

BBS Project

Step 0

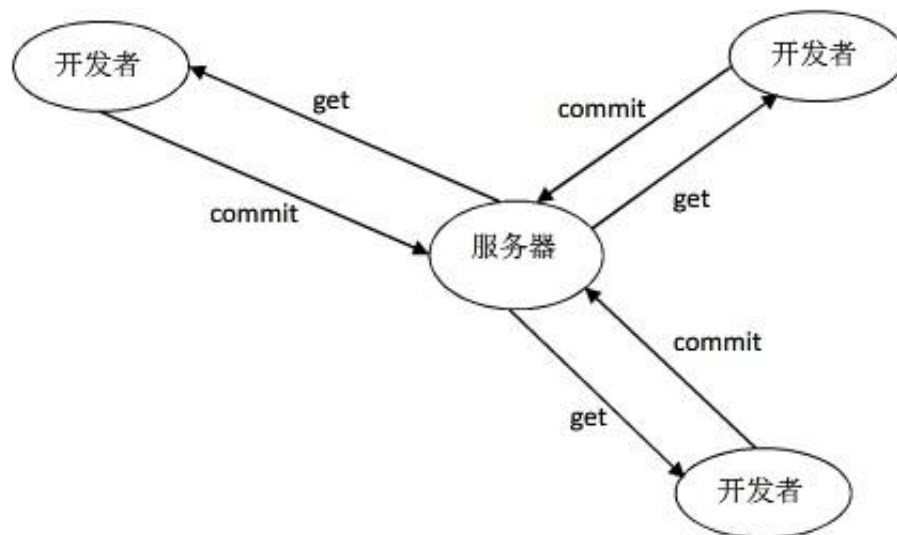
1. 项目目标
2. Web 项目布局
 - views.py 不适合写逻辑，添加 helper.py 放置通用逻辑函数
 - 通用的算法、功能放到 common 目录
 - 底层的功能放到 lib 目录
 - 配置文件放到项目目录

```
proj/
├─ app1/
│   ├─ migrations/
│   ├─ __init__.py
│   ├─ apps.py
│   ├─ helper.py
│   ├─ models.py
│   └─ views.py
├─ app2/
│   ├─ migrations/
│   ├─ __init__.py
│   ├─ apps.py
│   ├─ helper.py
│   ├─ models.py
│   └─ views.py
├─ common/
│   ├─ __init__.py
│   ├─ middleware.py
│   └─ keys.py
├─ lib/
├─ proj/
│   ├─ __init__.py
│   ├─ settings.py
│   ├─ urls.py
│   └─ wsgi.py
└─ manage.py
```

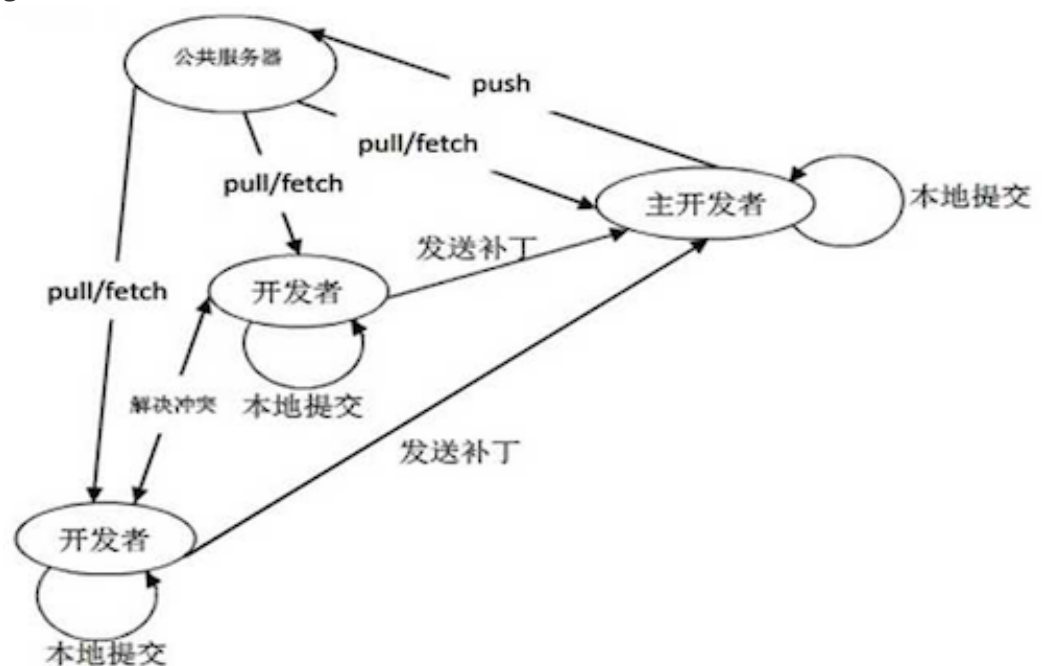
Step 1

1. Git
 - 版本控制工具
 - CVS: 基本退出了历史舞台

- svn: 中心化的版本控制工具, 需要有一台中心服务器



- git: 分布式的版本控制工具, 中心服务器不再是必需的



- hg: 纯 Python 开发的版本控制工具
- 文本类的东西都可以交由版本控制工具来管理
- 常用操作
 - `git clone` 从远程克隆一个已存在的项目
 - `git init` 初始化一个 git 库
 - `git add` 把文件添加到缓冲区
 - `git commit -m 'xxxxxxx'` 把修改的文件提交到 git 库中
 - `git push` 把本地修改推送到服务器
 - `git pull` 把远程修改拉取到本地并且合并
 - `git fetch` 把远程修改拉取到本地
 - `git log` 查看提交日志
 - `git checkout` 切换分支, 切换代码版本
 - `git branch` 创建分支
 - `git merge` 合并
 - `git diff` 对比差异

- 要忽略的文件 `.gitignore`
- 安装
 - Windows: <https://gitforwindows.org/>
 - Linux: `apt install git`
 - Mac: `brew install git`

2. 代码管理

- 分支: master / develop / feature
- 通过提交 Pull Request 将合并过程暴露给团队成员, 让其他人帮助自己做代码审核, 保证代码质量
- 代码审核 (Code Review)
 - 发现代码逻辑问题
 - 代码风格及规范化问题
 - 算法问题
 - 错误的使用方式
 - 能够学习其他人的优秀代码

3. 项目分组

4. 网站基础功能

- 创建帖子
- 修改帖子
- 阅读帖子
- 删除帖子
- 查看帖子列表
- 帖子列表分页功能 (自行实现)
- 根据正文搜索帖子 `Post.objects.filter(...)`

5. 新建项目流程

```
$ mkdir demo
$ cd demo
$ cat > .gitignore << EOF
*.pyc
*.sqlite3
.idea
__pycache__
*.log
.venv
medias/*
EOF
$ python -m venv .venv
$ source .venv/bin/activate
$ pip install ipython django==1.11.7 redis django-redis gevent
  gunicorn Pillow requests
$ pip freeze > requirements.txt
$ django-admin startproject demo ./
$ git init
$ git add ./
```

```
$ git commit -m 'first commit'
$ git remote add origin git@github.com:yourname/demo.git
$ git push -u origin master
```

Step 2

1. Cookie / Session 机制剖析

1. 产生过程

1. 浏览器: 向服务器发送请求
2. 服务器: 接受并创建 session 对象 (该对象包含一个 session_id)
3. 服务器: 执行 views 函数, 并得到一个 response 对象
4. 服务器: 执行 response.set_cookie('sessionid', session_id) 将 session_id 写入 cookie
5. 服务器: 将 response 传回浏览器
6. 浏览器: 读取 response 报文, 从 Cookies 取出 session_id 并保存

2. 后续请求

1. 浏览器: 向服务器发送请求, session_id 随 Cookies 一同发给 Server
2. 服务器: 从 Headers 的 Cookies 中取出 session_id
3. 服务器: 根据 session_id 找出对应的数据, 确认客户端身份

2. 注册功能

- 使用 ModelForm

3. 登录功能

- 密码编码处理: make_password, check_password
- `from django.contrib.auth.hashers import make_password, check_password`

4. 个人信息页

- 昵称
- 头像
- 年龄
- 性别


5. 头像上传功能

- 模版的 form 标签添加属性: `enctype=multipart/form-data`
- settings 添加配置: MEDIA_ROOT, MEDIA_URL
- urls 添加处理: `urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)`

6. 退出功能

7. 以微博为例的第三方登录



 来自draft-ietf-oauth-v2

1. OAuth2.0

1. 引导用户至授权页面
2. 用户点击授权
3. 服务器向第三方平台申请 Access Token
4. 第三方平台回调服务器接口
5. 服务器保存 Access Token
6. 使用 Access Token 访问第三方资源

2. 微博接口

- [微博登录](#)
- [授权机制](#)

3. 接入步骤

- [引导用户到授权接口](#)
- [获取AccessToken接口](#)
- [根据UID获取用户信息接口](#)

Step 3

1. Redis 简介

- 常用数据类型
 - **String** 类：常用作普通缓存

CMD	Example	Description
set	set('a', 123)	设置值
get	get('a')	获取值
incr	incr('a')	自增
decr	decr('a')	自减
mset	mset(a=123, b=456, c=789)	设置多个值
mget	mget(['a', 'b', 'c'])	获取多个值
setex	setex('kk', 21, 10)	设置值的时候，同时设置过期时间
setnx	setnx('a', 999)	如果不存在，则设置该值

■ **Hash 类**：常用作对象存储

CMD	Example	Description
hset	hset('obj', 'name', 'hello')	在哈希表 obj 中添加一个 name = hello 的值
hget	hget('obj', 'name')	获取哈希表 obj 中的值
hmset	hmset('obj', {'a': 1, 'b': 3})	在哈希表中设置多个值
hmget	hmget('obj', ['a', 'b', 'name'])	获取多个哈希表中的值
hgetall	hgetall('obj')	获取多个哈希表中所有的值
hincrby	hincrby('obj', 'count')	将哈希表中的某个值自增 1
hdecrby	hdecrby('obj', 'count')	将哈希表中的某个值自减 1

■ **List 类**：常用作队列(消息队列、任务队列等)

CMD	Example	Description
lpush	lpush(name, *values)	向列表左侧添加多个元素
rpush	rpush(name, *values)	向列表右侧添加多个元素
lpop	lpop(name)	从列表左侧弹出一个元素
rpop	rpopt(name)	从列表右侧弹出一个元素
blpop	blpop(keys, timeout=0)	从列表左侧弹出一个元素, 列表为空时阻塞 timeout 秒
brpop	brpop(keys, timeout=0)	从列表右侧弹出一个元素, 列表为空时阻塞 timeout 秒
llen	llen(name)	获取列表长度
ltrim	ltrim(name, start, end)	从 start 到 end 位置截断列表

■ **Set** 类：常用作去重

CMD	Example	Description
sadd	sadd(name, *values)	向集合中添加元素
sdiff	sdiff(keys, *args)	多个集合做差集
sinter	sinter(keys, *args)	多个集合取交集
sunion	sunion(keys, *args)	多个集合取并集
sismember	sismember(name, value)	元素 value 是否是集合 name 中的成员
smembers	smembers(name)	集合 name 中的全部成员
spop	spop(name)	随机弹出一个成员
srem	srem(name, *values)	删除一个或多个成员

■ **SortedSet** 类：常用作排行处理

CMD	Example	Description
zadd	zadd(name, a=12)	添加一个 a, 值为 12
zcount	zcount(name, min, max)	从 min 到 max 的元素个数
zincrby	zincrby(name, key, 1)	key 对应的值自增 1
zrange	zrange(name, 0, -1, withscores=False)	按升序返回排名 0 到 最后一位的全部元素
zrevrange	zrevrange(name, 0, -1, withscores=False)	按降序返回排名 0 到 最后一位的全部元素
zrem	zrem(name, *value)	删除一个或多个元素

- Django 使用 Redis 缓存

```
pip install django_redis
```

```
# settings 添加如下配置
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "PICKLE_VERSION": -1,
        }
    }
}
```

2. 增加帖子缓存功能

1. 普通缓存
2. 通过装饰器实现缓存处理, 并通过参数调节缓存时间

3. 增加帖子热门排行, 显示点击最高的 10 篇帖子

1. 取出榜单数据: `redis.zrevrange('rank', 0, 9, withscores=True)`
2. 累加分数: `redis.zincrby('rank', 'yourname')`
3. 其他接口

```
redis.zadd
redis.zrange
redis.zrevrange
redis.zincrby
```

4. 开发一个中间件, 限制用户的访问频率最大为每秒 3 次, 超过 3 次时, 封禁 IP 24 小时

1. 记录用户访问的时刻 (该值记录在哪里)

2. 检查用户距离之前某次访问的时差是否在合理范围之内 (应该记录几个值)
3. 怎样区分出每个用户 (`user_ip = request.META['REMOTE_ADDR']`)

Step 4

1. 外键及分布式关系

- 内部系统、传统企业级应用可以使用 (需要数据量可控, 数据库服务器数量可控)
- 互联网行业不建议使用
 - 性能缺陷
 - 不能用于分布式环境
 - 不容易做到数据解耦

2. 用户和帖子的关联

3. 增加评论功能

4. 增加 Tag 模块

1. 修改帖子时, 创建Tag, 创建、修改 Post-Tag 关系
2. 点击 Tag, 筛选出 Tag 下的所有帖子
3. 关联 Post 的所有 Tag
4. 关联 Tag 的所有 Post

5. 增加权限管理模块

1. 分级权限模型

- level-4 admin 允许添加、删除帖子, 允许发表、删除评论、删除用户、任命管理员
- level-3 manager 允许添加、删除帖子, 允许发表、删除评论
- level-2 user 允许添加帖子, 允许发表评论
- level-1 guest 允许看帖子

2. 基于角色的权限模型

- 用户表
- “用户-角色”关系表
- 角色表
- “角色-权限”关系表
- 权限表

6. 数据分片

- 分表
- 分库
 - 垂直切分: 将多个字段拆分到子表
 - 水平切分
 - 基于范围拆分: 扩容方便, 但冷热数据不均匀
 - 基于哈希拆分: 扩容不方便, 冷热数据分布均匀
 - 取模: $uid \% n$
 - 哈希
 - 一致性哈希: 使用虚拟节点解决扩容不方便的问题

7. DB 集群

- 主从备份
- 一主两从
- 双主互备
- 服务高可用

Step 5

1. Web 性能

- 单台服务器最大连接数
 - 文件描述符：限制文件打开数量
 - 内核限制： `net.core.somaxconn`
 - 内存限制
 - 修改文件描述符： `ulimit -n 65535`

2. 压力测试

- 常用工具
 - [ab \(apache benchmark\)](#)
 - [siege](#)
 - `webbench`
 - `wrk`
- Web 系统关键指标: **RPS** (Requests per second)
- 其他: QPS (每秒查询数) / TPS (每秒事务数, 数据库指标)
- Ubuntu 下安装: `apt-get install apache2-utils`
- 压测: `ab -n 1000 -c 300 http://127.0.0.1:9000/`

3. 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)



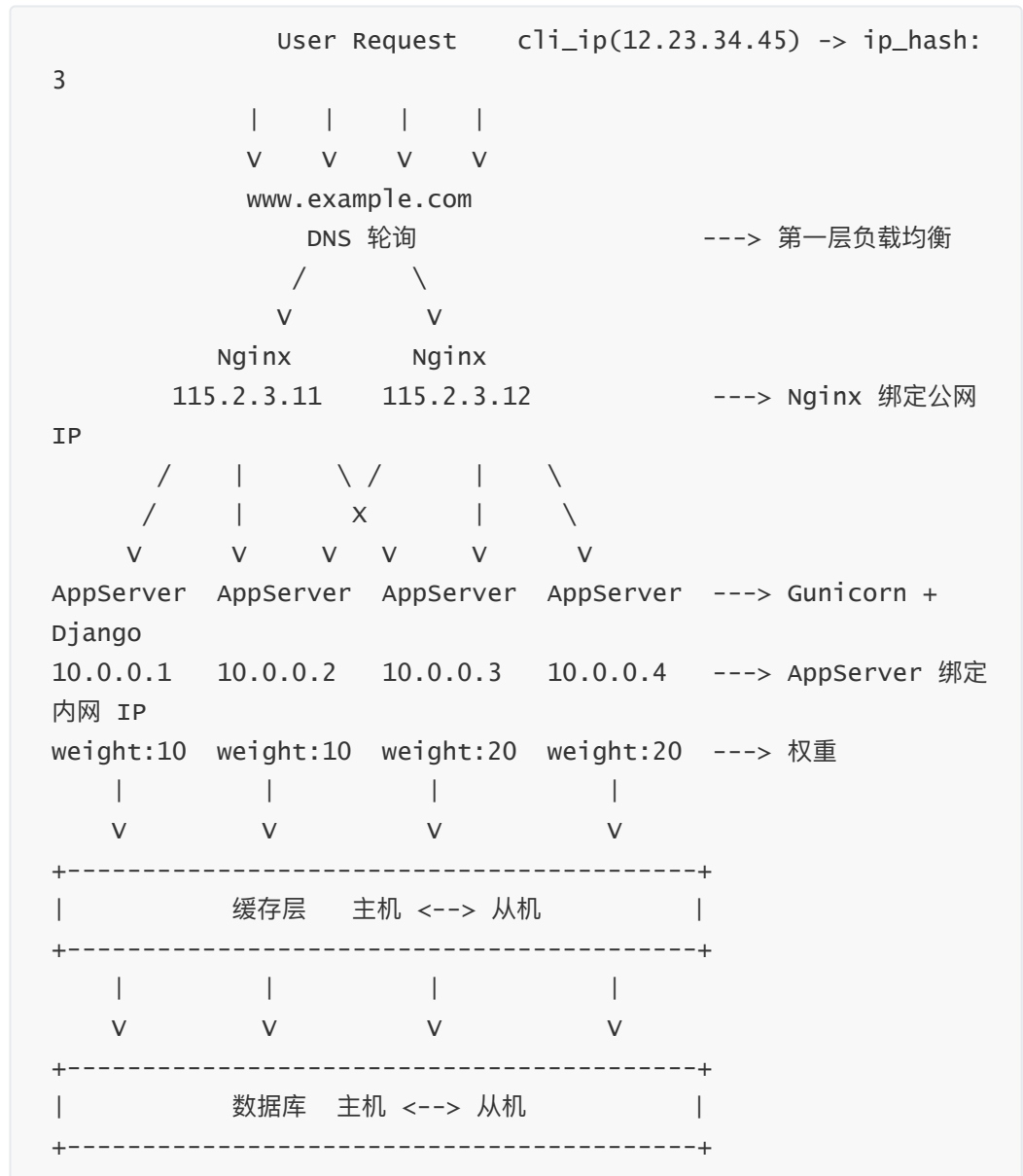
4. 使用 Bash 脚本进行部署

- 代码发布脚本
- 程序启动脚本
- 程序停止脚本
- 程序重启脚本

- 不间断重启: `kill -HUP [进程 ID]`

5. 增加 Nginx

- 反向代理
- 负载均衡
 - 轮询: rr (默认)
 - 权重: weight
 - IP哈希: ip_hash
 - 最小连接数: least_conn



- 可以不使用 Nginx, 直接用 gunicorn 吗?
 - Nginx 相对于 Gunicorn 来说更安全
 - Nginx 可以用作负载均衡
- 处理静态文件相关配置

```
location /statics/ {  
    root    /project/bbs/;  
    expires 30d;  
    access_log off;  
}
```

6. 生产环境中静态文件存储

- 线上系统 Django 会关掉自身的静态文件处理
- 用 Nginx 代理静态文件
- CDN (内容分发网络)
 - 基于缓存技术为静态资源 (主要是多媒体资源) 提供访问加速的服务
 - 在不同地区部署镜像服务器节点
 - 定期与源站做内容同步