long_url}) \% 62$ 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





- 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

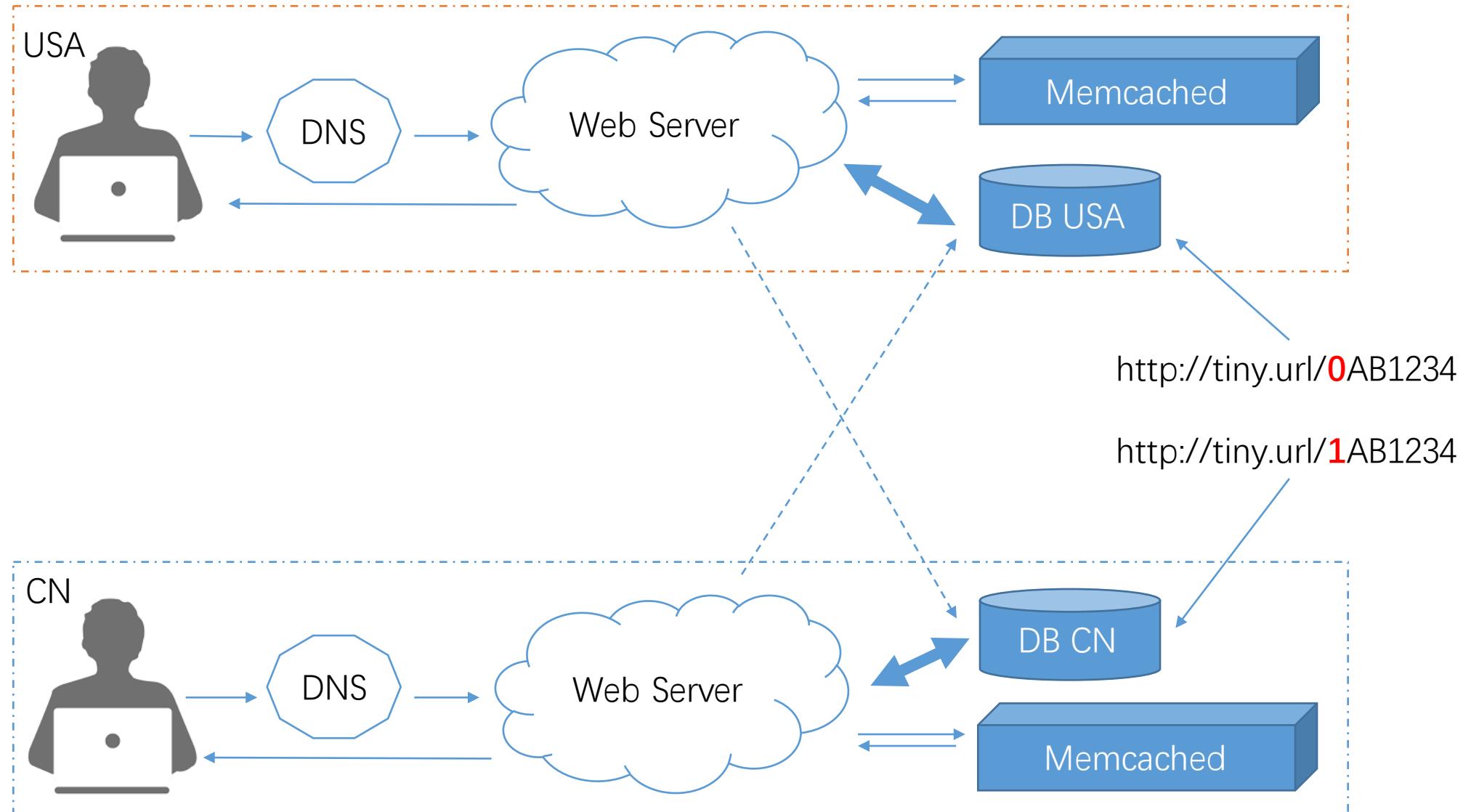
Hired

* Interviewer: 还有可以优化的么？



NI GE SHA BI
你他吗有完没完

- 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



- 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/>

- 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/

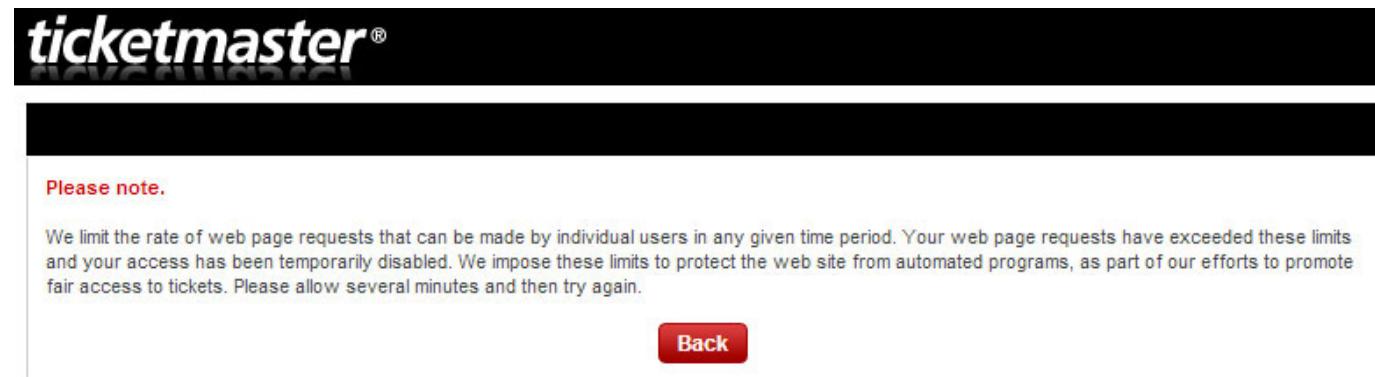
- 课后练习：<http://www.lintcode.com/problem/tiny-url-ii/>

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

- 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
- 提供 Custom URL 的功能

- * Interviewer: How to limit url creation?

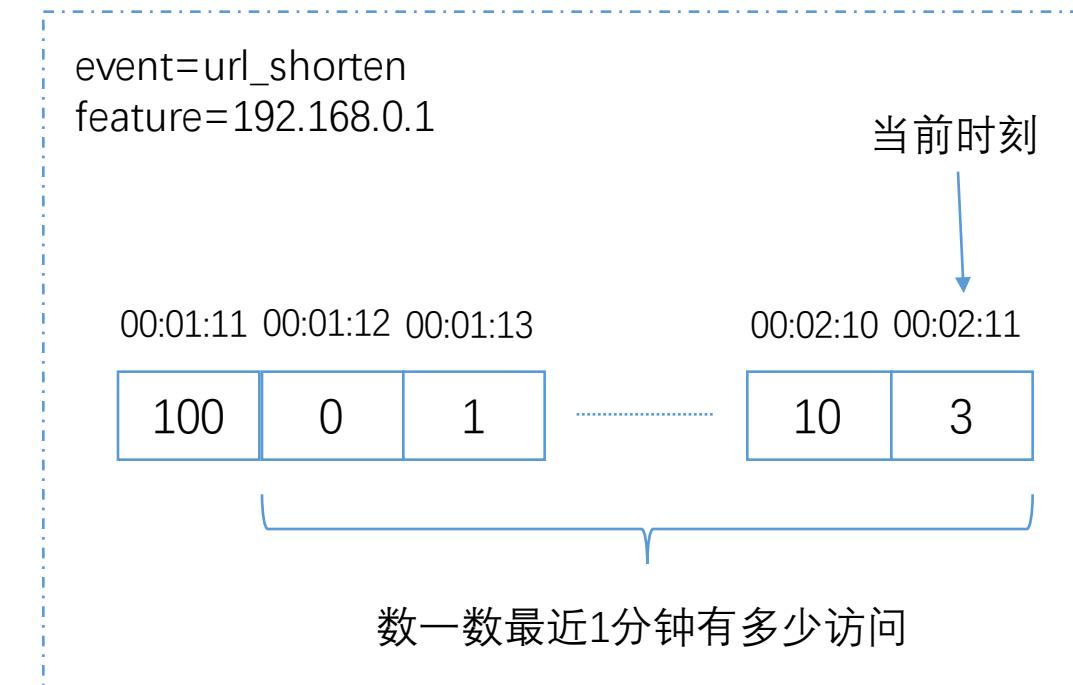
如果有人恶意创建URL怎么办？



- Introduce Rate limiter
 - 网站必用工具
 - 比如一分钟来自同一个邮箱的密码输入错误不能超过5次，一天不超过10次
- 一些Open source的资源
 - Ruby: <https://github.com/ejfinnernan/ratelimit>
 - Django: <https://github.com/jsocol/django-ratelimit>
 - 建议：有空读一读源码
- Rate Limiter 已经是一个小型的系统设计问题
- 我们同样可以用SNAKE分析法进行分析！

- Scenario 场景
 - 根据网络请求的特征进行限制 (feature的选取)
 - IP (未登录时的行为), User (登录后的行为), Email (注册, 登录, 激活)
 - 限制哪些操作?
 - 一般来说只限制 POST (可以理解为就是会产生写操作的请求)
- Needs 系统需要做到怎样的程度
 - 如果在某个时间段内超过了一定数目, 就拒绝该请求, 返回 4xx 错误
 - 2/s, 5/m, 10/h, 100/d
 - 无需做到最近30秒, 最近21分钟这样的限制。粒度太细意义不大
- Application 应用与服务
 - 本身就是一个最小的 Application 了, 无法再细分
- Kilobyte 数据存取
 - 需要记录(log)某个特征 (feature) 在哪个时刻 (time) 做了什么事情 (event)
 - 该数据信息最多保留一天 (对于 rate=5/m 的限制, 那么一次log在一分钟后已经没有存在的意义了)
 - 必须是可以高效存取的结构 (本来就是为了限制对数据库的读写太多, 所以自己的效率必须高与数据库)
 - 所以使用 Memcached 作为存储结构 (数据无需持久化)

- 算法描述
- 用 event+feature+timestamp 作为 memcached 的key
- 记录一次访问：
 - 代码：memcached.increament(key, ttl=60s)
 - 解释：将对应bucket的访问次数+1，设置60秒后失效
- 查询是否超过限制
 - 代码：
 - for t in 0~59 do
 - key = event+feature+(current_timestamp - t)
 - sum+= memcahed.get(key, default=0)
 - Check sum is in limitation
 - 解释：把最近1分钟的访问记录加和



- Evolve 进一步优化：
 - 问：对于一天来说，有86400秒，检查一次就要 86k 的 cache 访问，如何优化？
 - 答：分级存储。
 - 之前限制以1分钟为单位的时候，每个bucket的大小是1秒，一次查询最多60次读
 - 现在限制以1天为单位的时候，每个bucket以小时为单位存储，一次查询最多24次读
 - 同理如果以小时为单位，那么每个bucket设置为1分钟，一次查询最多60次读
 - 问：上述的方法中存在误差，如何解决误差？
 - 首先这个误差其实不用解决，访问限制器不需要做到绝对精确。
 - 其次如果真要解决的话，可以将每次log的信息分别存入3级的bucket（秒，分钟，小时）
 - 在获得最近1天的访问次数时，比如当前时刻是23:30:33，加总如下的几项：
 - 在秒的bucket里加和 23:30:00 ~ 23:30:33 (计34次查询)
 - 在分的bucket里加和 23:00 ~ 23:29 (计30次查询)
 - 在时的bucket里加和 00 ~ 22 (计23次查询)
 - 在秒的bucket里加和昨天 23:30:34 ~ 23:30:59 (计26次查询)
 - 在分的bucket里加和昨天 23:31 ~ 23:59 (计29次查询)
 - 总计耗费 $34 + 30 + 23 + 26 + 29 = 142$ 次cache查询，可以接受



你觉得是否需要解决误差？

- 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
- 提供 Custom URL 的功能
- 提供 访问限制功能：Rate Limiter

* Interviewer: How to analysis?

Google url shortener



- <https://www.datadoghq.com/>
- 同样进行SNAKE 分析！
- Scenario 设计些啥
 - 对于用户对于某个链接的每次访问，记录为一次访问
 - 可以查询某个链接的被访问次数
 - 知道总共多少次访问
 - 知道最近的x小时/x天/x月/x年的访问曲线图
- Needs 设计得多牛逼？
 - 假设 Tiny URL 的读请求约 2k 的QPS
- Application
 - 自身为一个独立的Application，无法更细分

- Kilobyte

- 基本全是写操作，读操作很少
- 需要持久化存储（没memcached什么事儿了）
- SQL or NoSQL or File System?
 - 其实都可以，业界的一些系统比如 Graphite 用的是文件系统存储
 - 这里我们假设用NoSQL存储吧
- 用NoSQL的话，key 就是 tiny url 的 short_key，value 是这个key的所有访问记录的统计数据
 - 你可能会奇怪为什么value可以存下一个key的所有访问数据（比如1整年）
- 我们来看看value的结构
- 核心点是：
 - 今天的数据，我们以分钟为单位存储
 - 昨天的数据，可能就以5分钟为单位存储
 - 上个月的数据，可能就以1小时为单位存储
 - 去年的数据，就以周为单位存储
 - ...
- **用户的查询操作通常是查询某个时刻到当前时刻的曲线图**
- **也就意味着，对于去年的数据，你没有必要一分钟一分钟的进行保存**
- 多级Bucket的思路，是不是和Rate Limiter如出一辙！

```
...  
2016/02/26 23 1h 200  
...  
2016/03/27 23:50 5m 40  
2016/03/27 23:55 5m 30  
2016/03/28 00:00 1m 10  
2016/03/28 00:01 1m 21  
...  
2016/03/28 16:00 m 2
```

- Evolve
 - 问：2k的QPS这么大，往NoSQL的写入操作也这么多么？
 - 答：不是。
 - 可以先将最近15秒钟的访问次数 Aggregate 到一起，写在内存里
 - 每隔15秒将记录写给NoSQL一次，这样写QPS就降到了100多
 - 问：如何将昨天的数据按照5分钟的bucket进行整理？
 - 答：对老数据进行瘦身
 - 当读发现一个key的value比较多的时候，就触发一次“瘦身”操作
 - 瘦身操作把所有老的记录进行 Aggregate
 - 这些旧数据的记录的专业名词叫做：Retention

- 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
- 提供 Custom URL 的功能
- 提供 访问限制功能：Rate Limiter
- 提供 访问统计功能：Data dog

} S+N: 各种问面试官问题，搞清楚要干嘛

} A+K: 根据问到的内容进行分析
得到一个基本可以work的方案

WEAK

E: 提速， Scale

Hired

} E: 做点新功能



Hired

- 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很多fancy的概念更有用

今天我们学会了什么？

- 学会了4道题目：
 - What happened if you visit www.google.com
 - Design Tiny URL
 - Design Rate limiter
 - Design Data dog
- 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

- 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/21LAplb>
 - 选做：搭建一个tiny url的网站
 - 加QQ群：186738019
- Global Sequential ID
 - <http://bit.ly/1RCyPsy>
- Consistent Hashing
 - <http://bit.ly/1XU9uZH>
 - <http://bit.ly/1KhaPEr>
- Rate Limiter
 - 源码阅读：<https://github.com/jsocol/django-ratelimit>

- <http://www.lintcode.com/problem/tiny-url/>
- <http://www.lintcode.com/problem/tiny-url-ii/>
- <http://www.lintcode.com/problem/rate-limiter/>
- <http://www.lintcode.com/problem/web-logger/>
- <http://www.lintcode.com/problem/consistent-hashing/>
- <http://www.lintcode.com/problem/consistent-hashing-ii/>

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

