2.3 分布式存储及 WEB 服务器性能

- 后端知识体系
    - 语言：语法 20%
    - 语言生态：库、框架 20%
    - 服务器：Linux 15%
    - 网络：15%
    - 存储：
        - File 10%
        - SQL 20%

高可用与负载均衡

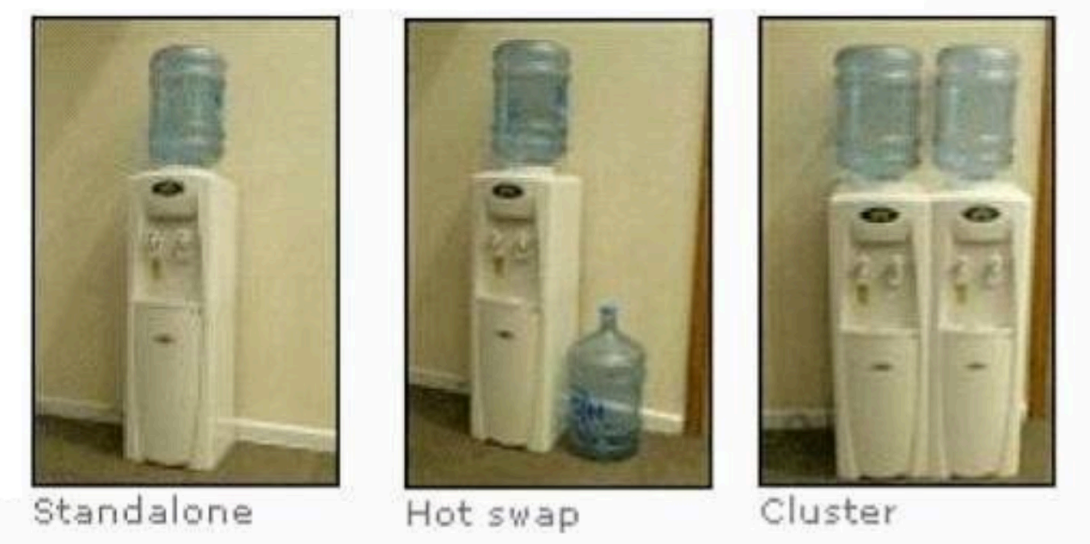
- 单点故障：
    - 一个服务只有一个可工作的节点，一旦发生问题，就完全无法提供服务，这样的故障称为单点故障
- 冗余：多余的，一大好处是，有问题的时候，有一个备用的，缺点是成本双倍。
    - 部件一级的
        - 磁盘的冗余
            - RAID
    - 服务器一级的
        - 读写分离
        - 热备
        - 负载均衡
- 故障迁移：
    - 当一个服务器发生故障，无法对外提供服务时，由专门的检测程序监测到异常，把它原本负担的工作，转交由其他服务器处理，这个过程称为故障迁移，或者故障转移
- 高可用：HA：High Available
    - 什么时候都要能工作，任何一次请求不允许出错，不允许出现单点故障
    - 用冗余来对抗故障
    - **数据库强调高可用**
    - 高可用不关注性能提升，最关注服务和数据的可靠性
- 负载均衡：LB：Load Balance

    - 访问量上来了，要能分配好，总体上保证服务可用
    - 如果某个服务器坏了，再把定向到这台服务器的请求分配到其他服务器，在重新分配期间，可能暂时的一小部分请求受影响，无法工作，能够接受
    - **Web server 和 App(lication) server(py,java,php) 强调负载均衡**
- 热备份：备份时数据库不停止工作
    - 一个服务器工作时，旁边有一个相同配置的服务器 Stand by，不断与主服务器同步，随时待命，一旦主服务器发生故障，立即切换顶上去工作
    - 缺点：浪费一半硬件资源
- 冷备份：备份时数据库要停止工作
    - 有一个完整的时间点的备份
- 面试题：
    - 如何处理高并发问题

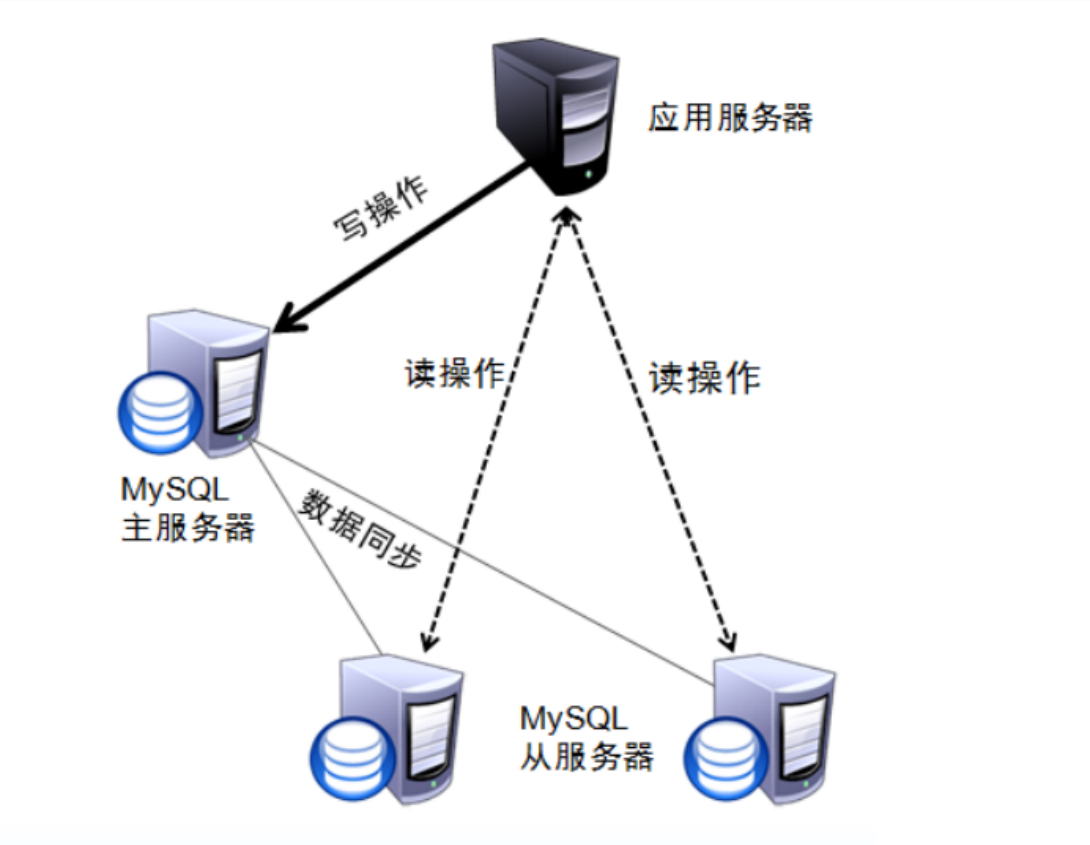
MYSQL 的性能升级

- 搞清楚业务的读写比例：
    - CRUD：RW
    - 读密集：加**读缓存**，用 NoSQL 做 Cache
        - 社区
        - 广告
        - 视频
    - 写密集：加**写缓冲**或者分表分库，在真正写库之前，先写到一个高速的中间层，然后再合并写入数据库
        - 电商：
            - 浏览商品：读密集
            - 下单：写密集
        - 游戏：操作多，全记录，写密集
            - 偷菜：500万DAU
            - 单用户操作200
            - 单日10亿次写入的应用
    - 读写均衡：
        - 论坛
        - 微博
- 把观察问题的视角上升到整个系统架构上
    - 搞清楚问题是不是在 MySQL 上
    - 表现在 MySQL 上，问题不一定在 MySQL 上
    - MySQL 并发连接数超高
        - 可能是程序没有正确使用**连接池**
- 确认瓶颈在SQL， MySQL 本身，针对 MySQL 如何解决性能问题
    - **单机优化**

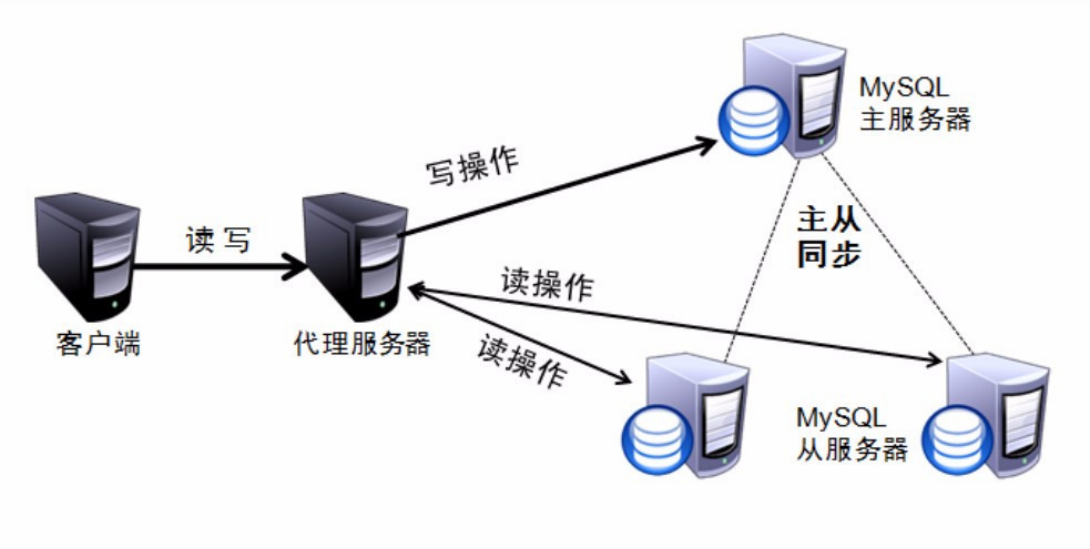
        - 80% : 分析优化 SQL语句，充分利用慢查询日志，打开监控慢查询
        - 索引优化
        - 优化表设计，调整结构，增加冗余的字段，节约计算
            - 大表拆小表
                - 把可变的数据与不变的分开
                - 尽量保持小表设计
        - 水平拆分：
            - 按某个规则把数据分散放到结构相同，但名字不同的表中
            - User_00..User_ff
        - 分析优化对 MySQL 的程序访问连接
    - 修改 my.cnf 配置，修改表和排序的缓存值
        - innodb_buffer_pool_size
    - **更换系统malloc的库，就能提升性能**
        - **Jmalloc**
        - **google**
        - 更换更高效的存储引擎：MyISM -> InnoDB -> XtraDB / PerconaDB: 5%-10%
    - 硬件投入能马上解决的，升级内存，升级存储，先硬件投入，见效最快
    - 通过增加冗余的数据库服务器可构建数据库集群



    - 同一集群中的多台数据库保存的数据必须完全一致
    - 集群中一台服务器宕机，其他服务器可以继续提供服务
    - 常见结构：一主多从，主从之间通过 binlog 进行数据同步
- 读写分离
    - 主机用来做数据写入；从机用来数据读取
        - 程序自身实现读写分离



- 使用第三方代理实现读写分离
  - mysql_proxy
  - Atlas:360
  - Amoeba:alibaba
  - *sohu*
  - netease



- **数据库 Sharding** / 分库分表 / 数据库分片：**重点介绍**
- **分布式数据库**
- 云数据库：新兴方式
- **增加缓存**
  - file
  - mem

        - NoSQL
- 面试题：
    - 数据库的优化方法

数据库 SHARDING / 分库分表 / 数据分片

1. 数据分片
    - 单表查询能力上限: 对 MySQL 来说，约为 500 万 左右，设计优良的小表(字段很少) 1000 万 也是上限了
    - 方式: 分库、分表
2. 垂直拆分：
    1. 拆分原则：把经常变化的数据与不变的数据分开

    单表字段太多的时候会进行垂直拆分，不是为了分布式存储，而是为了提升单表性能

```
                                垂直拆分
                                   |
user                               | ext_info
                                   |
| id  name  sex  age  location | uid aa  bb  cc  dd  ee  ff |
| ---------------------------- | -------------------------- |
| 1   xxx   f    11   beijing  | 1   x   x   x   x   x   x  |
| 2   xxx   f    11   beijing  | 2   x   x   x   x   x   x  |
| 3   xxx   f    11   beijing  | 3   x   x   x   x   x   x  |
| 4   xxx   f    11   beijing  | 4   x   x   x   x   x   x  |
| 5   xxx   f    11   beijing  | 5   x   x   x   x   x   x  |
| 6   xxx   f    11   beijing  | 6   x   x   x   x   x   x  |
| 7   xxx   f    11   beijing  | 7   x   x   x   x   x   x  |
| 8   xxx   f    11   beijing  | 8   x   x   x   x   x   x  |
| 9   xxx   f    11   beijing  | 9   x   x   x   x   x   x  |
```

3. 水平拆分

    水平拆分既可以用在 “分表” 处理，也可用在 “分库” 处理

```
user
id  name  sex  age  location  aa  bb  cc  dd  ee  ff
------------------------------------------------------ user_1
1   xxx   f    11   beijing   x   x   x   x   x   x
2   xxx   f    11   beijing   x   x   x   x   x   x
3   xxx   f    11   beijing   x   x   x   x   x   x
------------------------------------------------------ user_2
4   xxx   f    11   beijing   x   x   x   x   x   x
5   xxx   f    11   beijing   x   x   x   x   x   x
6   xxx   f    11   beijing   x   x   x   x   x   x
------------------------------------------------------ user_3
7   xxx   f    11   beijing   x   x   x   x   x   x
8   xxx   f    11   beijing   x   x   x   x   x   x
```

```
9   xxx   f    11   beijing   x   x   x   x   x   x
```

- 按范围拆分
  - 优点: 构建简单, 扩容极其方便.
  - 缺点: 不能随运营发展均衡分配资源
  - 示例

```
Database-1       1 -  500W   <- uid: 3120482
Database-2    500W - 1000W
Database-3   1000W - 1500W   <- post_id: 20278327
Database-4   1500W - 2000W
```

- 按余数拆分
  - 优点: 能够随着运营发展均匀分配负载
  - 缺点: 扩容不方便, 前期投入大
  - 示例

```
uid = 3120483
mod = uid % len(Databases) -> 3
db_name = 'Database-3'

Database-0      10  20  30   ...  3120480
Database-1   1  11  21  31   ...  3120481
Database-2   2  12  22  32   ...  3120482
Database-3   3  13  23  33   ...  3120483
Database-4   4  14  24  34   ...  3120484
Database-5   5  15  25  35   ...  3120485
Database-6   6  16  26  36   ...  3120486
Database-7   7  17  27  37   ...  3120487
Database-8   8  18  28  38   ...  3120488
Database-9   9  19  29  39   ...  3120489
```

- 按日期分拆
- 如何分拆完全取决于业务
  - 跟日期强相关的数据，可以按日期分拆
    - 订单
    - 经常会变更日期的数据
  - 如果没有什么特征的数据，尽量平均分散，用传统的 hash 或者 一致性 hash 等方式尽量均匀分布

4. 分布式数据库的 ID
   - 必须保证全服多机上产生的 ID 唯一，否则会发生主键冲突
   - 常见全局唯一 ID 生成策略
     1. 基于存储的自增 ID

        - 可在 Redis 中为每一个表记录当前最新 ID 是多少，获取下一个 ID 时进行自增
        - 优点：思路简单，ID 连续
        - 缺点：有存储依赖，一旦 Redis 出现问题，则会影响全部数据库存储
    2. 基于算法确保唯一
        - 常见算法有 UUID、COMB、Snowflake、ObjectID 等
        - 优点：快速、无存储依赖
        - 缺点：一般产生的 ID 数值都比较大，某些算法的 ID 并非是增序
  - 一个真实的全局唯一 ID 生成策略：
  - ID生成器，序号发生器，发号器，发号服务
    - snowflake：独立服务
    - flickr：
      - 用一个MySQL服务器的自增表负责发号
      - Vego项目：看研发文档
5. 面试题：
    1. 如何分表分库?

2019年的做法

1. 直接使用云数据库
    1. 阿里分布式云数据库（基于 PostgreSQL 修改）
    2. TiDB等国产新兴分布式数据库
    3. GreenPlum：PostgreSQL 系的新兴分布式数据库，比较吃硬件，网络要好
2. 特点：
    1. 直接支持 SQL 标准
    2. 扩容、缩容对程序透明
    3. 不需要自己去维护数据库扩容问题
    4. 起步成本低，可以随着业务量逐渐加
    5. 节省维护的人力成本

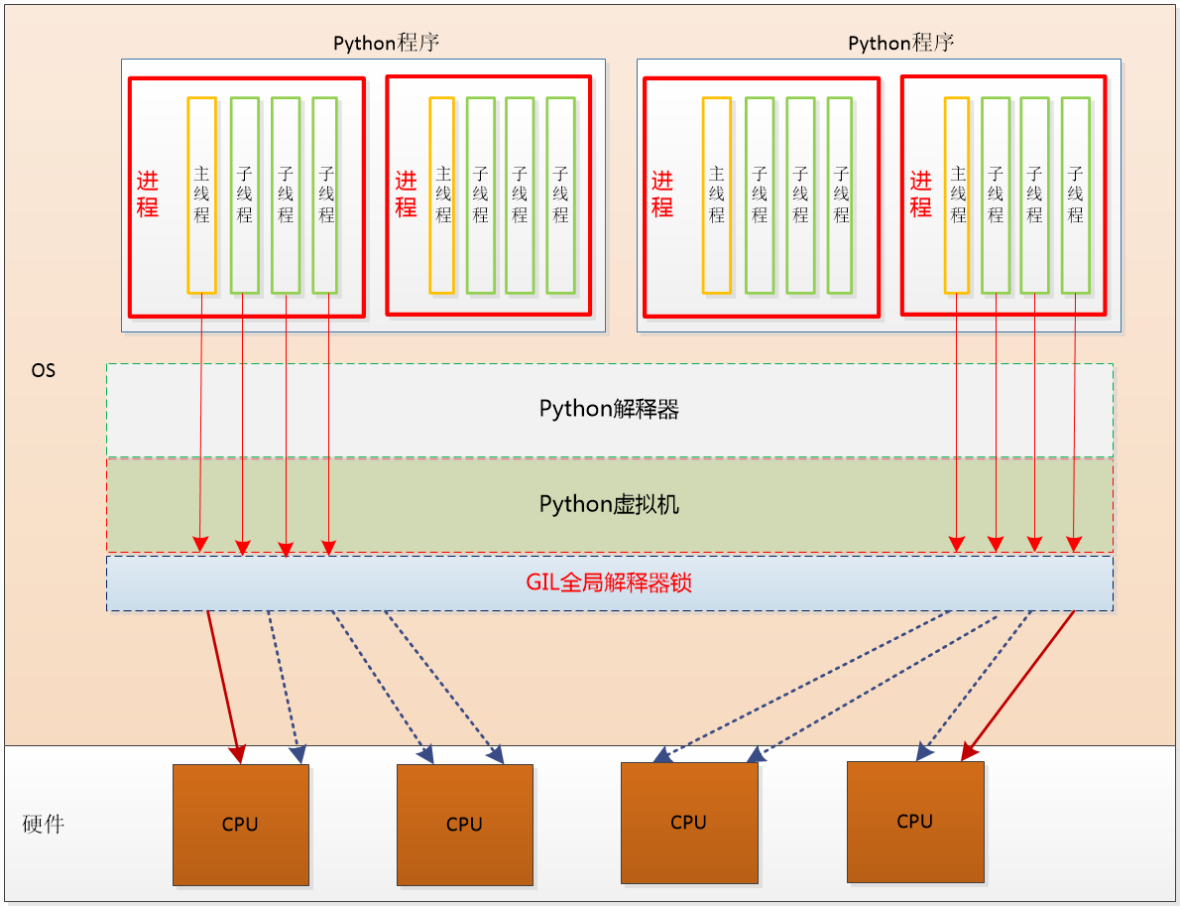
数据库集群

- 换一个角度看并发与性能
- 概念
  - 理解 I/O 的概念
    - Input / Output
      - 网卡：
        - 1000 M bps：bit per second / Bps : Bytes per second
      - 1000M / 8 = 125MB

        - 万兆网卡：infini band
    - 磁盘：SSD：500MB
  - 理解“同步/异步”、“阻塞/非阻塞”
  - 了解“事件驱动”和“多路复用”
  - 异步模型并不会消灭阻塞，而是在发生 I/O 阻塞时切换到其他任务，从而达到异步非阻塞
- 计算密集型
  - python为什么在这些场景能立足呢?
  - numpy pandas都是高效的实现
    - python入门对不懂程序的人，是最简单的
  - CPU 长时间满负荷运行，如图像处理、大数据运算、科学运算等
  - GPU
  - 计算密集型: 用 C 语言或 Cython 补充
- I/O 密集型：python程序
  - 网络 IO，文件 IO，设备 IO 等
  - Unix: 一切皆文件，所有设备都可以模拟成像文件一样的接口，可以像访问文件一样去访问
  - 打开
    - 读
    - 写
    - 关闭
- 多任务处理
  - 进程、线程、协程调度的过程叫做上下文切换
  - 进程、线程、协程对比

名称	资源占用	数据通信	上下文切换 (Context)
进程	大	不方便 (网络、共享内存、管道等)	操作系统按时间片切换，不够灵活，慢
线程	小	非常方便	按时间片切换，不够灵活，快
协程	非常小	非常方便	根据I/O事件切换，更加有效的利用CPU

- 全局解释器锁（GIL）

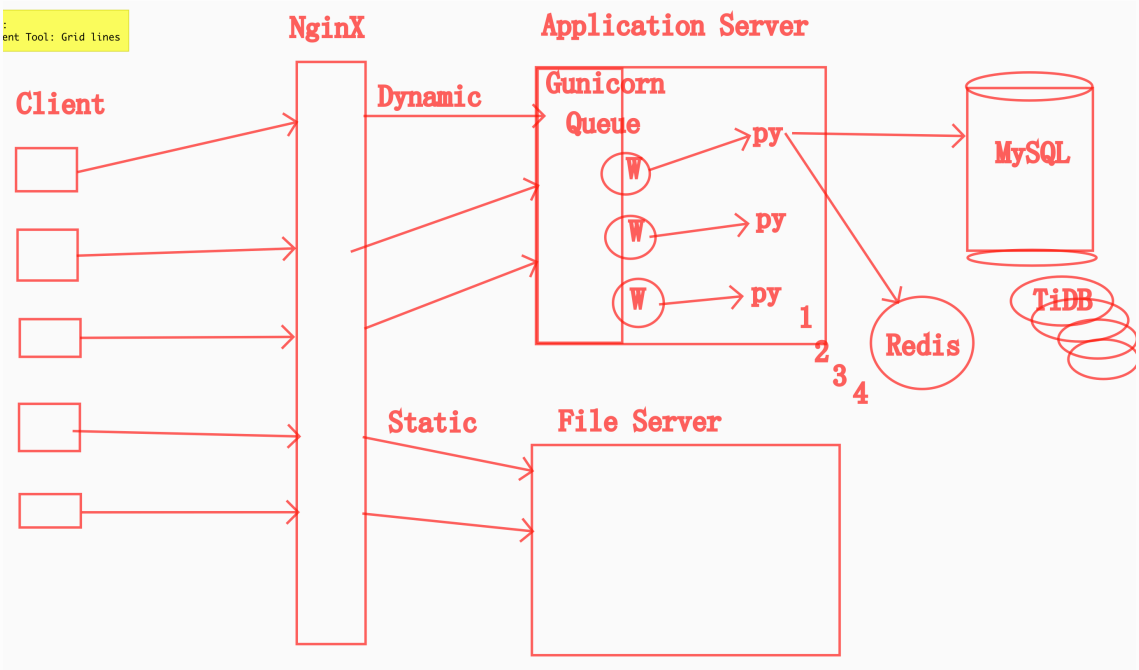


  - 它确保任何时候一个进程中都只有一个 Python 线程能进入 CPU 执行。
  - 全局解释器锁造成单个进程无法使用多个 CPU 核心
  - 通过多进程来利用多个 CPU 核心，一般进程数与CPU核心数相等，或者CPU核心数两倍
- 协程
  - Python 下协程的发展:
    - stackless / greenlet / gevent(monkey_patch)
    - tornado 通过纯 Python 代码实现了协程处理 (底层使用 yield)
    - asyncio: Python 官方实现的协程
  - async-io 实现协程

```
import asyncio

async def foo(n):
    for i in range(10):
        print('wait %s s' % n)
        await asyncio.sleep(n)
    return i

task1 = foo(1)
task2 = foo(1.5)
tasks = [asyncio.ensure_future(task1),
         asyncio.ensure_future(task2)]

loop = asyncio.get_event_loop()  # 事件循环，协程调度器
loop.run_until_complete( asyncio.wait(tasks) )
```

- **结论：通常使用多进程 + 多协程达到最大并发性能**
    - 因为 **GIL** 的原因，Python 需要通过**多进程**来利用多个核心
    - 线程切换效率低，而且应对 I/O 不够灵活
    - 协程更轻量级，完全没有协程切换的消耗，而且可以由程序自身统一调度和切换
- HTTP Server 中，每一个请求都由独立的协程来处理
- **单台服务器最大连接数**
    - 文件描述符: 限制**文件打开数量** (一切皆文件)
    - 内核限制: `net.core.somaxconn`
    - 内存限制
- 修改文件描述符: `ulimit -n 65535` 实际上单台服务器可以远远超过这个限制，单服务器可以处理上百万的并发连接
- 使用 Gunicorn 驱动 Django
    - http://docs.gunicorn.org/en/latest/install.html
    - Gunicorn 扮演 HTTPServer 的角色
    - HTTPServer: 只负责网络连接 (TCP握手、数据收/发)
- 分清几个概念
    - CGI
    - common gateway interface
        - Unix shell, Python, Ruby, PHP, Tcl, C/C++,和Visual Basic都可以用来编写CGI程序。
    - Fast-CGI
    - PHP
    - **WSGI:** 全称是 Web Server Gateway Interface，它是 Python 官方定义的一种描述 HTTP 服务器 (如nginx)与 Web 应用程序 (如 Django、Flask) 通信的规范。全文定义在 PEP333
- **uwsgi:** 与 WSGI 类似，是 uWSGI 服务器自定义的通信协议，用于定义传输信息的类型(type of information)。每一个 uwsgi packet 前 4byte 为传输信息类型的描述，**与 WSGI 协议是两种东西**，该协议性能远好于早期的 Fast-CGI 协议。
    - uWSGI: uWSGI 是一个全功能的 HTTP 服务器，实现了WSGI协议、uwsgi 协议、http 协议等。它要做的就是把 HTTP协议转化成语言支持的网络协议。比如把 HTTP 协议转化成 WSGI 协议，让 Python 可以直接使用。

```
HTTP Server  => 负责 1. 接受、断开客户端请求; 2. 接收、发送网络数据
    ^
    |
    v
  WSGI       => 负责 在 HTTPServer 和 WebApp 之间进行数据转换
    ^
    |
    v
Web App      => 负责 Web 应用的业务逻辑
```



- 面试题:
  - 请描述你做过的服务器结构/架构/部署

压力测试

- 常用工具
  - ab (apache benchmark)
    - Ubuntu 下安装 ab: `apt-get install apache2-utils`
    - 压测: `ab -k -n 1000 -c 300 http://127.0.0.1:9000/`
  - wrk
  - wrk -t2 -c20 -d5s http://www.baidu.com
    - locust
- Web 系统性能关键指标:
  - 并发连接数
  - RPS (Requests per second)
- 其他: MySQL
  - **QPS (每秒查询数): Query per second:**
    - QPS :300
  - QPS: 1000-3000
    - QPS: 5000
    - TPS (每秒事务数, 数据库指标): Transaction per second
- 面试题:
  - 如何做压力测试?