

# 网站与设计短网址系统

## Web System & Design Tiny Url

课程版本 v5.0    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

- 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](http://www.google.com)?
- 设计一个短网址系统 Design Tiny Url
  - 4S 分析法
  - No Hire / Weak Hire / Hire / Strong Hire

# 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

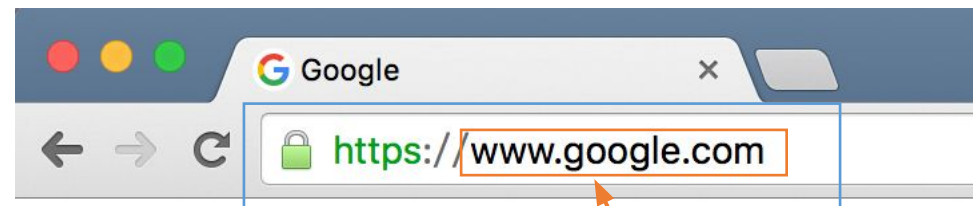
面试官：当你访问[www.google.com](http://www.google.com)的时候发生了什么？



- 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](https://www.google.com) 的时候发生了什么

- 你在浏览器输入 [www.google.com](https://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](https://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
```

## 当你访问 [www.google.com](http://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](http://www.google.com) 的时候发生了什么

django



动手用Django搭建一个网站(约1天)



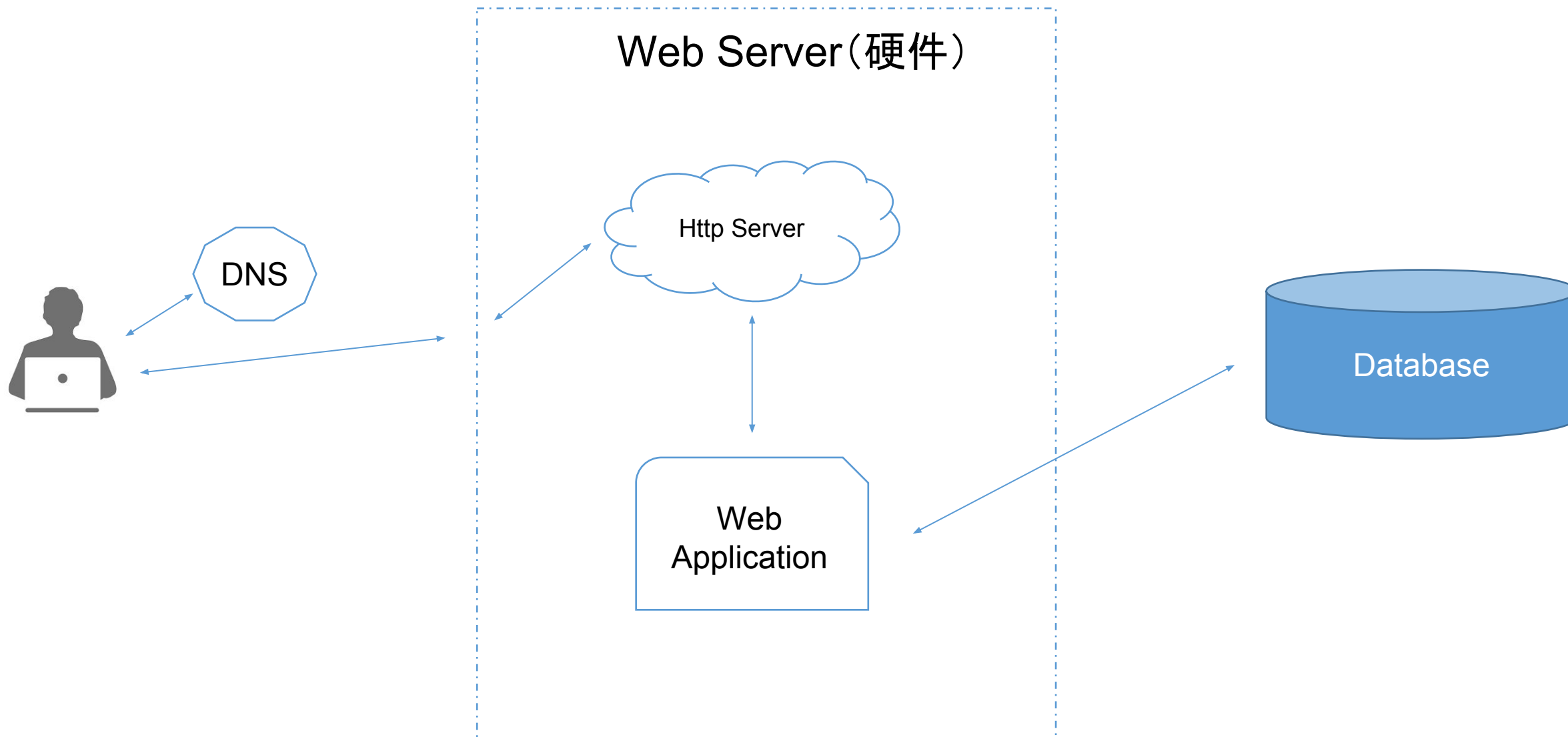
Read more: <http://bit.ly/21LAplb>

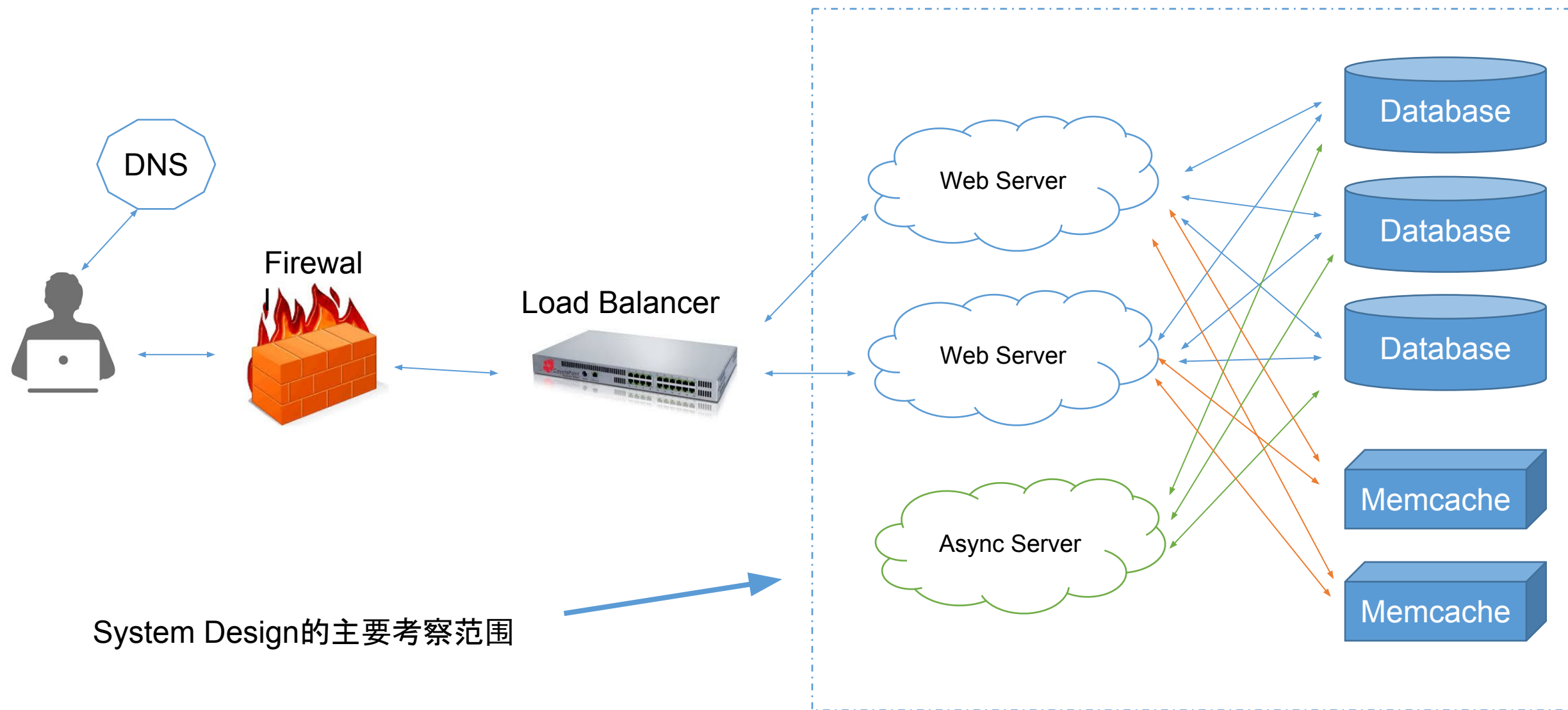
node JS



U B E R







# 设计短网址系统 Design Tiny URL

<https://bitly.com/>

<https://goo.gl/>

bitly

Google url shortener

## Google url shortener

Paste your long URL here:

<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.

Google

<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://loooooooooong.url>

<http://short.url>

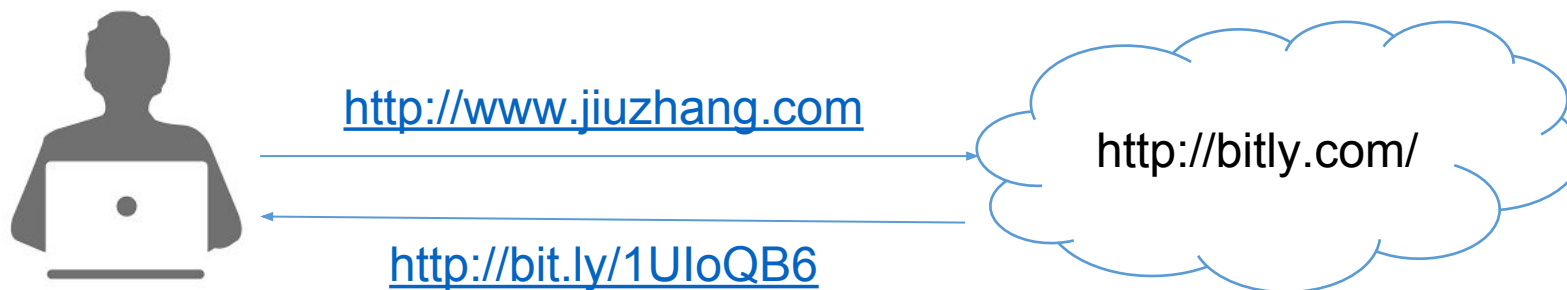


动起手来试试看



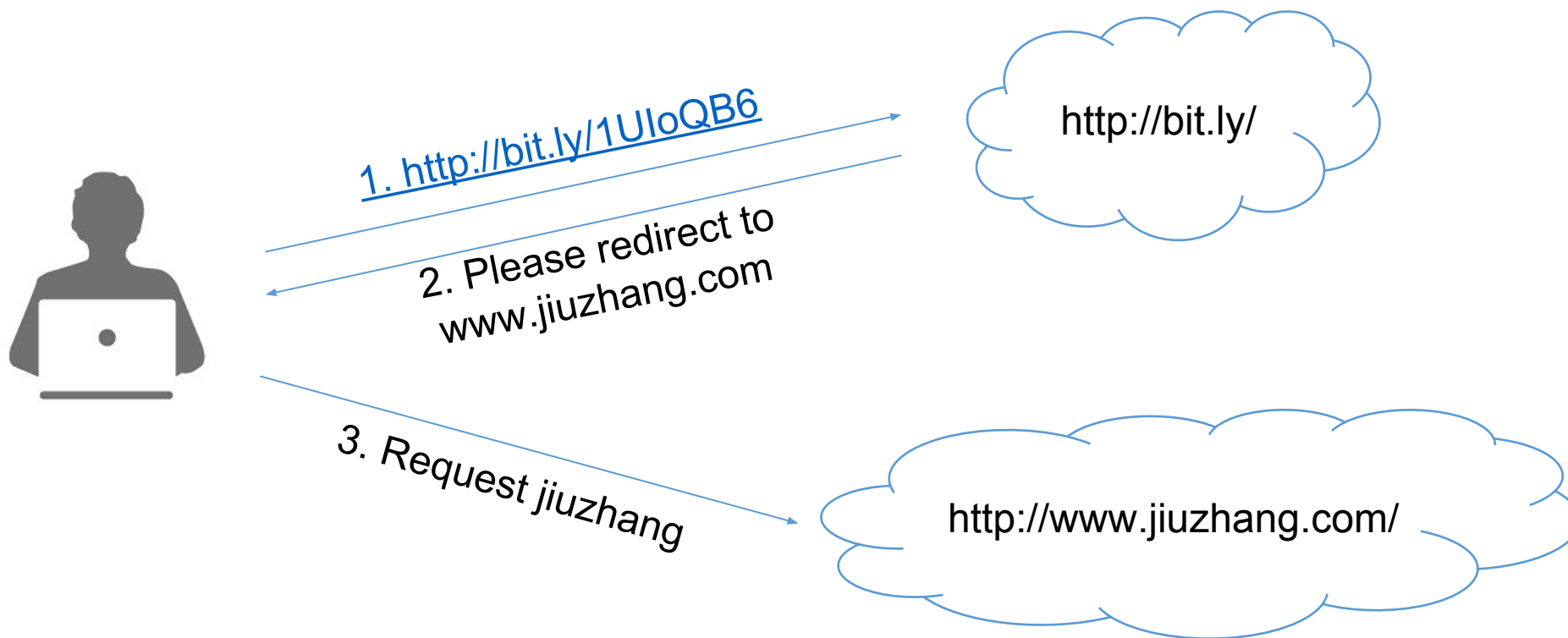
## Scenario 场景 —— 我要设计啥

- 根据 Long URL 生成一个 Short URL
  - <http://www.jiuzhang.com> => <http://bit.ly/1UloQB6>



## Scenario 场景 —— 我要设计啥

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



# QPS



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
  - $\text{Average Write QPS} = 100\text{M} * 0.1 / 86400 \sim 100$
  - $\text{Peak Write QPS} = 100 * 2 = 200$
- 3. 推算点击一条Tiny URL的QPS
  - 假设每个用户平均点1个Tiny URL
  - $\text{Average Read QPS} = 100\text{M} * 1 / 86400 \sim 1\text{k}$
  - $\text{Peak Read QPS} = 2\text{k}$
- 4. 推算每天产生的新的 URL 所占存储
  - $100\text{M} * 0.1 \sim 10\text{M}$  条
  - 每一条 URL 长度平均 100 算, 一共1G
  - 1T 的硬盘可以用 3 年

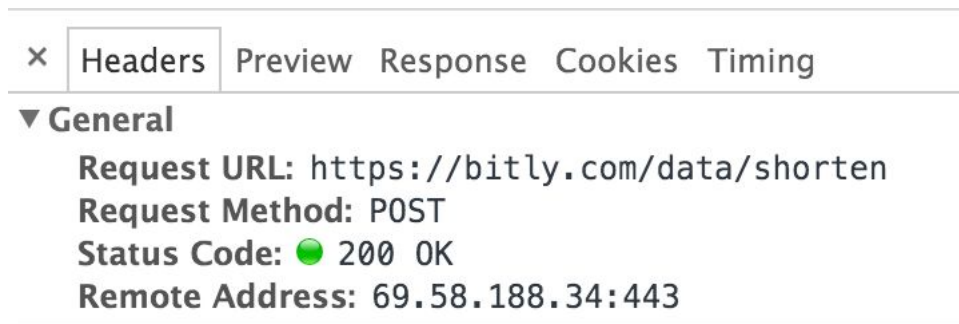
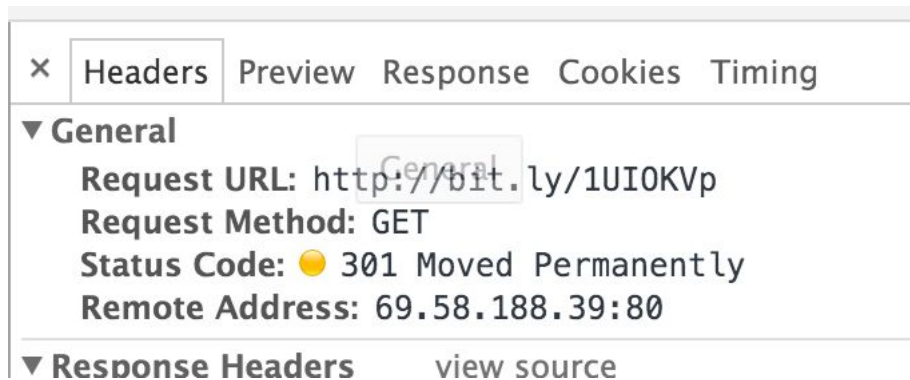
2k QPS

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

- 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 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 or NoSQL



选什么？标准是什么？

## 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/

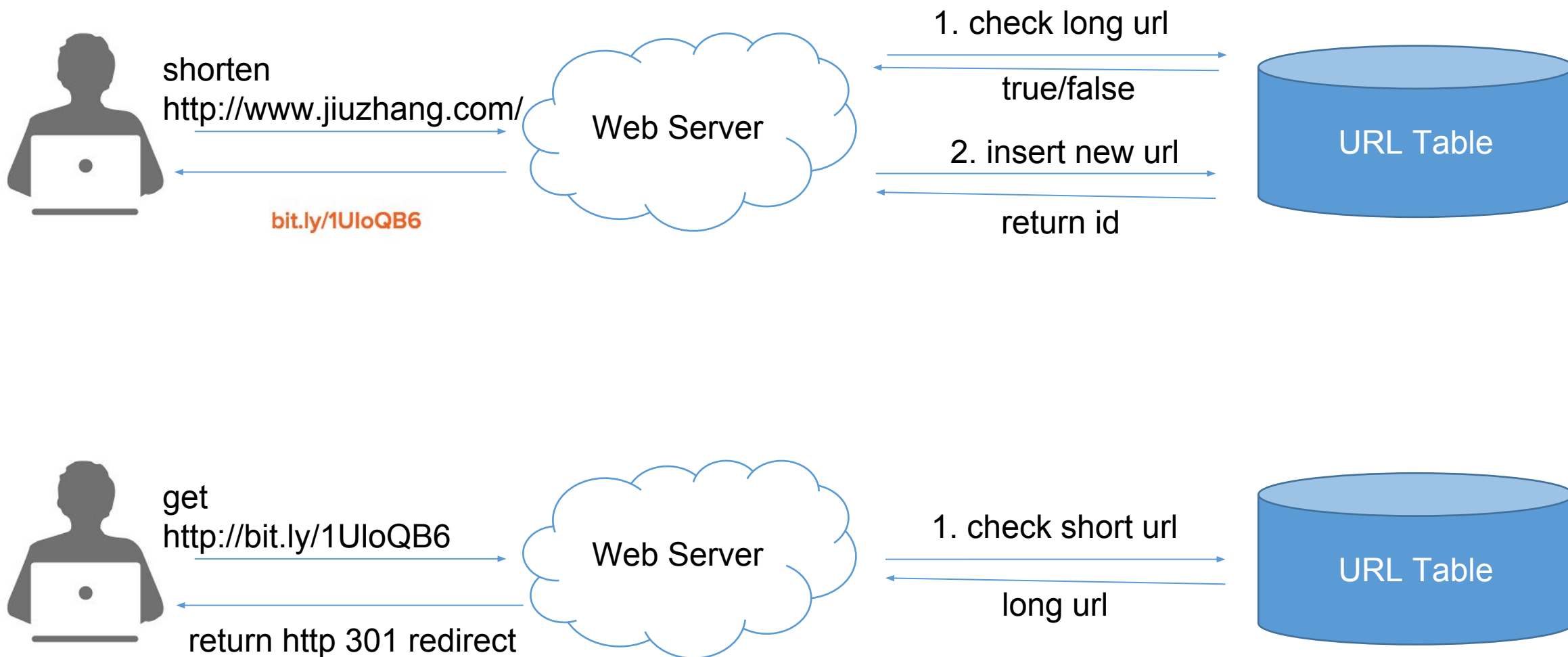
short url 无需存储在table中, 因为可以通过Id计算得到



## 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



## 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

WEAK

# 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？

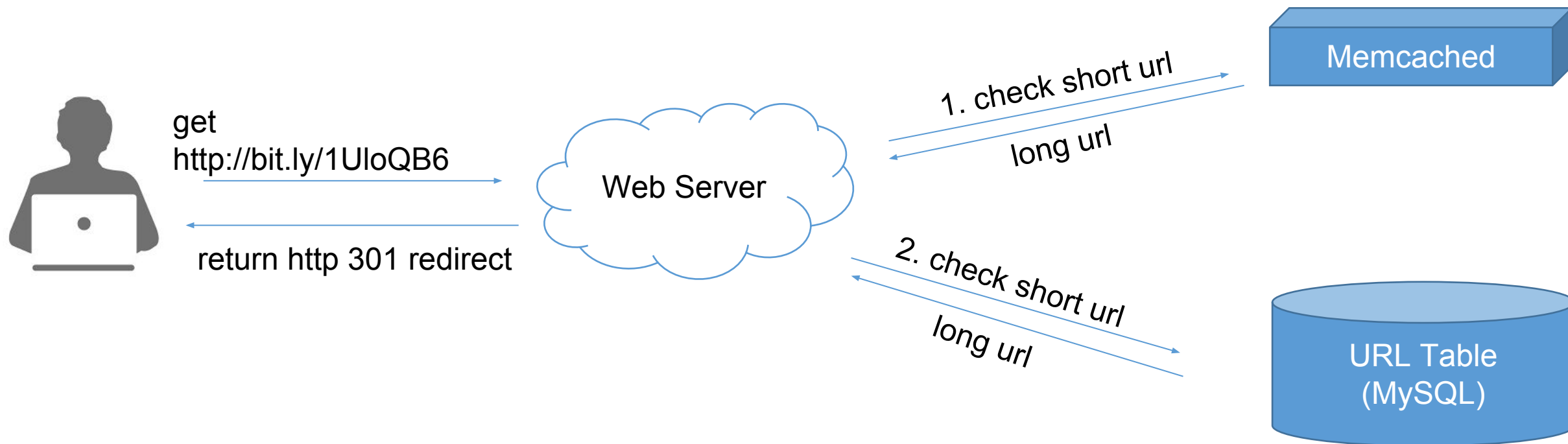


说说看有哪些地方可以加速？



提高读的速度还是写的速度？

- 利用缓存提速



## 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

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

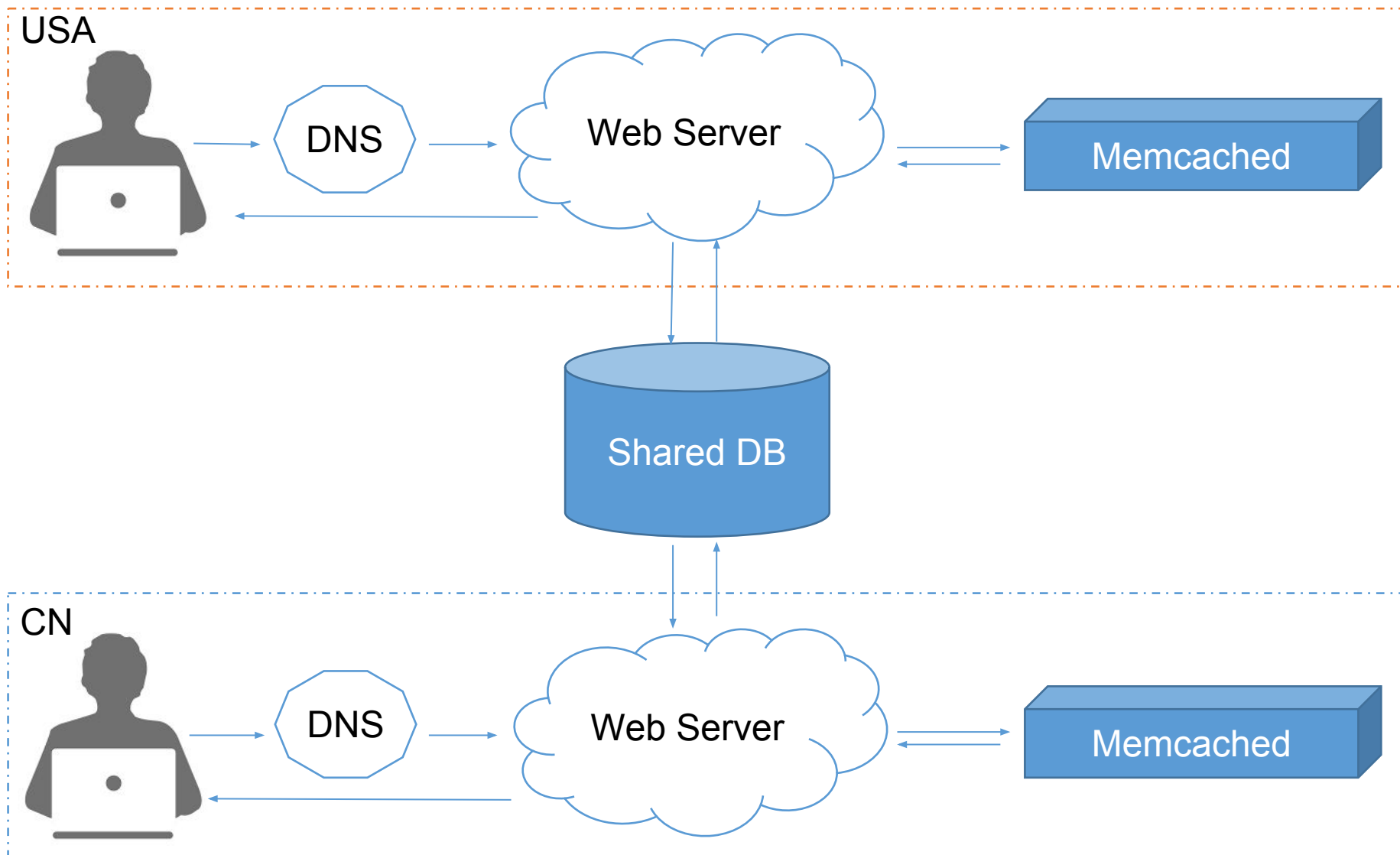


You can you up a  
你行你上啊！



- 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





## 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

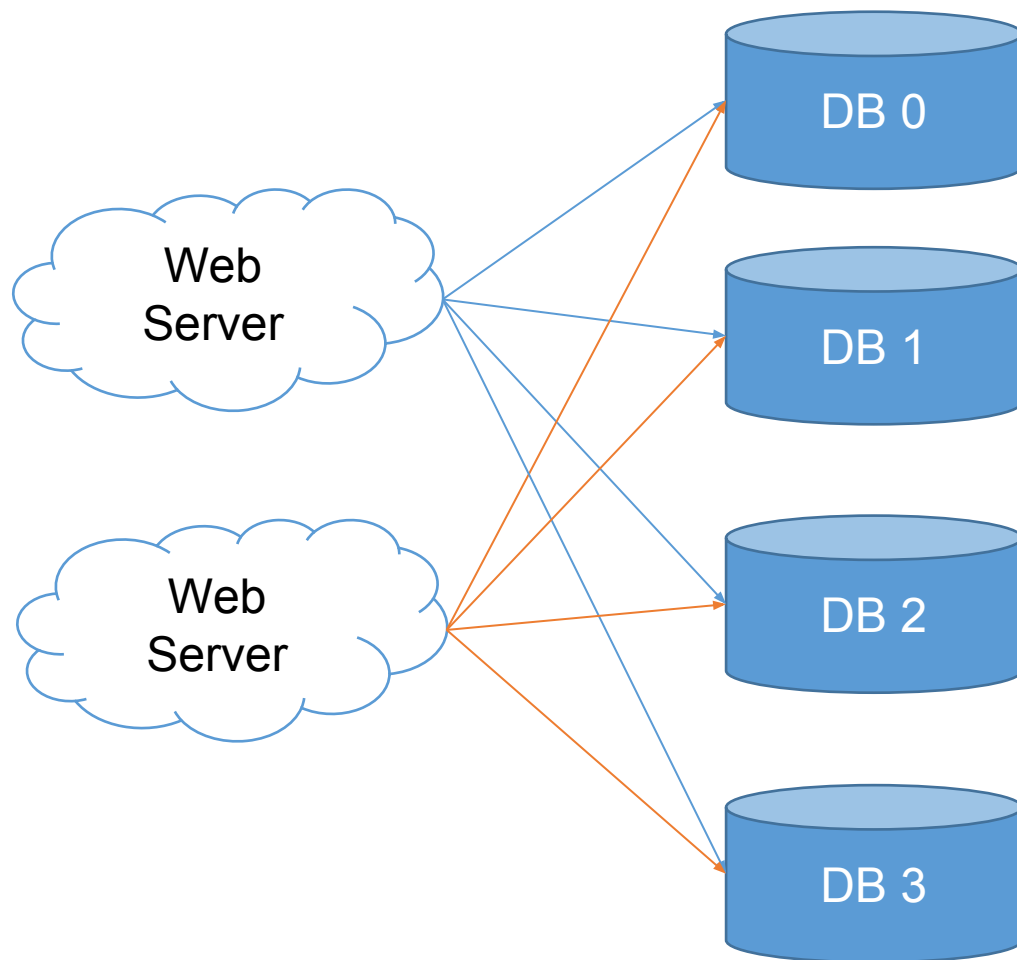


## Scale —— 如何扩展？

- 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？




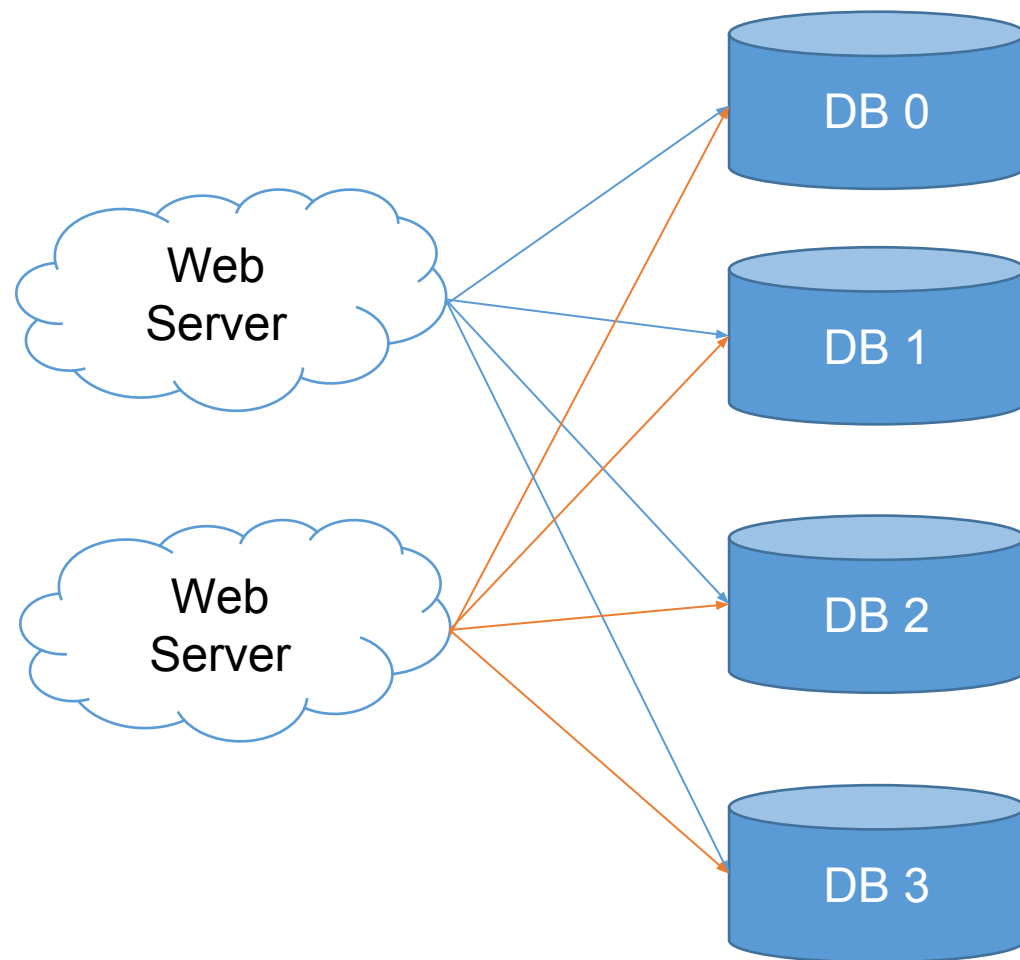
如何将数据分配到多台机器？



## 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
  - + 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?



## 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

- 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>



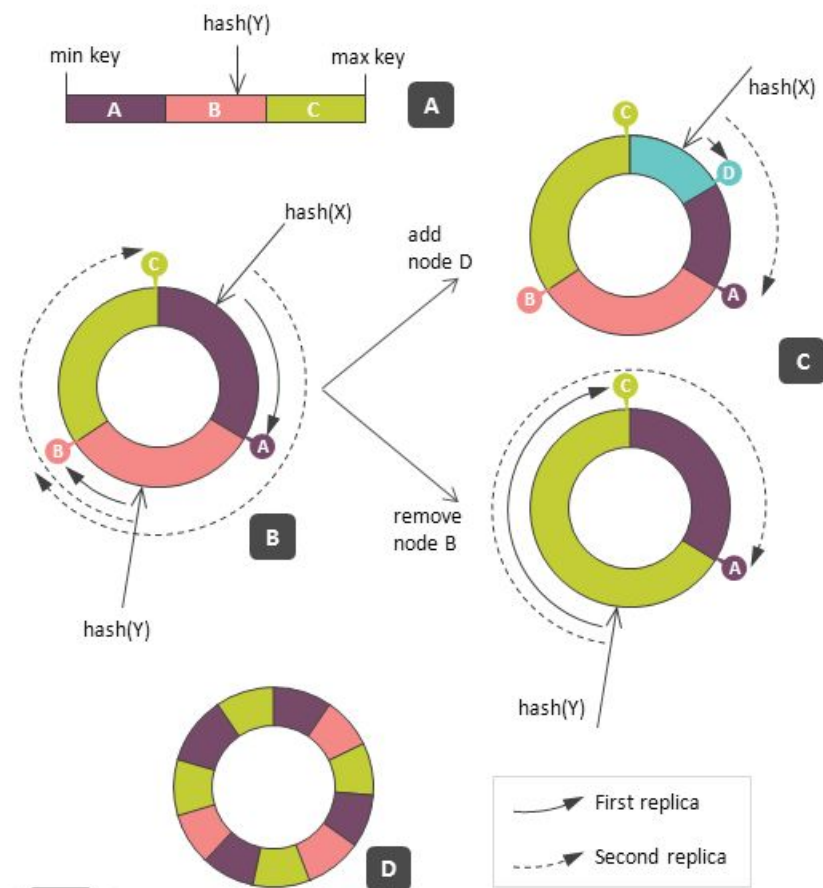
卧槽, 那到底怎么做?



我要死了

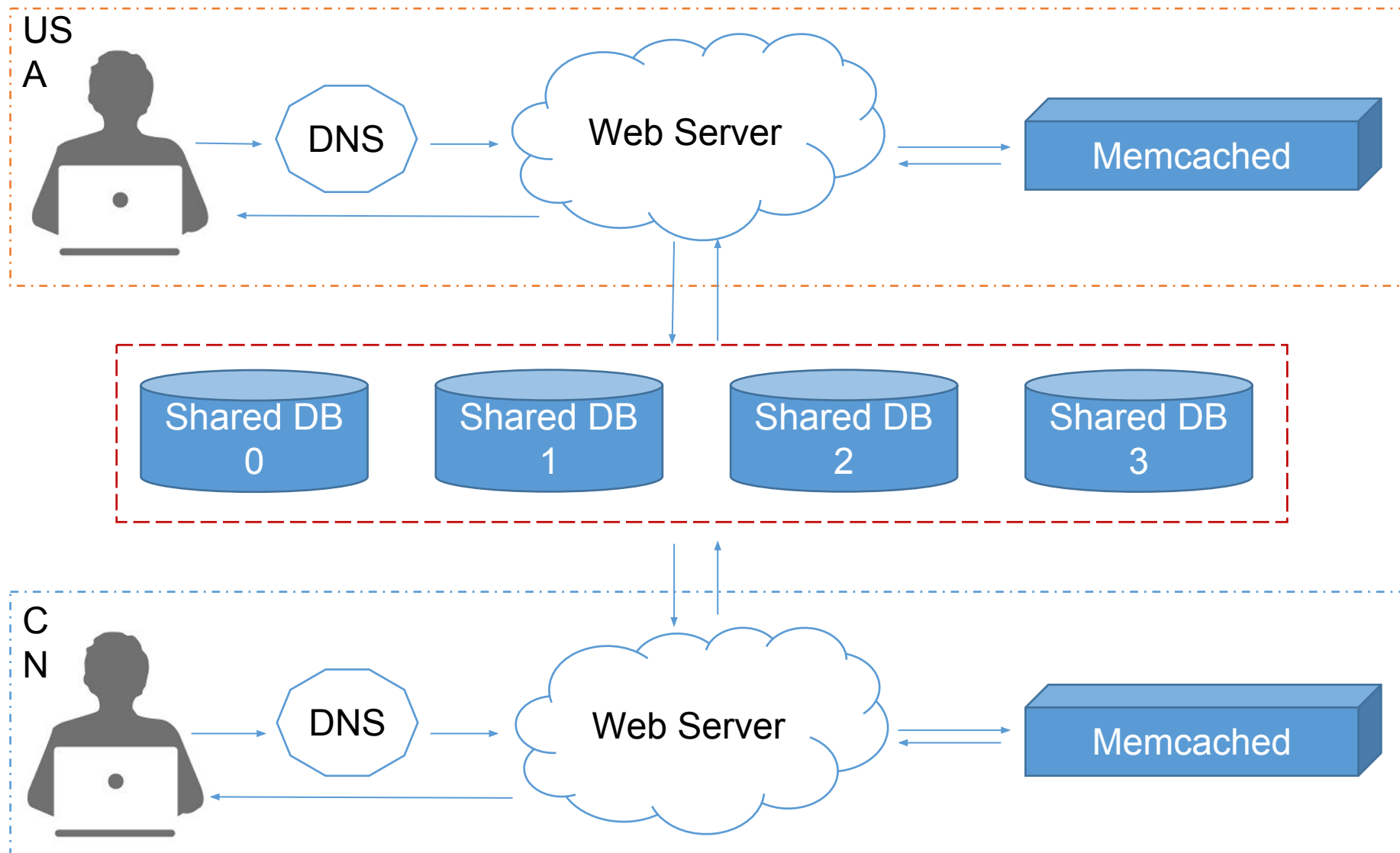


- 如果最开始, short key 为6位, 下面为short key增加 1 位前置位
  - AB1234 → 0AB1234
  - 还有一种做法是把第1位单独留出来做 sharding key, 总共还是6位
- 该前置位的值由  $\text{Hash}(\text{long\_url}) \% 62$  得到
- 该前置位则为 sharding key
  - Consistent Hashing
    - 相关练习 <http://www.lintcode.com/en/problem/consistent-hashing/>
  - 将环分为62个区间
  - 每台机器在环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



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

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



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



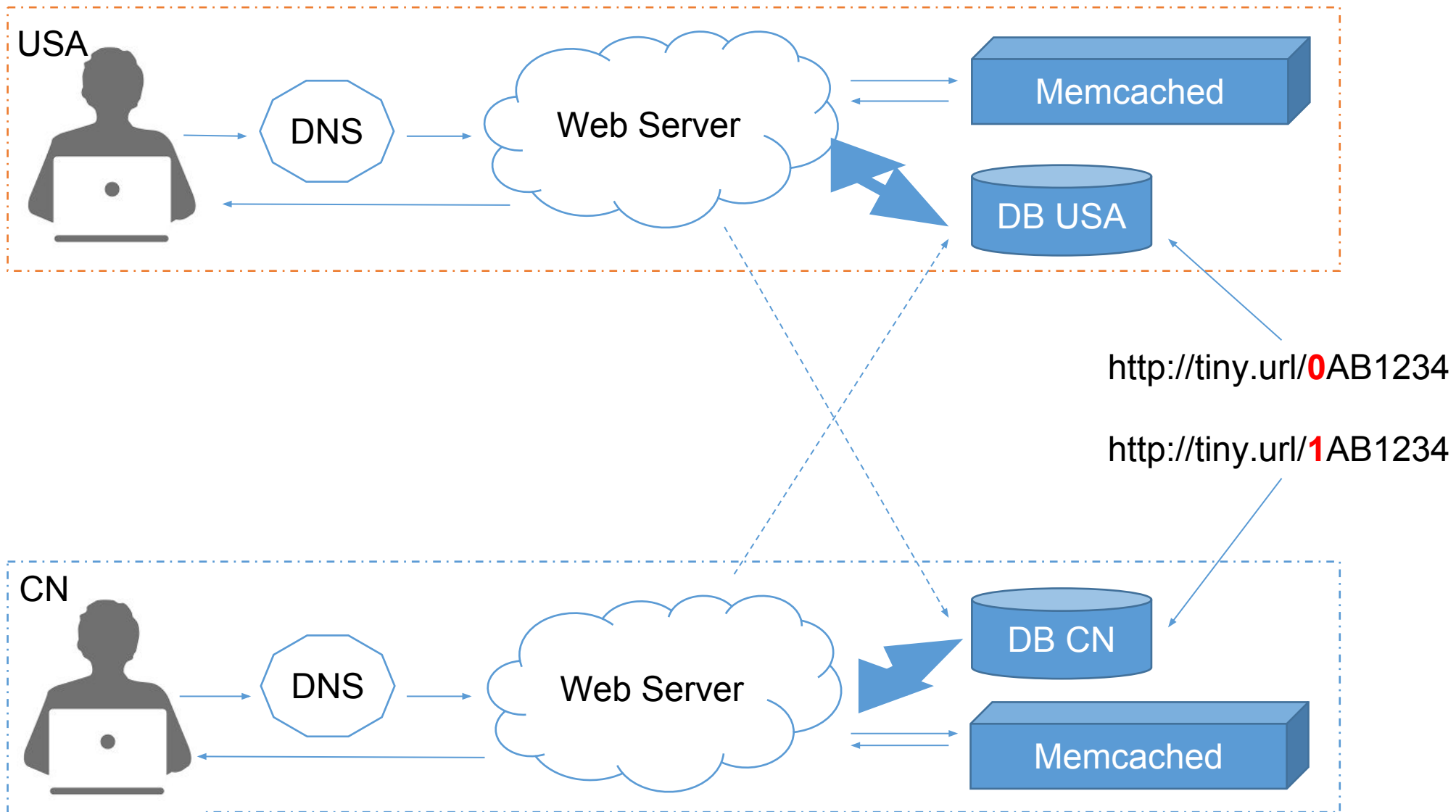
\* Interviewer: 还有可以优化的么？



NI GE SHA BI  
你他吗有完没完

## 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



## 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/>



- 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/>

- 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

Scenario: 各种问面试官问题, 搞清楚要干嘛

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

**WEAK**

- 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

**HIRED**

follow up



**HIRED**

# 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

# 今天我们学会了什么？

---

- 学会了2道题目：
  - What happened if you visit [www.google.com](http://www.google.com)
  - 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/21LAplb>
  - 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>

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

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

