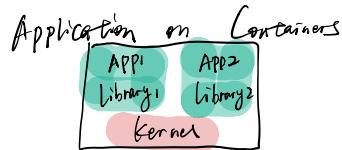


Docker

1. Docker 是一个容器化平台，它将应用程序及其所有依赖项以容器的形式打包在一起。
2. Dockerfile
 - ① ADD / COPY
 ADD / COPY <src> ... <dest>
 ADD指令 ① 可以让你使用 URL 作为 <src> 参数，另外一种形式是指定下载目录。
 ② 有能力自动解压文件。ADD (foo.tar.gz /tmp)
 直接解压到 /tmp！
 ③ 编写 Dockerfile 文件； docker build 命令 image； docker run 容器运行。
 指令大多从以下执行，除非指令会创建镜像层。
 ④ EXPOSE 暴露端口。
 ENV 环境变量
 VOLUME 存储卷（保存和持久化）
 CMD 启动后运行时要运行的命令。如果有多个，但只有最后一个有效。CMD会被run之后
 无法被替换。
 ⑤ WORKDIR 绝对路径认为是工作目录（菰脚本）
 ⑥ CMD ["top"] 不会通过 shell 执行相关命令。因为 \$HOME
 这种环境变量又不同
 ⑦ 指定进程序列
 ⑧ CMD ["sh", "-c", "echo \$HOME"]
 指定 shell 可以执行
 ⑨ FROM ubuntu
 ENTRYPOINT ["top", "-b"]
 docker run tester -c=and[c]
 ; top -b -c !
 2. Shell CMD top docker会执行 /bin/bash -c "task command"
 执行。先进入到 bash 进程。
 3. CMD ["param1", "param2"] 为 Entry Point 提供参数。
 ENTRYPOINT exec 模式。命令行字符串参数会添加到参数列表中。

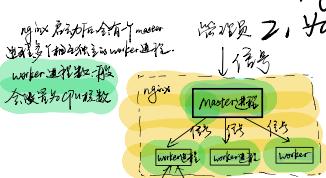
3. docker-compose 帮助同时管理多个容器。 docker-compose.yml 配置。

4. Application on Host



Nginx

1. Nginx 是一款轻量级高性能的反向代理 web 服务器，可以处理 2-3 万并发连接数。



2. 为什么用 Nginx：

- ① 跨平台，配置简单，高并发，内存消耗少（10个 Nginx 才占 150M 内存）
- ② 加强健康检查功能（如果有 1 个服务器宕机，系统不会触发到强杀掉）
- ③ 支持 GZIP 压缩，稳定性高。
- ④ 解决跨域问题

3) 跨域方法：

- JSONP
- document.domain 属性
- windows.name
- 使用代理服务器。eg. Nginx 如果请求不是访问本地，就不在代理了。

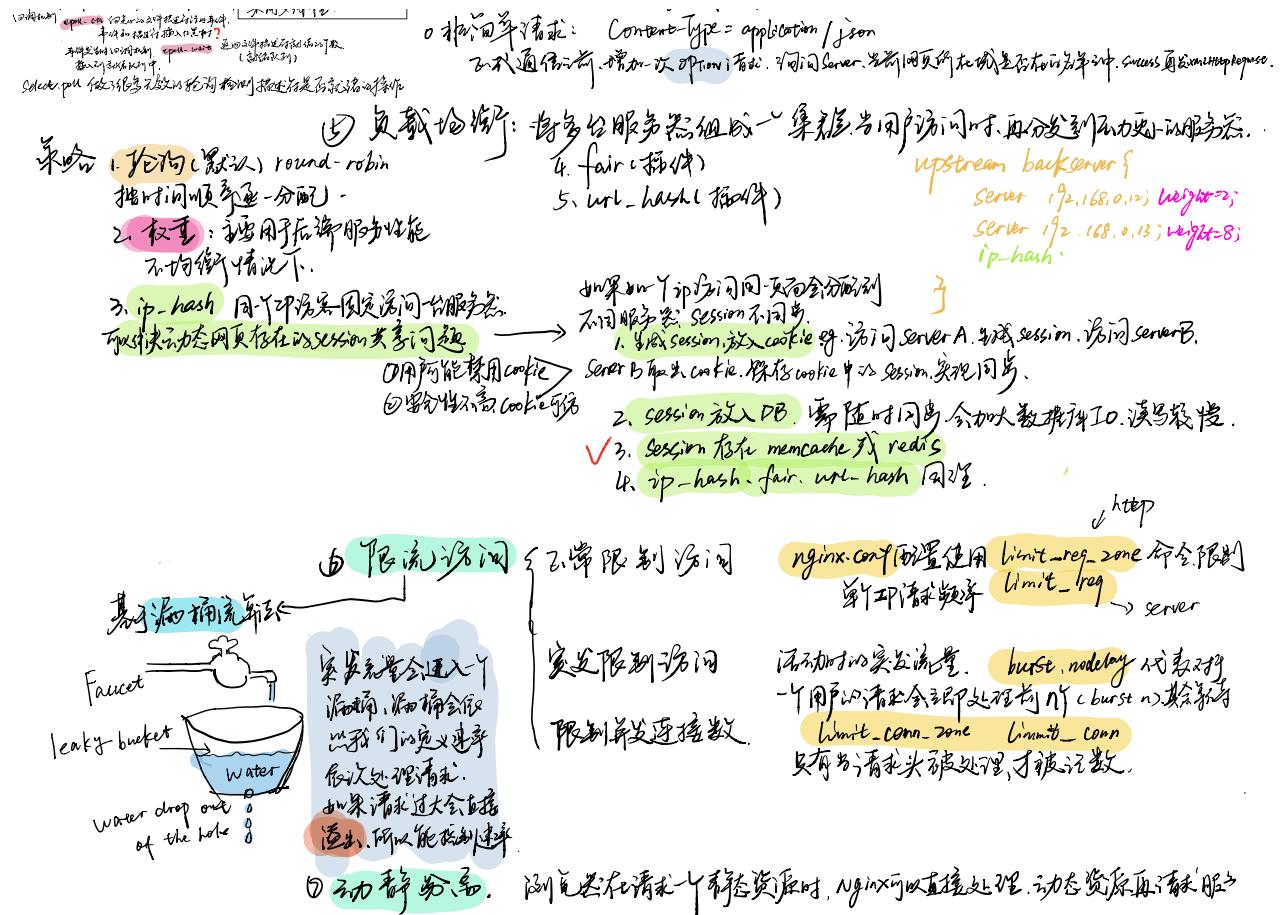
Nginx: ProxyPass /api http://localhost:8081/api

nginx 采用多进程 + 10 多线程模型（CPU 模型）提高并发。

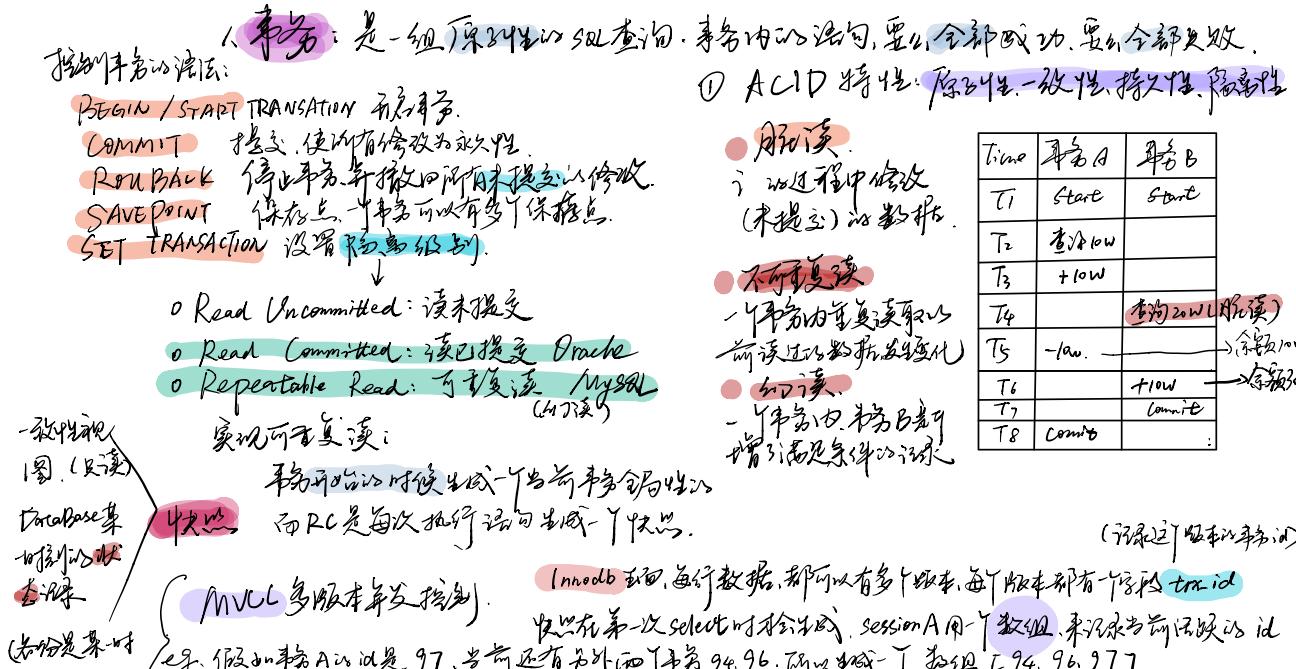
epoll 模型
epoll_create() (网络连接事件)
select(2) (文件读写事件)

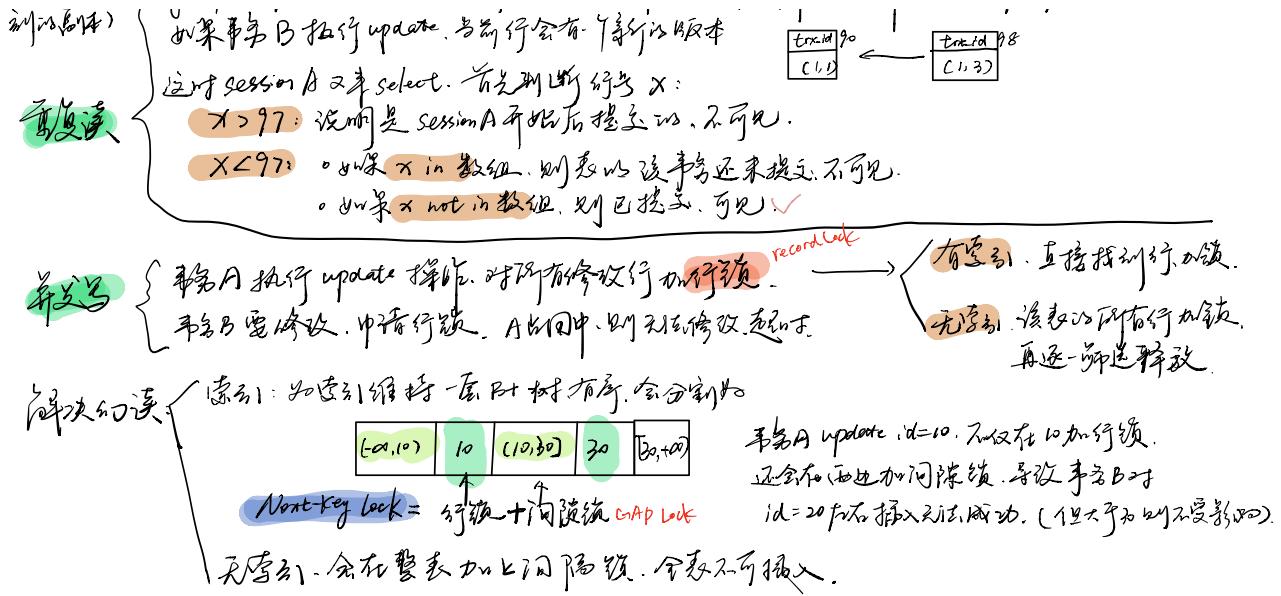
CORS (Cross-Origin-Resource-Sharing)

① 同源策略：① HEAD, GET, POST
② Content-Type: form, formdata, text/plain
access-control-allow-origin: *

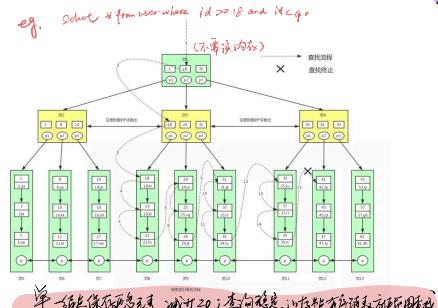
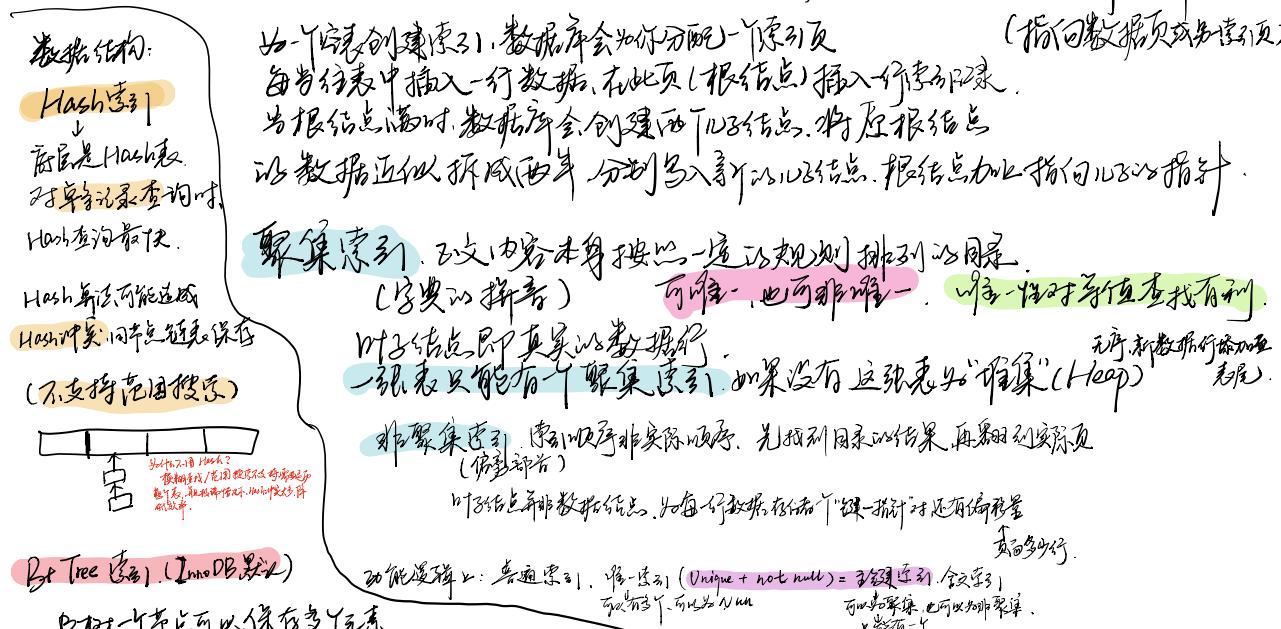


DataBase



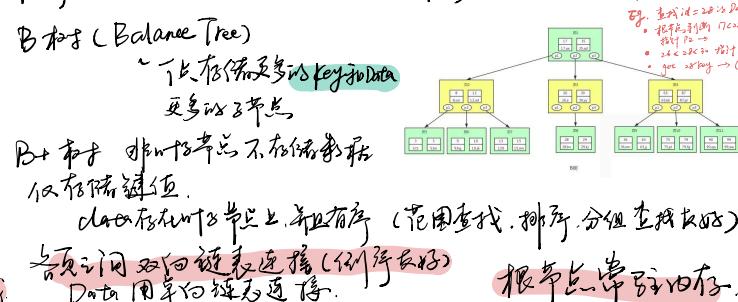


2. 索引 包括键值（定义索引时指定的所有字段的值）+ 逻辑指针



二叉树 (BST)

$root.left < root \& root < root.right$



索引设计原则： **最长匹配原则**: 创建组合索引，以最左边为准，只要条件中带有最左边的列就该用到。
Key(a,b,c) a, ab, abc 有效。ac, bc 不有效
把单列的非主键索引，修改为多字段的联合索引，在一棵树状有分支，这样回表 覆盖索引

原则设计原则：

- 增长原则
- 索引以查询条件
- 更新频繁 不适合索引
- 不能有空值的列不适合（如性别）
- 全量扫描，而不是过滤
- 索引值很长会增加速度（使用前缀）

查询优化器

一些语句可能不会命中索引。
 全局优化器会根据语句的逻辑。
 (选择率,代价)

原则类比: 检查SQL是否使用了索引。EXPLAIN

- Like % 和 大于号
- OR 不好
- 组合索引不是第一个时无效(部分)
- 远列例 IS NULL 或者 is not null. 类似
- not, <, !=, 全部扫描。
- 当分表扫描连接大于等于时。

3. 锁

InnoDB 的锁:

对行加锁，如果还有索引

优化器调用锁

扫描，叫全表锁

开销大，加锁慢，并发性能低

行锁

InnoDB

表锁

锁的粒度划分

页锁

MyISAM 半物理化操作。

锁的使用方式划分

共享锁(s锁) 读锁不能写。

排它锁(x锁)

锁种类

Mysql 的锁机制实现，悲观锁

自己实现，比较耗时

乐观锁

Session 1	Session 2
update t Set num=1 when id=2	
insert into t values (1,2)	update t Set num=2 when id=2
get id=2	get id=2 when id=2
wait	get id=1 release
wait	wait

GAP Lock: 当递增唯一性字节不会加。
 Select * from employee where num>=3 for update;
 num to num+1, num num>=3, [1,3] 范围会+GAP Lock

不再需要该表，drop DDL，记得先

保留表，删除视表 truncate 不可回收。

删除部分记录 (always with a where clause) 需要显式 commit 操作

4. 元数据

对存储数据的管理
 和完整性校验。
 不能为NULL。

对外部

在一级中存储
 第一级没有锁
 (共享, 可以NULL)

5. Char

速度快，效率高，一般用于存储

定长度数据：身份证号、电话。

(不满足时，会在右边填充空格)

Varchar

不稳定 (0~255)

(会多加一个字符串长度，如果超过255，则需要另外两个字节)

6. 异步复制

用来建立一个和主数据库完全一样的数据环境。写-Master，读-Slave

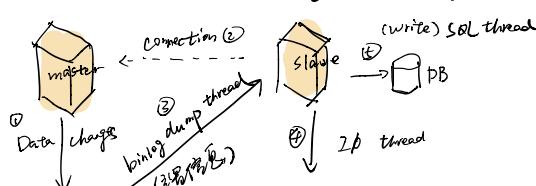
作用

- 所备DB，主数据库故障后，可切换从库继续工作。
- I/O访问频度过高，单机无法满足
- 读写分离，支持更大并发，如报表SQL慢，易锁表。
- 前台服务用master，报表用slave。

读远写近，且读对数据实时性不高，使用M-S，否则会有延时。
 对于需要实时读取的数据，比如用户修改商品，直接从Master读

原理

- 数据流是 bin-log 由 MySQL 写入
- 把 master 的文件 SQL 是写出来
- 从 slave 读 relay-log，再执行

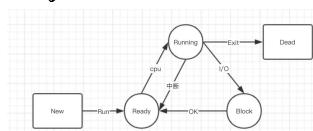




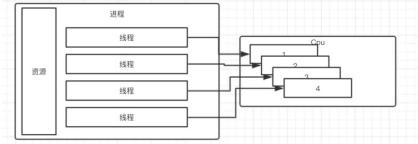
Python

1. 线程、进程、协程。

进程：后台独立运行的程序，占有操作系统分配资源，互不干涉。
多个线程在一下进程中，共享进程资源。



并行：每个CPU核只有一线程在运行
线程 → 4个线程在并发



同步阻塞：有一个线程较长的IO操作，一直等待完成，无法进行其他操作。
非阻塞：有一个轮询查看是否完成，期间可以进行其他操作。

异步多阻塞：异步I/O操作时并不影响已有程序后续运行。I/O完成后会主动通知主程序，减少轮询的资源损耗。

一个线程由协程来级别驱动
程序操控协程生命周期，节省了线程切换的损耗

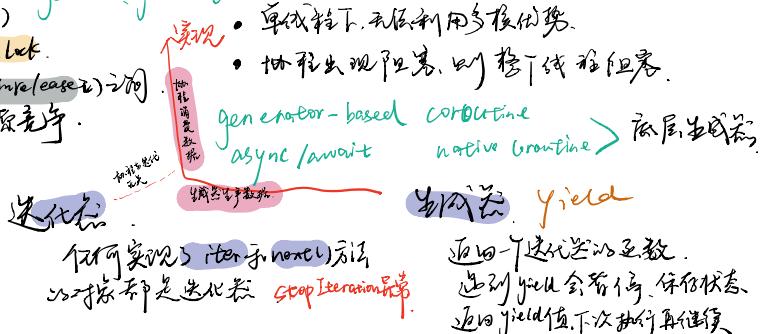
状态：
Multiprocess queue, pipe
进程间通信: IPC: 队列, 管道 (消息)
尽量避免共享数据操作。
计算密集 I/O密集型 → 协程 (微线程) (coroutine, greenlet)

多线程共享变量，使用 Thread 的 Lock 和 RLock。
只允许一个线程操作的数据，做 acquire()和 release()之间。
线程间使用队列通信，线程安全，不会资源竞争。

Threading: * 线程遇到 I/O 慢，会阻塞

GIL (Global Interpreter Lock)

Cython 解释器持有，如保证同一时间内
只有一个线程在执行



2. 单例模式（只能创建一个实例）

① 模型类

```
from functools import wraps
def singleton(cls):
    instance = {}
    @wraps(cls)
    def wrapper(*args, **kwargs):
        if cls not in instance:
            instance[cls] = cls(*args, **kwargs)
        return instance[cls]
    return wrapper
@singleton
class A():
    pass
```

3. Python ref计数：引用计数、垃圾回收、分代收集
obj.refcnt=0，会被销毁

② 闭包

class

Singleton(Meta(type)):

```
def __init__(cls, *args, **kwargs):
    cls._instance = None
    super().__init__(*args, **kwargs)
def __call__(cls, *args, **kwargs):
    if cls._instance is None:
        cls._instance = super().__call__(*args, **kwargs)
    return cls._instance
```

class A(metaclass=SingletonMeta):
pass

Mark and sweep

[0, 1, 2]代
(对象存活的时间越长，越可能不是垃圾，所以不同代)

这样设计对象，是通过

元类来创建的。

json 只处理基本数据类型，用各种语言间实行转换

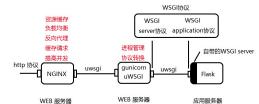
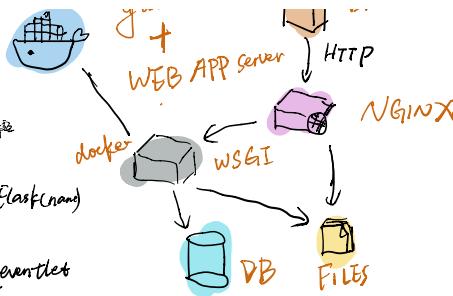
pickle For Python，所有数据都类型

unicorn Browser

4. Flask, Django, Tornado

WSGI (web server gateway Interface) web 服务的网关接口，类似于WEB如何与Python程序交互

Gunicorn: gunicorn -w 4 -b 0.0.0.0:8000 main=app
- 用 web 脚本。4 workers process 2 * CPU main.py app=flask:main
Flask 是单进程 worker，但长连接 worker_class: • sync • gevent • tornado • eventlet
worker_class (defalut) 1000



gevent 第三方库，通过 greenlet 实现多线程模型，适用于长连接，会自动找到其他线程
greenlet 与 I/O 非阻塞完成，再切换线程继续执行。

用 gevent 作为 WSGI 服务端，就可以提升性能 (Gunicorn)
greenlet 对于线程实现，gevent 是二次封装，使用 _process 方法调用 monkey_patch_all()
通过 monkey_patch，对调用 time.sleep() 会直接调用 gevent.sleep()
3 级别的高性能线程模型

Redis

分布式缓存

1. 基本数据类型:
 - 字符串 String: K-V 缓存，不适合大量数据，共享用户 session，最大容量 512M
 - hash: 存储对象，dict 的结构化信息。
 - list 双向链表，消息队列，如 Celery [broker]
 - Set: 无序、独立集 (共同归属)
 - Sorted set: (加了分数、排序) 排序榜
 - Bitmap: Bloom filter
2. 哈希 HSET: 主要场景

2. 哈希 HSET:
 - RDB 快速持久化方式 (2 进制)
 - AOF 另一种持久化方式 (逐条操作)

- 3.淘汰策略:
 - 内存限制大限制，maxmemory
 - noeviction: 不删除，直接报错
 - lru / lfu: keys-lru, 最近最少使用 key
 - volatile-lru: expire 三分之二
 - random: 随机
 - volatile-random: 到期时间随机



5. 过期策略:

定期删除，过期的 key 会放入一个列表中，每隔 10s 食心扫描，随机拿出一些 key，如果过期了就 DEL，过期比例超过 1/4，重启。

惰性删除：用户访问到某 key，过期时设置为 new，并更新。

4. MC & free memcache

- 每个 value 大约 1G, MC 1 MB
- 基本的数据类型 MC 只有 string
- 速度更快，纯内存
- 持久化，可以将所有事物写入磁盘
- redis 从 2.6 支持 Cluster

6. 雪崩:
 - 同一时间太多 key 失效，请求 DB
 - [在设置过期时间 + random]

穿透：黑客查询一个不存在的 key，如 val = -1
[用户盗取，参数校验，中间件缓存结果，Bloom Filter]

击穿：指一个 key 失效，大量请求到 DB
[热点数据不更新，反压锁]

RESTFUL API (Representational State Transfer)

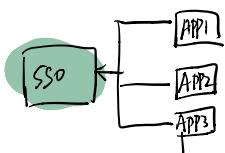
表现层、状态转移化

1. RESTful 结构: ① 每一 URL 代表一种资源
④ 客户端和服务端之间传递这种资源的表现层.
③ 客户端通过 HTTP 四个动词(方法)对资源进行操作.

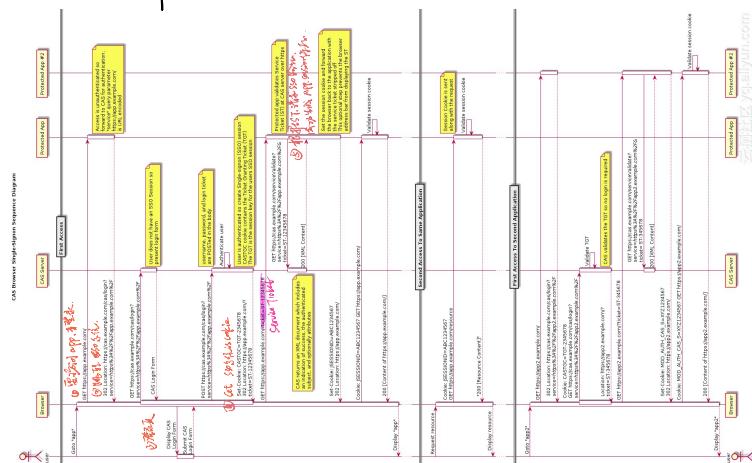
GET: 带参数的查询 URL 被限制为资源. 如果请求体携带, 则视为失败

2. URL 具有限制非 HTTP 协议层面上的. 是服务端侧览器的限制.

3. 单点登录 (SSO) 在多个应用系统中, 只需要登录一次, 就可以访问其他相互信任的系统



在设置 Cookies, 通过 cookie 实现跨域访问
从 this. a.com 这样跨域访问就不存在了.

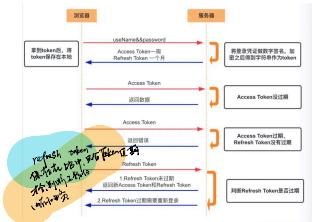


GET: 读取 read
POST: 新建 create
PUT: 更新 update
PATCH: 部分更新
DELETE: 删除 delete

Cookie: 通过客户端实现的一种数据存储机制, KV 存储在本地.

Session: 对当前用户会话的一次标识, cookie 保存, 但那是在客户端的
不安全区 [nginx ip, hash, redis 等]

Token: { header }
• user_id + pwd 生成于服务端
• 服务端会话, 返回 Token 给 client
• 每次请求携带 token 传递给 data



李家兴 ZivLi
133-1153-7312 | zivli23@163.com
Github: <https://github.com/ZivLi>

教育经历

山东科技大学 【本科】 计算机科学与技术 2011-2015

工作经历

2018.12 -- 至今 创新奇智 高级后端开发工程师

项目名称： 供应链智能平台

项目描述： 平台致力于为企业构建以 **AI** 为基石，数据融合、需求驱动的全网协同供应网络。主要服务客户为玛氏公司（MARS）。

涉及技术： Flask、Flask-RESTful、Flask-sqlalchemy、Redis、MySQL、Docker-compose、Celery 等

设计技术：

- 采用**微服务**架构，整个项目拆分为：需求规划、产能规划、智能补货、供应控制塔四大模块服务。使用 **Docker-compose** 编排前后端分离应用，通过 **Nginx** 进行部署。
- 基于 **Celery** 异步执行后台任务的特性，实现数据预测、**AI** 算法模型优化、邮件通知等业务场景。从而避免服务阻塞或前端请求超时等问题。
- 通过 **Gitlab CI/CD** 进行持续集成及部署。在 runner 中完成打包、测试、发布等工作。
- 使用 **Flask-RESTful** 设计遵循 **RESTful** 风格的 APIs。Flask-RESTful 提供的 Resource、RequestParser 等基础类，支持更优雅地处理接口请求和返回。
- 嵌入 **Sentry** 日志平台，用于监视错误以及提取一切事后处理所需的信息，而不必依赖于麻烦的用户反馈。

责任描述：

- 担任项目的技术经理，负责客户沟通、产品对接、需求剖析、框架搭建、技术选型、成员组成等相关工作。
- 深度实践**敏捷**开发工作流。初期担任 **Scrum Master**，带领团队进行项目迭代性开发工作。后期作为 **Scrum Coach** 在公司内部进行过关于敏捷开发的分享。（PMI-ACP在考）

项目名称： 购创新智能货柜

项目描述： 基于计算机视觉的商品识别技术，针对货柜场景定制训练识别模型。消费者打开货柜门后，随意取出、放回商品，通过摄像头识别货柜内拿取过程并校验结果，关门后实现系统自动结账，订单推送。

涉及技术： Flask、Redis、Amazon EC2、Amazon RDS、Amazon S3、Spring Cloud Zuul、Celery、WebSocket 等

设计技术： 整个项目包含商户 **OMS** 平台、标注系统、广告平台、**OpenAPI** 服务、**BI Dashboard**、购创新小程序（商户版/用户版）

- 用户通过扫码（跳转微信小程序）/ **AI** 人脸识别系统打开货柜门，自行拿取货柜内商品，选购完毕后关门，系统自动对用户进行结账扣款操作。
- 通过对比货柜端上传的选购前后的图像数据和重力校验值，标注系统调用 **AI** 算法模型检测，给出 SKU 变化结果，生成用户购物订单详情。

- 货柜屏幕定时轮询对应商户，在广告平台设定的广告投放策略，进行广告内容投放展示。
- 第三方应用/服务可通过调用对外 **OpenAPI** 服务，对接我方商户 **OMS** 平台，进行订单管理、SKU 模型管理、账单查询、客户服务等操作。

责任描述：

- 负责商户 OMS 平台，SKU 及模型管理相关需求后端设计开发
- 负责广告平台业务需求后端设计开发
- 参与订单系统、标注系统、Open API 服务、购创新小程序后端接口开发

2017.3 -- 2018.4

世界邮轮网

高级后端开发工程师

项目名称：ERP系统

项目描述：企业内信息化管理系统。主要服务于业务套餐搭配、订单管理、财务对接等。

涉及技术：Django、Vagrant、Jenkins、[Github 开源工具开发](#)

设计技术：

- 帮助从 **Kanban** 工具 TargetProcess API 中获取数据的 **python package**
- 系统后端基于 **Django** 框架进行开发，并利用 **Django-admin** 做系统后台管理

责任描述：

- 学习并参与完整**敏捷开发**工作流，实习担任 **Scrum Master**，组织 **Scrum Meetings**。熟练掌握使用的敏捷工具，如：**Slack**、**Trello**、**TargetProcess**、**Burndown Chart** 等
- 参与业务模块需求后端开发
- **K-Means** 算法知识分享

2015.6 -- 2017.3

赤子城科技有限公司

Python后端开发工程师

项目名称：广告业务平台

项目描述：公司广告业务管理平台。基于平台可以实现与上游广告主的对接，Offer设置以及对接下游放量控制等操作。

涉及技术：Tornado、MongoDB、Redis、Scrapy、GoogleADS 等

责任描述：

- 参与业务模块需求后端开发、上下游对接 API 接口。包括上下游跳转，日志统计记录等功能。
- 实现脚本从 Google 商店爬取转化率较高 APP 的相关信息，如：icon、title、rate 等。并下载 apk。

其他

-
- 参与开源社区 [freeCodeCamp 文章翻译计划](#)。
 - 具有良好的沟通能力和团队合作精神，善于理解业务场景及需求剖析。
 - 有良好的设计和编码品味，严谨且优雅的代码风格，遵循 [Google Style Guide](#)。
 - 兴趣爱好：羽毛球、滑雪、狼人杀、剧本杀、密室逃脱等。