

# 网站系统, API 设计与短网址 Web System, API Design & Tiny URL

课程版本 v6.0    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>

# 为什么了解网站系统如此重要？

System Design 几乎都是 Backend Design

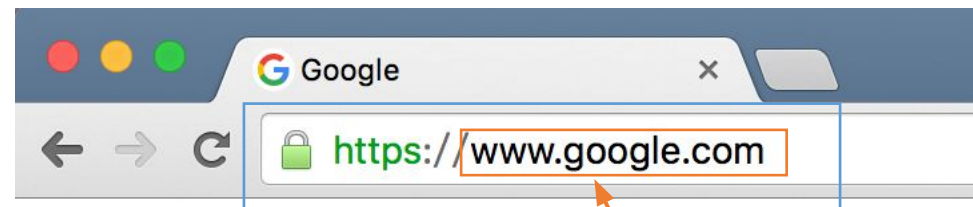
Backend Design 几乎都是 Web Backend Design

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



# 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
  - Python 新宠: Flask
- 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

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

RPS, QPS, IOPS 的大小顺序是?

RPS = Requests Per Second (to web server)

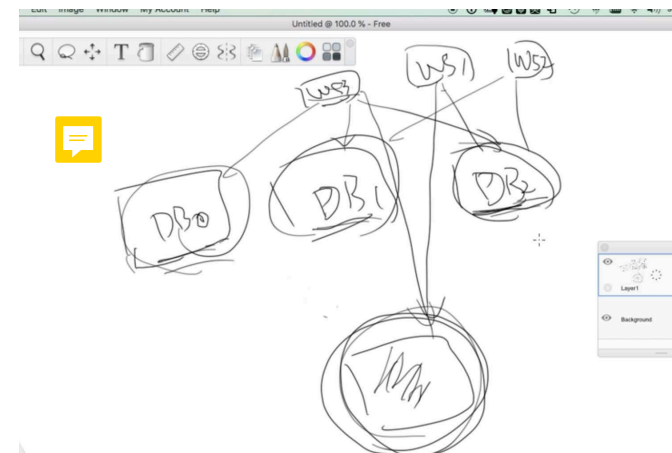
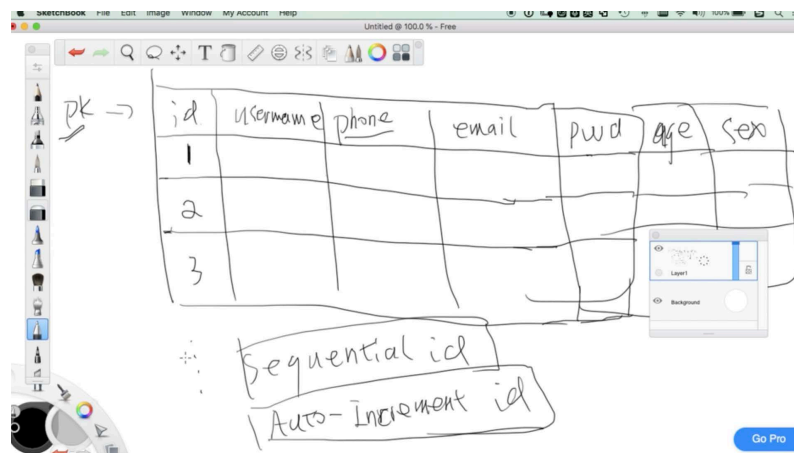
QPS = Queries Per Second (to database)

IOPS = I/O Per Second (to disk)

A: RPS < QPS < IOPS

B: QPS < RPS < IOPS

C: IOPS < RPS < QPS



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:

<https://www.djangoproject.com>

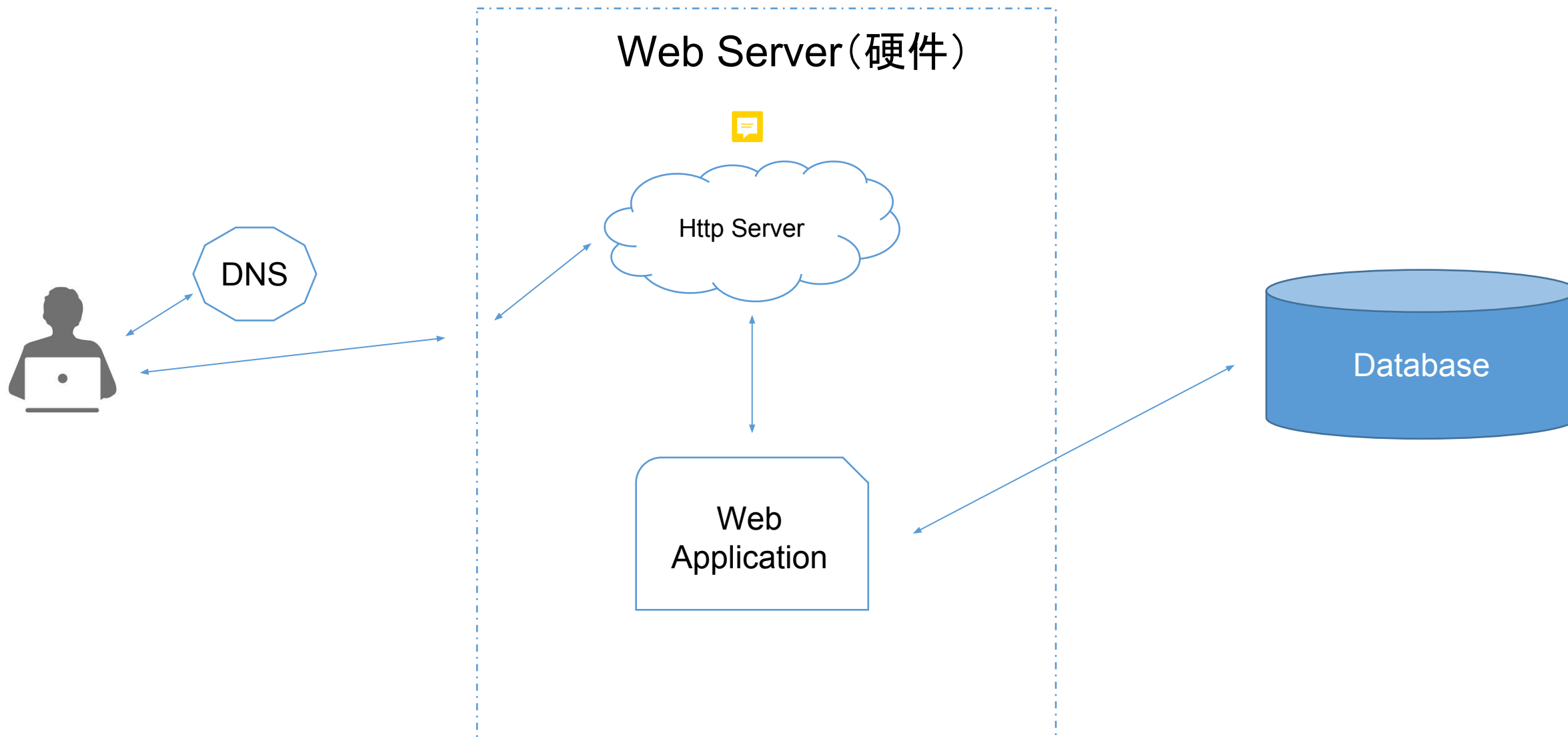


node JS

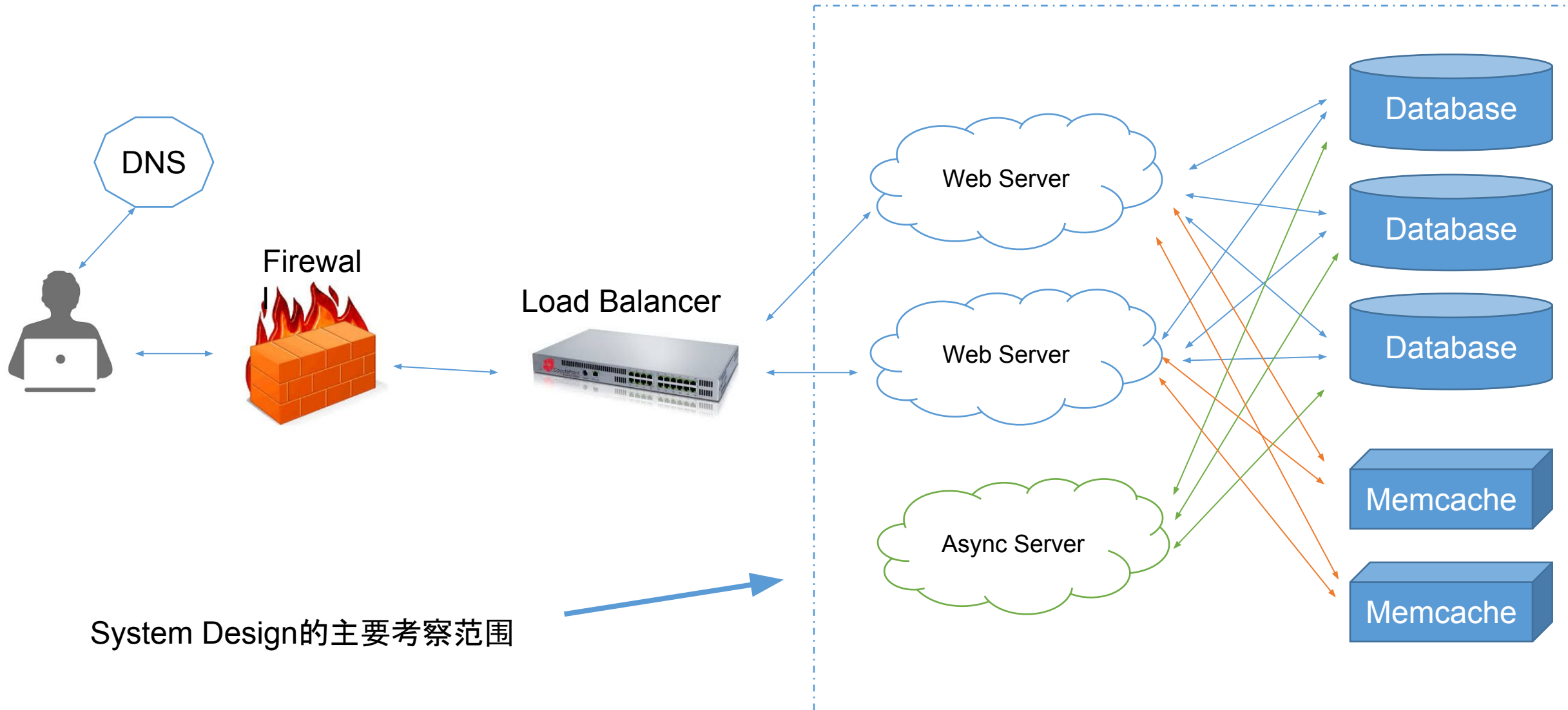


U B E R









System Design的主要考察范围

# API Design

常见的一类系统设计知识点的考察

API = Application Programming Interface

样例1：

你实现了一个叫做 Utility 的 class, 然后这个 class 提供一个 sortIntegers 的 method, 可以对参数中的整数数组进行排序。这就是一个 API。

样例2：

你访问了 <https://www.awebsite.com/api/users/> 获得了所有的用户的信息。这就是一个 Web API

通常来说, 只要是你提供了一些方法, 函数, 功能给别人用, 别人通过直接的函数调用或者 http 等方式进行调用, 得到了返回结果, 这就是 API。

## 练习: 下面哪种 API 的设计更合理

目标: 获得当前登陆用户在 LintCode 上某个题上的所有提交记录

注: 完整的API地址应该为 <https://www.lintcode.com/api/...> 以下简写为 /api/...

A: /api/users/<current\_user\_id>/submissions/?problem\_id=1000



B: /api/users/me/submissions/?problem\_id=1000

C: /api/submissions/?problem\_id=1000&user\_id=<current\_user\_id>

D: /api/submissions/?problem\_id=1000 # 后台用当前用户去筛选

E: /api/problems/1000/submissions/ # 后台用当前用户去筛选

# Rest API

你要获取的数据是什么，路径的主目录就是什么

要获取 problem 就是 /api/problems/

要获取 submission 就是 /api/submissions/

REST = Representational State Transfer = 你不认识他他也不认识你的**某种协议**

满足 REST 协议的架构和设计, 叫做 RESTful

RESTful API 的通俗定义:

- 每个 URL 代表某种类型的一个或者多个数据
  - 如 /api/problems/ 表示得到所有 problems 的数据
  - 如 /api/problems/1/ 表示得到 problem\_id=1 的这个 problem 的数据
- 使用 HTTP 的四个动作 (POST, DELETE, GET, PUT) 来代表对数据的增删查改
- 所有的筛选条件, 创建参数, 都放到 HTTP 的参数里

# REST 练习1: 增删查改某个用户的信息

错误的设计:

`/api/accounts/create/`

`/api/accounts/1/delete/`

`/api/accounts/1/show/`

`/api/accounts/1/update/`

正确的设计:

`POST /api/accounts/`

`DELETE /api/accounts/1/`

`GET /api/accounts/1/`

`PUT /api/accounts/1/`

# REST 练习2: 创建转账记录

错误的设计: `POST /api/accounts/1/transfer/500/to/2/`

正确的设计: `POST /api/transaction/?from=1&to=2&money=500`





# 面试真题：Design News Feed API

与 Design News Feed System 的区别是不需要管后台数据如何存储和获取，只需要进行和前端交互的接口设计

- 什么是 API, 设计获取 News Feeds List 的 Web API 请求格式
- 设计 API 返回的内容格式
- 设计翻页 Pagination
- 设计 Mentions 的数据格式

修改的主体是 Trip, 所以是 `/api/trips/` 开头。

因为是对单个 trip 进行修改, 所以是 `/api/trips/<trip_id>/`, id 放在 URL 路径里而不是参数里。Rest API 的规则里, 被修改的主体的 id 是放在 URL 里的。

最后修改是 PUT, 不是 POST 也不是 GET。

# 面试官提问1:设计 News Feed List 的 Web API 请求格式

GET <https://www.facebook-or-twitter.com/api/newsfeed/>

or

GET <https://api.facebook-or-twitter.com/newsfeed/>

Read request 用 GET

要获取的 Resource 是什么, 一级目录就是什么

## 面试官提问2: 设计API 的返回格式

问: 返回 Structured Data 还是 HTML ?

Structured Data 比如 JSON 或者 XML

目前主流的基于 HTTP 协议的数据格式  
一般都会选择用 JSON

XML 目前主要用于在 Android 的代码中  
记录一些格式化的数据

http://localhost:8080/Json/SyncReply/Contacts

```
{
  - Contacts: [
    - {
      FirstName: "Demis",
      LastName: "Bellot",
      Email: "demis.bellot@gmail.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Jobs",
      Email: "steve@apple.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Ballmer",
      Email: "steve@microsoft.com"
    },
    - {
      FirstName: "Eric",
      LastName: "Schmidt",
      Email: "eric@google.com"
    },
    - {
      FirstName: "Larry",
      LastName: "Ellison",
      Email: "larry@oracle.com"
    }
  ]
}
```

http://localhost:8080/Xml/SyncReply/Contacts

```
<ContactsResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```

## 面试官提问3：如何设计翻页 Pagination？

下面哪种方法是正确的？

方法1：/api/newsfeed/?page=1

方法2：/api/newsfeed/?max\_id=xxx

传统翻页方法: ?page=1

优点: 可以直接跳转到第 x 页

缺点: 如果有新数据被插入时, 翻到下一页可能会看到上一页的内容

适用于用户希望查阅很久以前的内容的使用场景

NewsFeed 是否适用? No

用户获得第一页数据, 得到前3个 Item

-----page1-----

item0

item1

item2

-----page2-----

item3

item4

item5

用户点击下一页之前新数据被加入:

-----page1-----

new-item

item0

item1

-----page2-----

item2

item3

item4

用户此时再请求  
第二页的数据,  
会看到重复 item



News Feed 一般不会采用页码进行翻页  
而是采用无限翻页 Endless Pagination

API 设计:

`/api/newsfeed/?max_id=1000` 

没有 max\_id 表示第一页

有 max\_id 找到所有  $id \leq max\_id$  的最顶上的一页的数据(假设倒序)





## 面试官提问3.1: 如何判定有没有下一页?

当 response 里什么数据都没有时表示没有下一页

问: 这种方法有什么问题?

正确方法：

每次多取一个数据，如果取到，把这个数据作为 next\_max\_id 返回给前端

PAGE\_SIZE = 20

从 request 中获取 max\_id

if 没有 max\_id

    获得最新的 PAGE\_SIZE + 1 条数据

else

    获得 id <= max\_id 的最新的 PAGE\_SIZE + 1 条数据

如果获取到的数据为 21 条，将第21条的 id 赋值给 next\_max\_id

否则 next\_max\_id = null

返回数据：

```
{
  'next_max_id': next_max_id,
  'items': [...最多 PAGE_SIZE 条数据]
}
```

【系统】如果 Post 存储在 NoSQL 中，没有 Sequential ID，还可以使用光标翻页法么？

Sequential ID 是递增的，可以通过 max\_id 和 min\_id 来筛选。但是如果在 NoSQL 中，ID 是 UUID，这个是完全无序的字符串，并不是后发的帖子 UUID 就会比先发的帖子 UUID 大。

A: 用不了

B: 可以用，用 timestamp 之类的有序信息翻页即可

## 面试官提问3.2: 如何获得更新内容

GET /api/newsfeed/?min\_id=<id>

找到所有 id > 目前客户端里最新的一条帖子的 id 的所有帖子

## 面试官提问4:设计 Mentions 的数据格式

当 News Feed 的文本内容里有 Mentions 的时候,即提到了某个人  
该如何在文本内容中体现并让前端可以显示成一个带 link 的样式?

## 方法1: 在文本中直接返回 html 格式的链接

Hello world, this is the first post from  
<a href="/users/someone/">@someone</a>  
please follow me!

缺点？

## 缺点1: 需要预防 javascript injection attack

举例: 人人网曾经遭受过一次 javascript injection attack, 看了某个帖子之后, 自动就分享出去了

## 缺点2: API 无法被 Mobile 端共享

Mobile 显示链接并不是用 `<a>`

## 缺点3: URL 可能会改动

如果将链接内容作为 post 的一部分存入数据库中, 以后如果链接发生了变化, 如从 `/users/<id>/` 改成了 `/users/<username>/` 那么旧的链接无法被马上弃用 (因为已经存在数据库里了, 改起来太费劲, 需要遍历所有的数据), 从而导致有安全隐患设计的 URL `/users/<id>/` 无法被弃用。

# 推荐方法：自定义链接结构

如 `<user username="someone">Hello World</user>`

让 Web 和 Mobile 分别对该格式进行解析

解析成各自平台认识的格式

# 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/>

九章算法，帮助更多中国人找到好工作

硅谷精英在线面试技巧、书籍算法、数据结构、系统设计等必备知识

► 查看全部课程

会员

立即购买

# 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

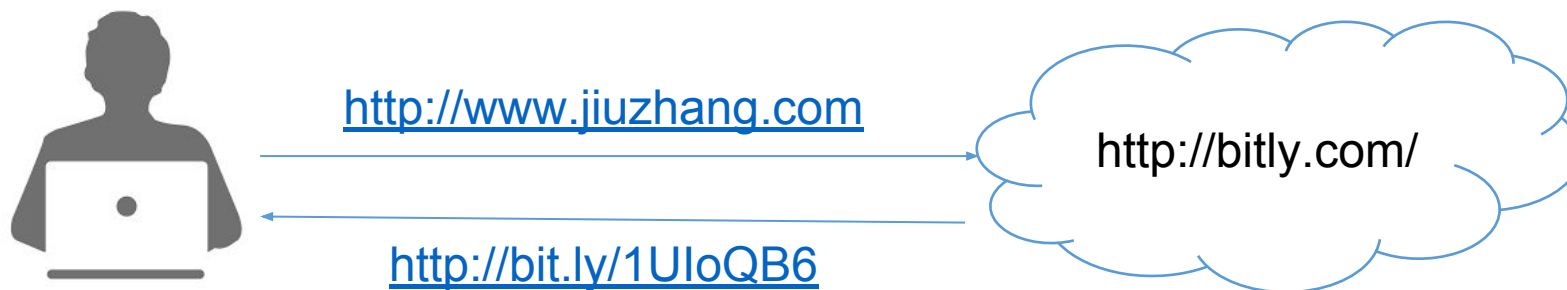
<http://loooooooooong.url>

<http://short.url>

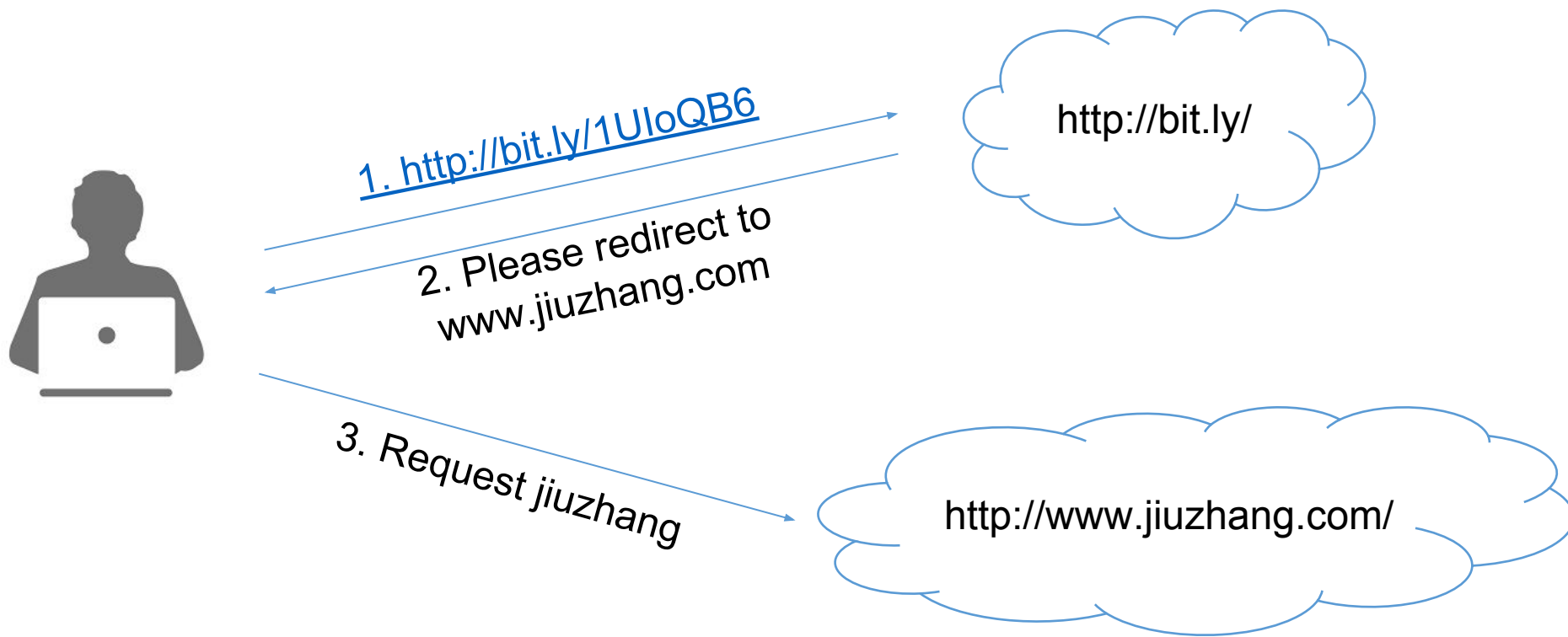


动起手来试试看

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



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



# 两个需要和面试官确认的问题

Long Url 和 Short Url 之间必须是一一对应的关系么？ 

Short Url 长时间没人用需要释放么？ 

不同的要求设计出来的系统完全不一样！

# 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

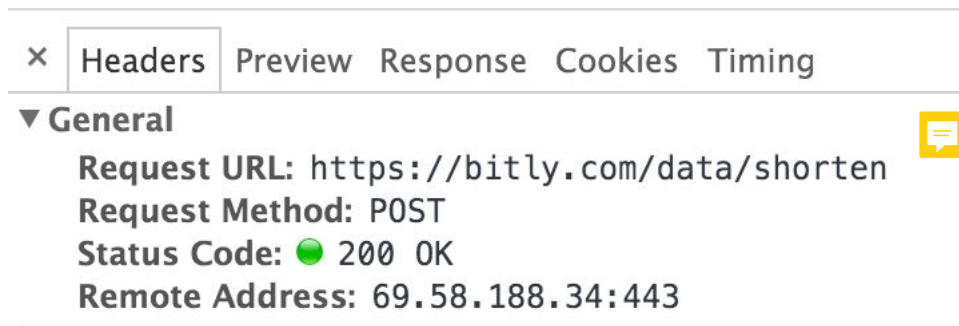
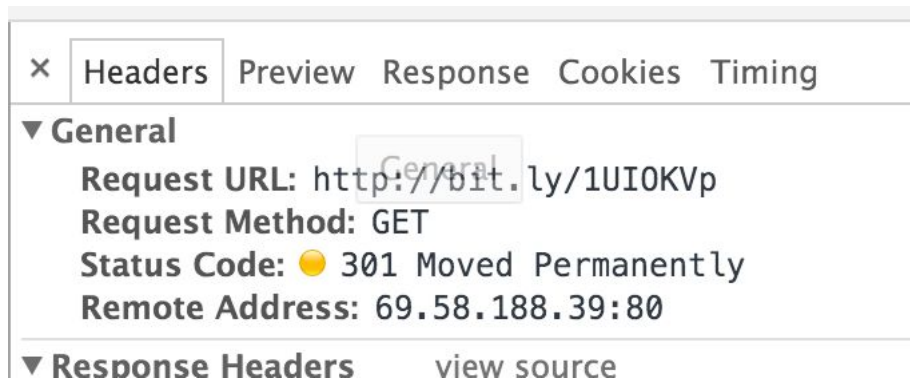
2k QPS

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



# 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？——不需要。NoSQL +1
  - NoSQL不支持Transaction
- 是否需要丰富的 SQL Query？——不需要。NoSQL +1
  - 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？——Tiny URL 需要写的代码并不复杂。NoSQL+1
  - 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

- 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
  - NoSQL 的性能更高
- 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
  - 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
- 是否需要Sequential ID？—— 取决于你用什么算法
  - 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



所以Tiny URL用什么比较合适？

# 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

- 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

[多选题] 【系统】下面哪些“码”可能是随机生成+数据库去重的?


- |            |           |
|------------|-----------|
| A. 短信验证码   | 20.80% 选择 |
| B. 邮箱激活码   | 21.03% 选择 |
| C. 酒店订单确认码 | 29.17% 选择 |
| D. 机票订单确认码 | 29.00% 选择 |

✖ 答错了, 您选择的答案是 A

正确答案: C D

解析: 验证码和激活码没有去重的需求。订单确认码才有去重的需求。

关闭

- 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



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>

SQL table的索引是不是越多越好, 假如我给每个column都建立索引有什么弊端吗

不是越多越好, 创建和维护索引组要耗费时间, 并且随着数据量的增加所耗费的时间也会增加。

索引需要占磁盘空间, 除了数据表占数据空间以外, 每一个索引还要占一定的物理空间。如果有大量的索引, 索引文件可能比数据文件更快达到最大文件尺寸。

当对表中的数据进行增加、删除和修改的时候, 索引也要动态维护, 这样就降低了数据的维护速度。

在cassandra中, row key和index有什么区别? row key是指定去那台database server上找, 有点类似index, 但是两者完全等价吗?

"索引是关系数据库中对某一列或多个列的值进行预排序的数据结构。通过使用索引, 可以让数据库系统不必扫描整个表, 而是直接定位到符合条件的记录, 这样就大大加快了查询速度。

cassandra找到value需要row key和column key一起才能确定, row key算是这里索引的一部分。

肯定不等价的, 一个是nosql型数据库一个是sql型数据库。"

nosql和mysql的索引怎么建? 是有一个特殊的代码去建吗?

是的, MySQL 有创建索引的语句:

```
CREATE INDEX indexName ON table_name (column_name)
```

通常所有的关系型数据库都支持索引。

NoSQL 通常是不支持索引的, 以 Redis 为例就不支持索引, 但是可以用其他方式自己维护一个索引, 具体方式可以参考。

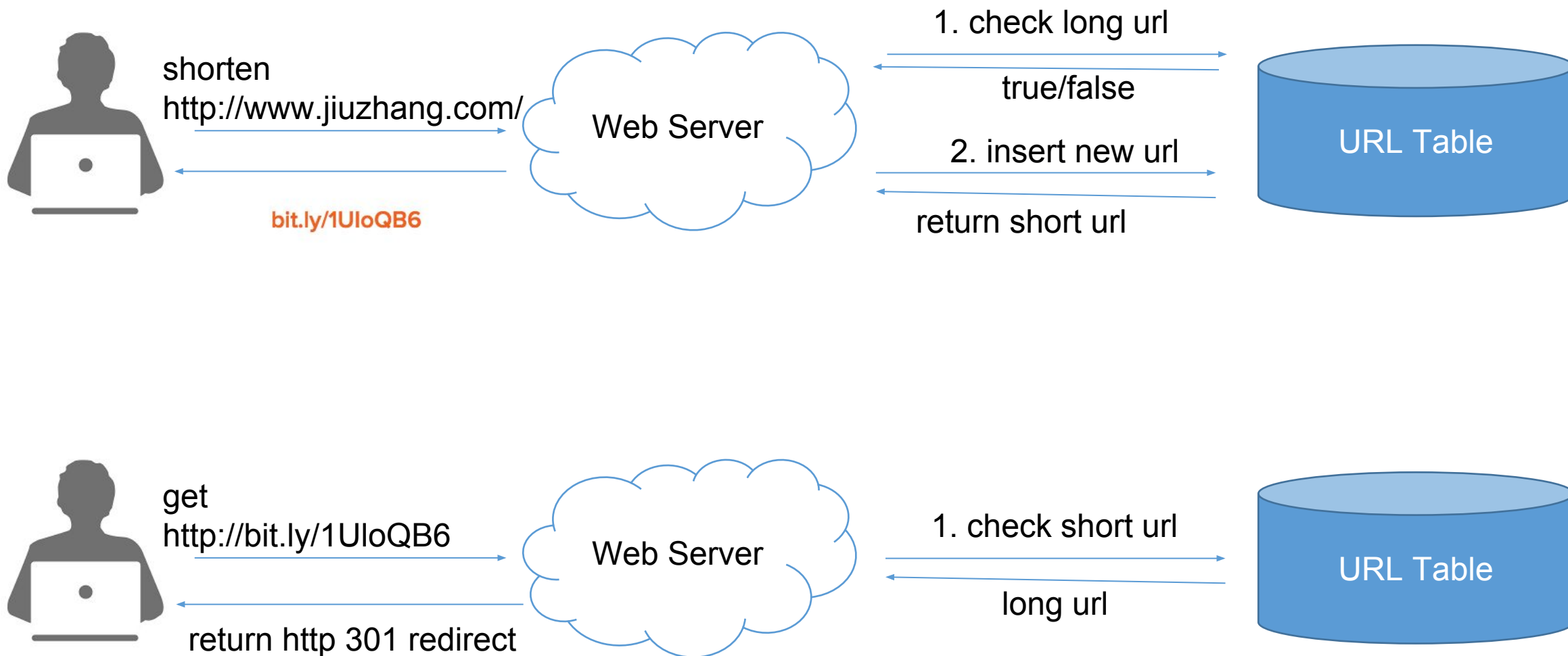
也可以选用 NoSQL 数据库, 但是需要建立两张表(大多数 NoSQL 不支持多重索引 Multi-index)  
以 Cassandra 为例子

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

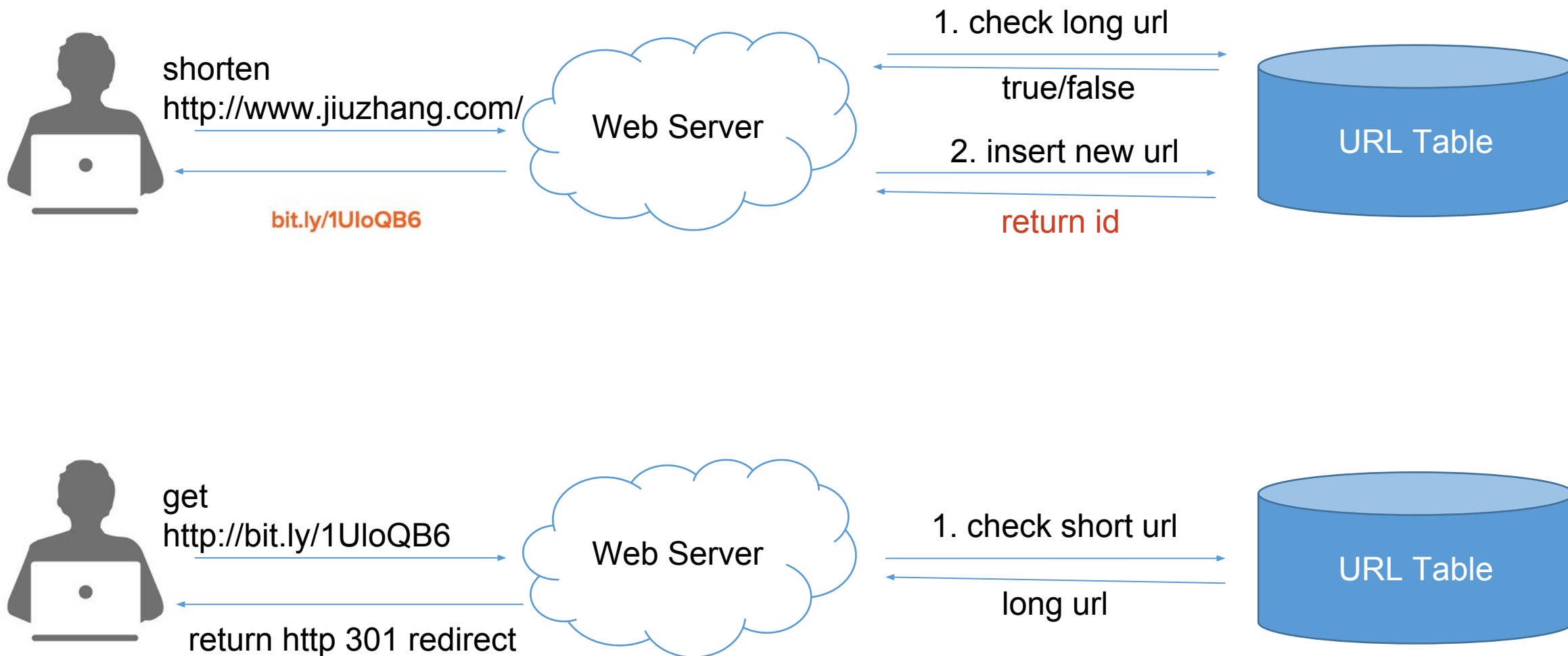


因为需要用到自增ID (Sequential ID), 因此只能选择使用 SQL 型数据库。

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	<a href="http://www.jiuzhang.com/">http://www.jiuzhang.com/</a>
2	<a href="http://www.lintcode.com/">http://www.lintcode.com/</a>
3	<a href="http://www.google.com/">http://www.google.com/</a>
4	<a href="http://www.facebook.com/">http://www.facebook.com/</a>





# 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？

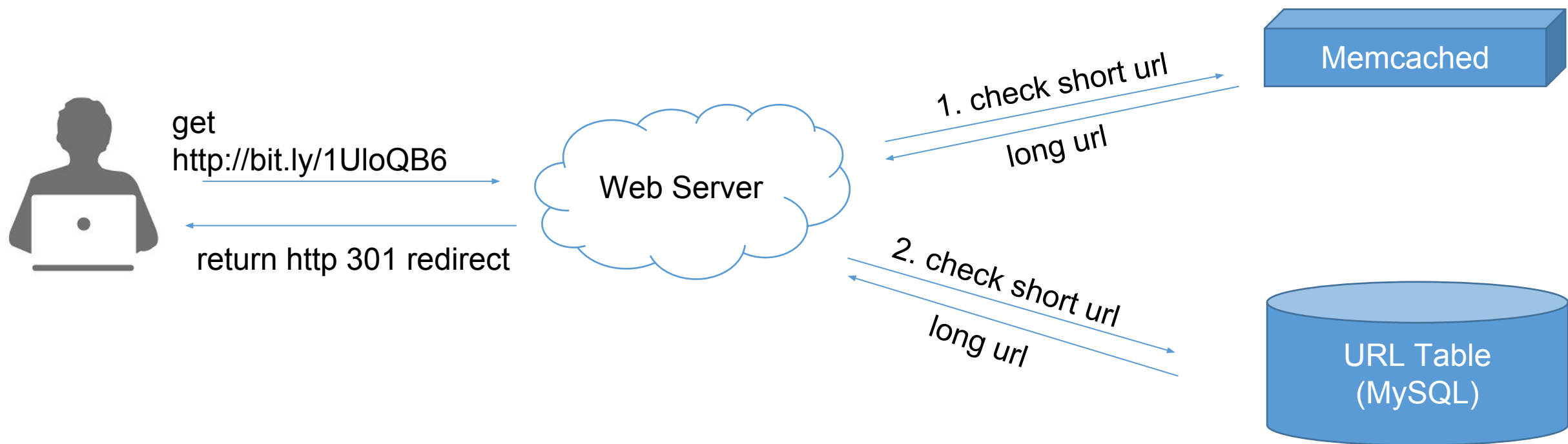


说说看有哪些地方可以加速？



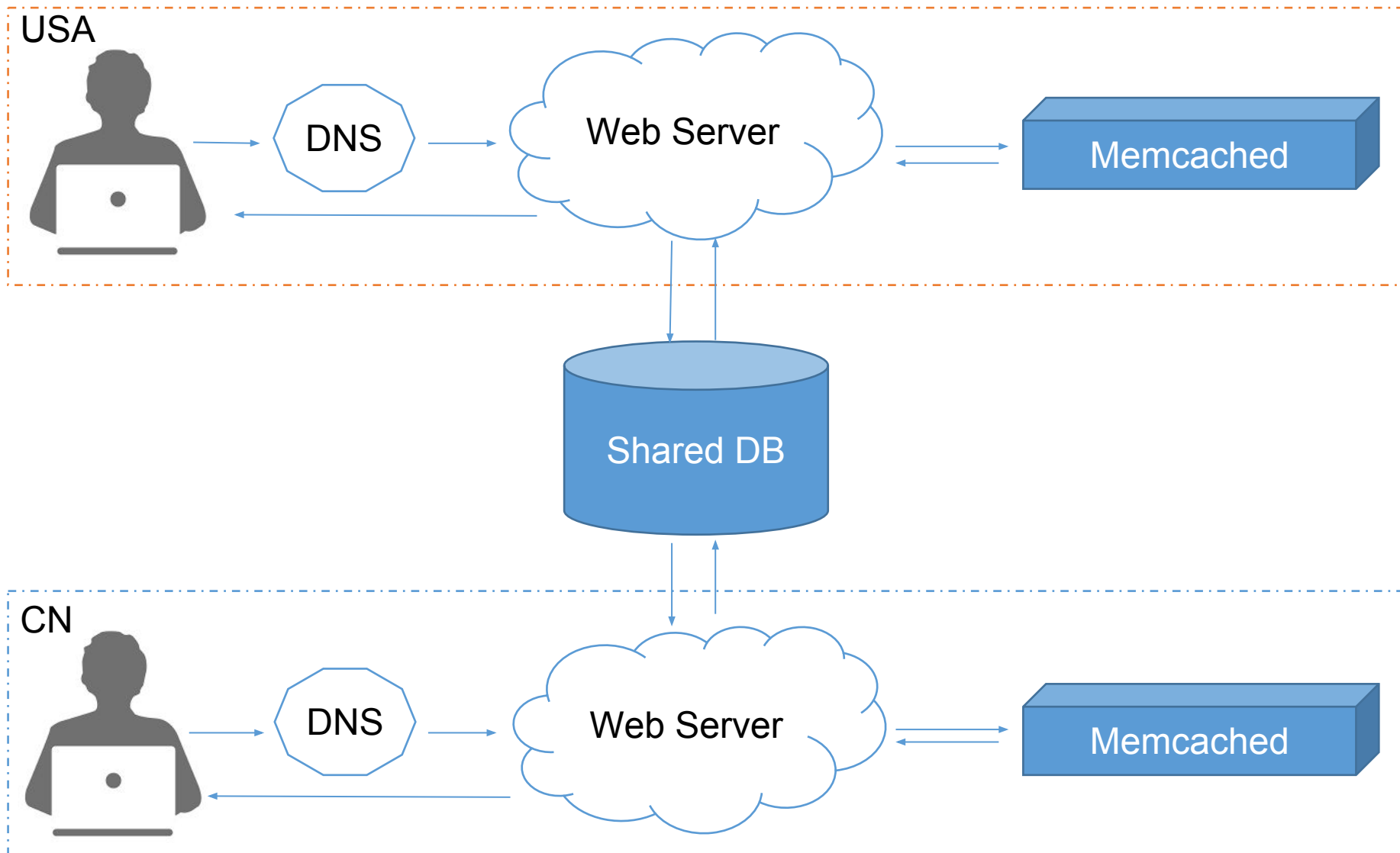
提高读的速度还是写的速度？

- 利用缓存提速 (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



- 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

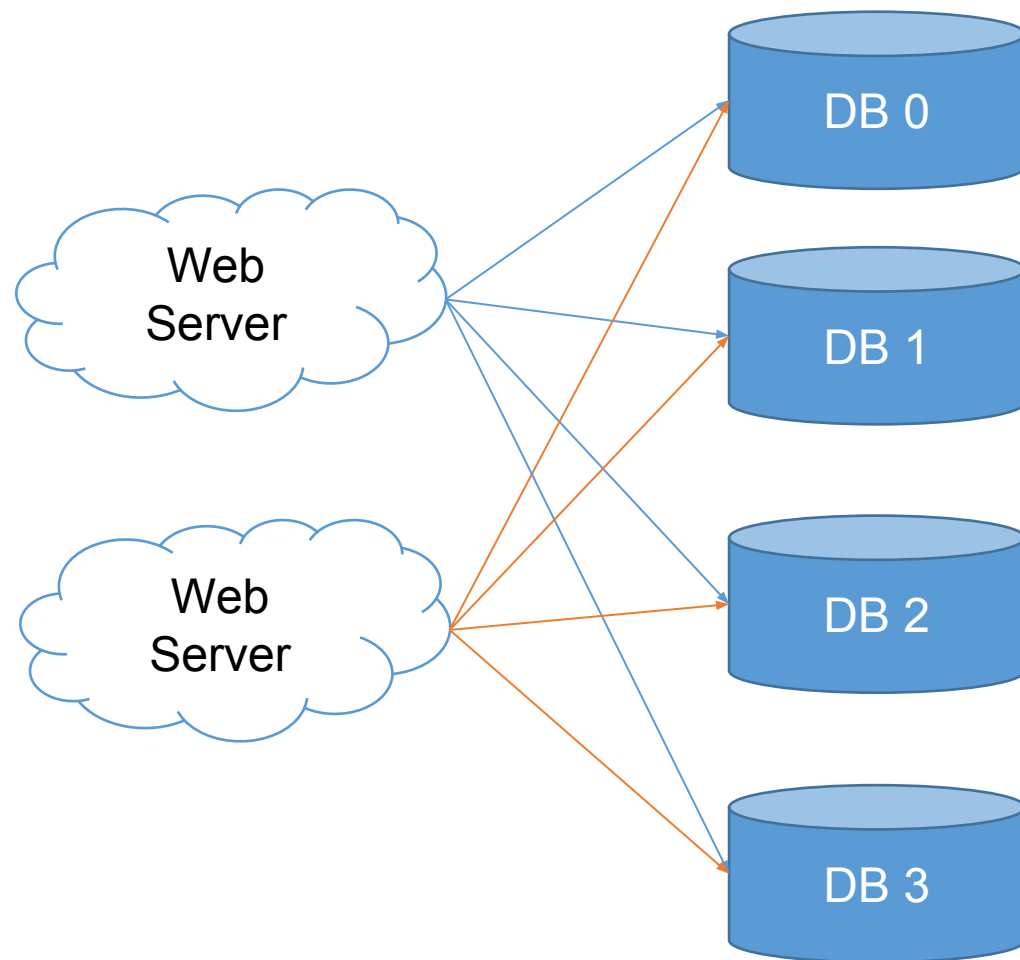




- 什么时候需要多台数据库服务器？
  - Cache 资源不够
  - 写操作越来越多
  - 越来越多的请求无法通过 Cache 满足
- 增加多台数据库服务器可以优化什么？
  - 解决“存不下”的问题 —— Storage的角度
  - 解决“忙不过”的问题 —— QPS的角度
  - Tiny URL 主要是什么问题？ 



如何将数据分配到多台机器？

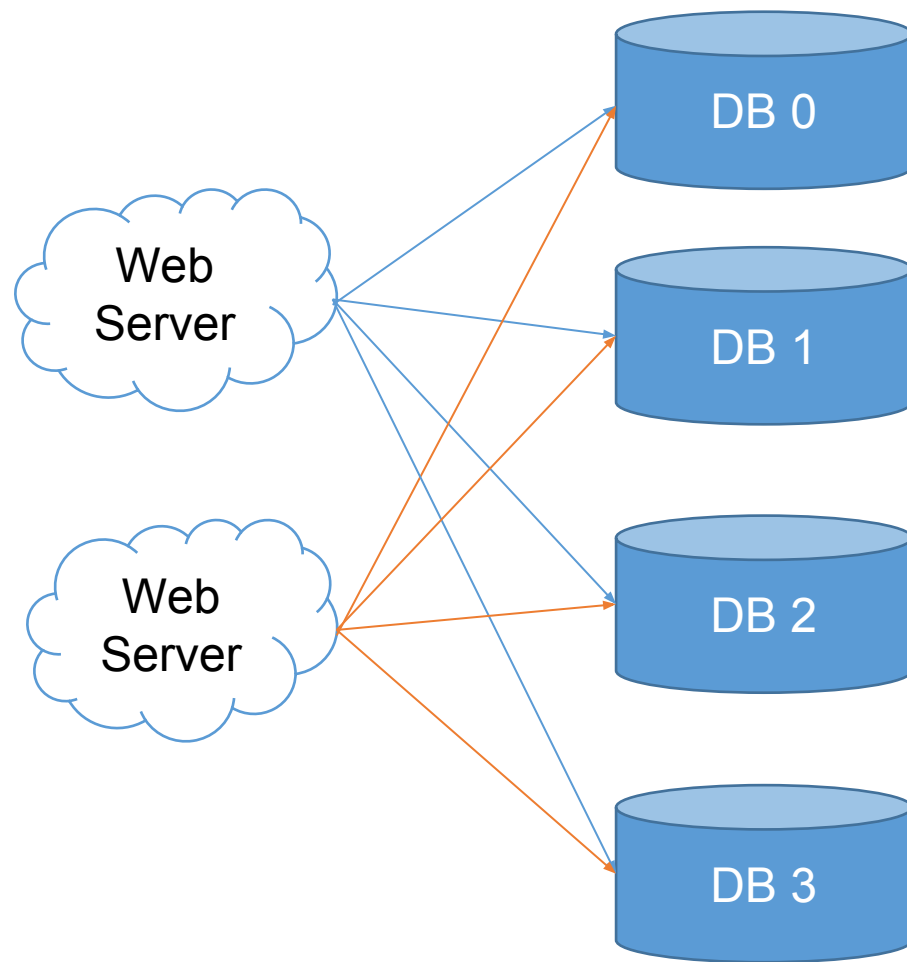


## Scale —— 如何扩展？

- Vertical Sharding —— 将多张数据表分别分配给多台机器
  - Tiny URL 并不适用, 只有两个 column 无法再拆分
- Horizontal Sharding
  - 如果用 ID / ShortUrl 做Sharding Key
    - 做 long to short 查询的时候, 只能广播给 N 台数据库查询
    - 为什么会需要查 long to short? 创建的时候避免重复创建
    - 如果不需要避免重复创建, 则这样可行
  - 如果用 Long Url 做Sharding Key
    - 做 short to long 查询的时候, 只能广播给 N 台数据库查询



用什么做 Sharding Key?



- 如果一个 Long 可以对应多个 Short:
  - 使用 Cache 缓存所有的 Long to Short
  - 在为一个 Long Url 创建 Short Url 的时候, 如果 cache miss 则去直接创建新的 short url 即可
- 如果一个 Long 只能对应一个 Short:
  - 如果使用随机生成 Short Url 的算法
    - 两张表单, 一张存储 Long to Short, 一张存储 Short to Long
    - 每个映射关系存两份, 则可以同时支持 long to short 和 short to long 的查询
  - 如果使用 base62 的进制转换法
    - 这里有一个很严重的问题是, 多台机器之间如何维护一个全局自增的 ID?
    - 一般关系型数据库只支持在一台机器上实现这台机器上全局自增的 ID





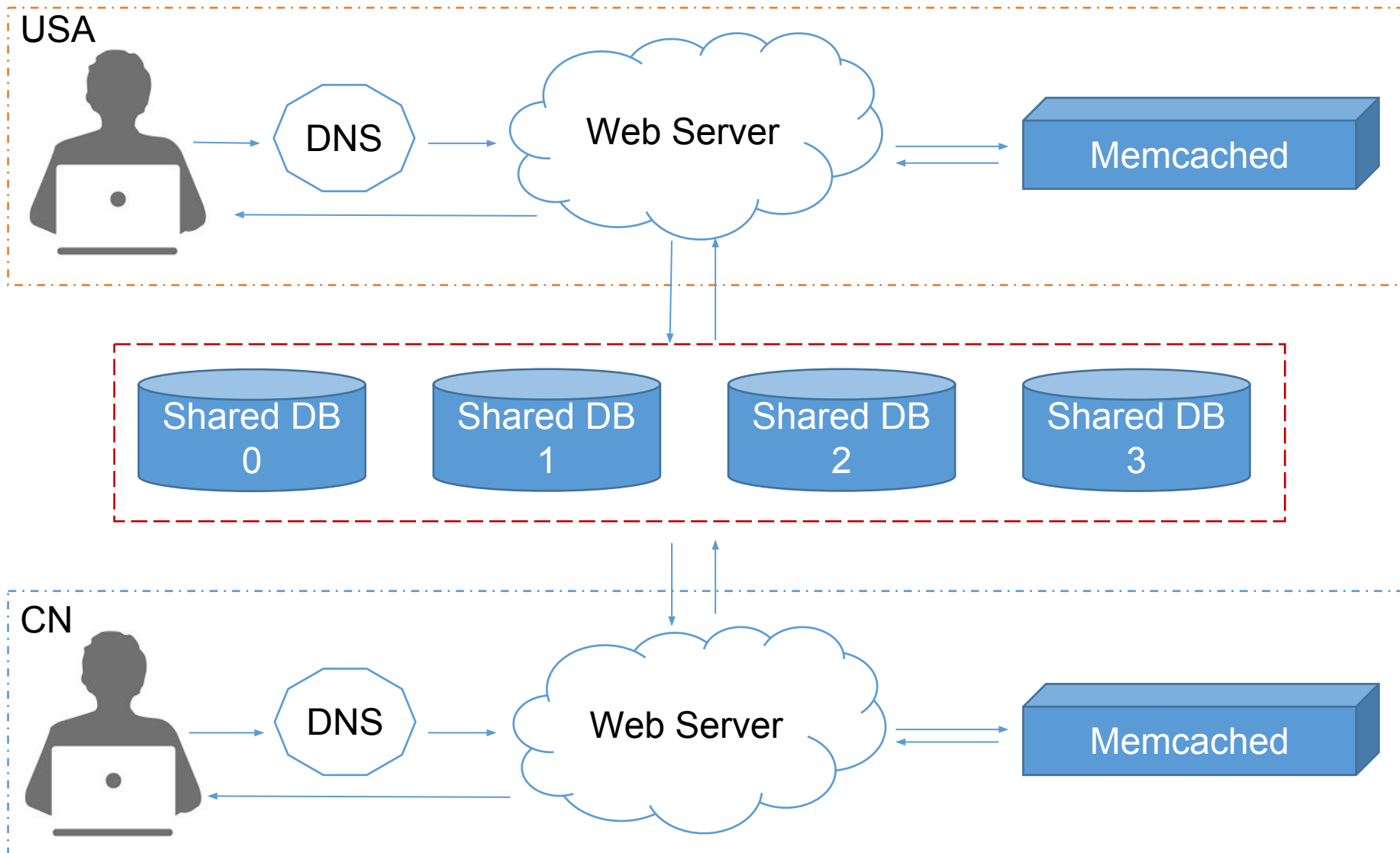
卧槽, 那到底怎么做?



- 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
  - 但是使用全局自增ID的方法并不是解决 Tiny URL 的最佳方法
  - 有兴趣的同学可以阅读一下拓展资料: <http://bit.ly/1RCyPsy>

## 基于 base62 的方法下的 Sharding key 策略


- 使用  $\text{Hash}(\text{long\_url}) \% 62$  作为 Sharding key 
- 并将  $\text{Hash}(\text{long\_url}) \% 62$  直接放到 short url 里
- 如果原来的 short key 是 AB1234 的话, 现在的 short key 是
  - $\text{hash}(\text{long\_url}) \% 62 + \text{AB1234}$
  - 如果  $\text{hash}(\text{long\_url}) \% 62 = 0$  那就是 0AB1234
- 这样我们就可以同时通过 short\_url 和 long\_url 得到 Sharding Key
- 缺点: 数据库的机器数目不能超过 62 



\* Interviewer: 还有可以优化的么？



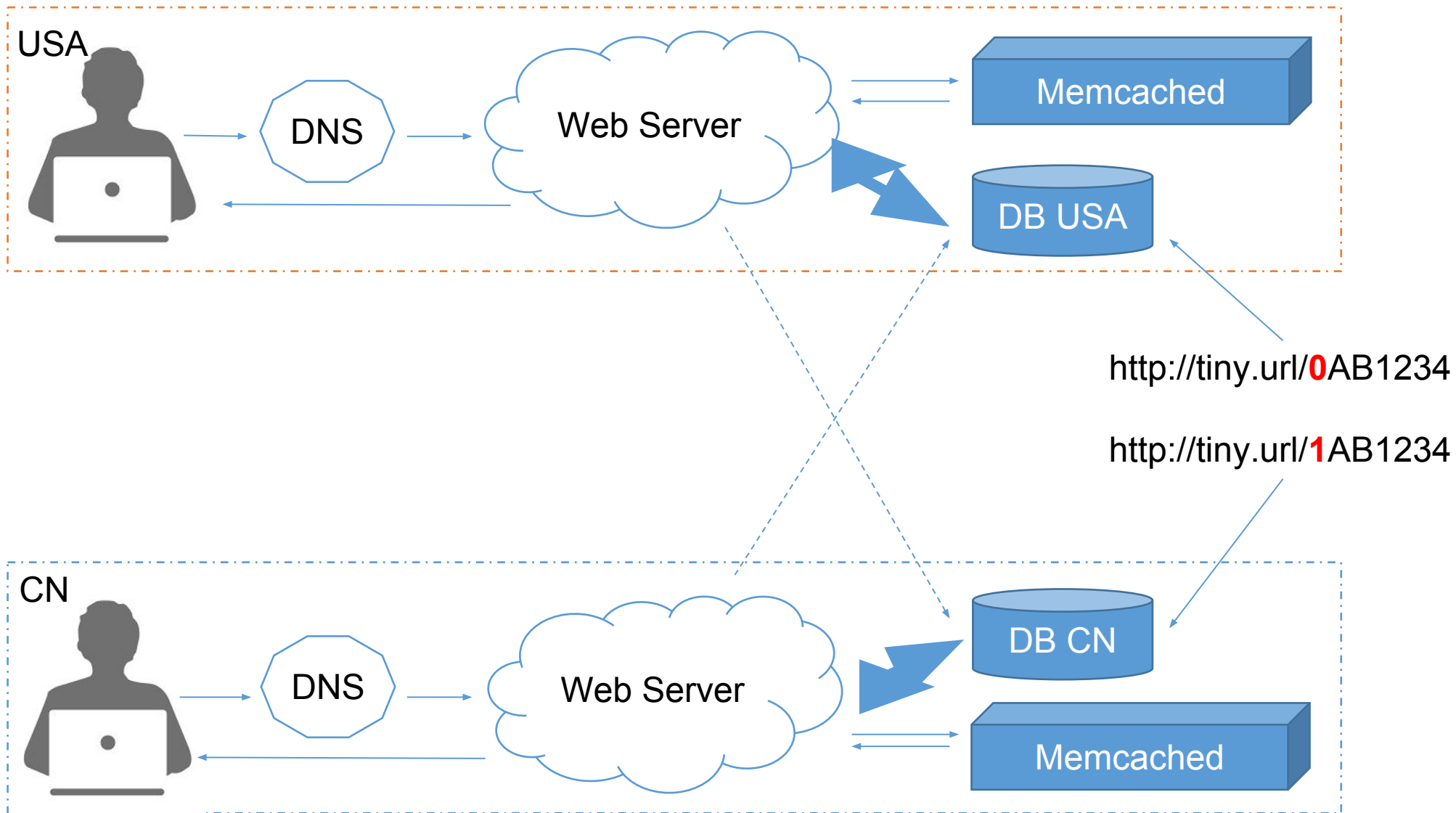
NI GE SHA BI  
你他吗有完没完

- 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

DNS是一个数据库吗？还是一个service？一般谁负责set up 呢？

"dns就是域名解析服务  
可以理解为一个service  
用户购买域名的时候 服务商就会更新这个域名数据库"





# 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

编程题：

[www.lintcode.com/problem/tiny-url](http://www.lintcode.com/problem/tiny-url)

[www.lintcode.com/problem/tiny-url-ii](http://www.lintcode.com/problem/tiny-url-ii)

打车软件中司机点击接到乘客，服务器将 Trip 的状态修改为进行中，正确的 Rest API 设计是？

请分别选出正确的 API URL 和对应的请求方法。

A: /api/start\_trip/?trip\_id=123

B: /api/trips/start/?id=123

C: /api/trips/123/?action=start

D: /api/trips/?id=123&action=start

E: 使用 GET 请求

F: 使用 POST 请求

G: 使用 PUT 请求

修改的主体是 Trip，所以是 `/api/trips/` 开头。

因为是对单个 trip 进行修改，所以是 `/api/trips/<trip_id>/`，id 放在 URL 路径里而不是参数里。Rest API 的规则里，被修改的主体的 id 是放在 URL 里的。

最后修改是 PUT，不是 POST 也不是 GET。

- 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



## 拓展练习: 实现 Custom URL

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

## Custom URL - 基于 Base62

问: 在 URLTable 里直接新增一项 custom url 记录对应的 custom url 是否可行?

## 实现 Custom URL - 基于 Base62 的方法

- URL Table 增加一个 Column 不可行
  - 因为大部分数据这一项都是空的
  - 会浪费存储空间 (即使是空也会有空间浪费)

- 新建一张表存储自定义 URL

- CustomURLTable

- 创新自定义短链接

- 在 CustomURLTable 中查询和插入

- 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

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

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

## Custom URL - 基于随机生成

无需做任何改动, 直接把 custom url 当 short url 创建就可以了!  
完全兼容!

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

