1.1 真实的公司组织结构及研发流程

自我介绍

- 余泽斌
- 19年互联网研发经验，从 2000 年开始从事互联网技术研发工作，历任多家公司技术总监、技术副总，做过社区、电商、软件、广告、游戏等众多业务。
- 工作用过的语言：
    - C/C++/ObjC 、 Java 、 C# 、 JavaScript 、 PHP 、 Python 、 Shell 、 ASP 、 Lua(Redis/NginX)
- 玩票学过的语言：
    - Erlang(高并发函数语言)、Rust(Go的竞争者)、Ruby(Python竞争者)、Perl、Lips/Schema

项目课程（10天）

- 1.1：真实的公司组织结构及研发流程、Git
- 1.2：开始一个新Git项目、前后端分离开发
- 1.3：验证码、缓存、登录、表单验证
- 1.4：上传文件、异步、初始化脚本
- 1.5：social接口
- 2.1：会员、权限、日志、Redis
- 2.2：缓存、排行榜、积分接口
- 2.3：分布式数据库、集群、高可用、压测
- 2.4：SSH、部署、脚本
- 2.5：WebServer、负载均衡、服务器架构
- 有空余时间，专门讲一下数据分析系统在公司中的应用

项目概览

Swiper Social 是一个类似于“探探”的社交类程序，采用前后端分离结构，主要包含以下模块：

1. 个人模块：注册、登录、查看资料、修改资料、上传照片
2. 社交模块：喜欢、超级喜欢、不喜欢
3. VIP 模块：反悔

4. 异步任务模块：把比较耗时的，没必要让用户等的任务异步处理
5. Redis 缓存模块：**对频繁读取的，更新很少的数据**，放入缓存，减轻数据库压力
6. 日志模块、异常处理模块
7. 短信模块、邮件模块
8. 运维、部署、shell 脚本
9. 前端模块
10. 其他

项目目标

1. ✓ 了解真实项目的开发流程
2. ✓ 掌握如何使用 Git 完成协作开发和代码管理
3. ✓ 掌握 **RESTful** 的概念，掌握前后端分离式的开发
4. ✓ 掌握日志的使用
5. ✓ 掌握缓存的使用
6. ✓ 掌握 Redis 不同数据类型的用法
7. ✓ 掌握 Celery 异步任务处理
8. ✓ 掌握 Nginx 的配置，及负载均衡的原理
9. ✓ 了解数据库优化、分布式数据库及数据分片
10. ✓ 掌握数据库关系建模，及不使用外键如何构建关系
11. ✓ 掌握服务器异常处理，及报警处理
12. ✓ 熟练掌握常用 Linux 命令，以及初级 bash 脚本的开发
13. ✓ 掌握线上服务器的安装、部署
14. ✓ 对服务器架构、服务高可用等有一个初步认识

企业中的组织结构

- 股东
    - 董事会：董事会代表股东行事，聘用管理层来管理公司
    - 监事会：监事会监督董事会
- 管理层
    - 核心高层四个 O：CEO（总管，可能是老板，也可能是高级打工）、CFO（管钱）、COO（管市场运营）、CTO（管产品技术）
        - 总裁、CEO谁大
    - 其他高层：总裁、VP 副总、HRD
    - 中层：各部门总监、部门经理、负责人
    - 基层：经理、主程、Leader、主管、执行、程序员
- 人力资源部门：HRVP / HRD 负责
    - 制定用人制度，负责人员的流入流出，简言之：招聘
    - 制定绩效考核制度，审批薪酬表，简言之：薪资
    - 负责人力资源发展，简言之：培训
    - **每个求职者都要经过人力面试**

- 行政部门：COO负责
    - 日常办公、卫生管理, 会议、活动管理
    - 内部物品、设备的预算和购置
    - (内网、IT)
- 财务部门：CFO负责，一般 CFO 兼董秘（董事会秘书）
    - 资产管理、预算及成本管理、风险管控
    - 薪酬管理，税务、
    - 上市、财报管理
    - 投融资
- 市场部门：COO负责
    - 负责花钱，拉新用户
- 运营部门：COO负责
    - 负责做活动，留住用户
- 研发部门：CTO负责
    - CTO（方向、大战略、领军人物）
        - 前端技术总监（某个方向的带头人和负责人）
        - 后端总监
        - 编解码技术总监
        - 工程总监(真实业务系统的实施)
        - 技术副总（实施和实现）
    - 产品
        - 产品人员
        - 设计人员
    - 技术
        - 大前端开发：用户交互和展现
            - HTML5 (3~4人)、小程序
            - iOS (3~4人)需求较少
            - Android (3~4人)需求很多
        - 后端开发：更多是出API，给数据
            - Python / PHP / Java / Go (4~8人)
            - 架构
        - 运维：实施、监控、安全
            - SA：System Administrator（ Linux 占 95%，Windows 占 5% ）
            - 网络安全：
            - (DBA) 现在很少独立岗位了，只有银行、传统制造业信息中心，这样的传统公司有独立的 DBA 岗
        - 测试：大公司很多测试岗外包了，测试越来越成为一个专业的独立业务
            - 白盒测试
            - 黑盒测试 (1~2)
            - 接口测试
            - 压力测试
            - 自动化测试
        - 数据分析
            - 大数据分析

        - 为企业决策提供支撑
        - 算法岗（偏业务）：用 Excel、SQL 在工程岗搭建好的系统上，做统计分析
        - 工程岗（偏技术）：系统部署、ETL(数据抽取、转换、加载)
- 组织结构各个公司都有差别，面试的时候可以问
- 互联网公司运转的本质：
    - 拉新：市场负责、网络营销、增长黑客；
    - 留存、转化：产品、运营负责，产品运营设计方案，研发实现；
    - 付费、转化率：运营负责，设计方案，研发实现，
        - toC的，用户付费：前端付费；
        - toB的，广告主付费：后端付费；
- 成本中心
    - 研发、市场
- 效益中心
    - 运营
- 以上部门众多，跨部门沟通有壁垒，怎么办?
    - 打散部门壁垒，相关职能合并到一个项目组
    - 项目经理：
        - 产品、运营、美术
        - 技术：前端、后端、架构、运维、增长、数据合为一个 Team
- 面试题：
    - 你们公司多少人?
    - 你们组多少人，都负责什么工作?

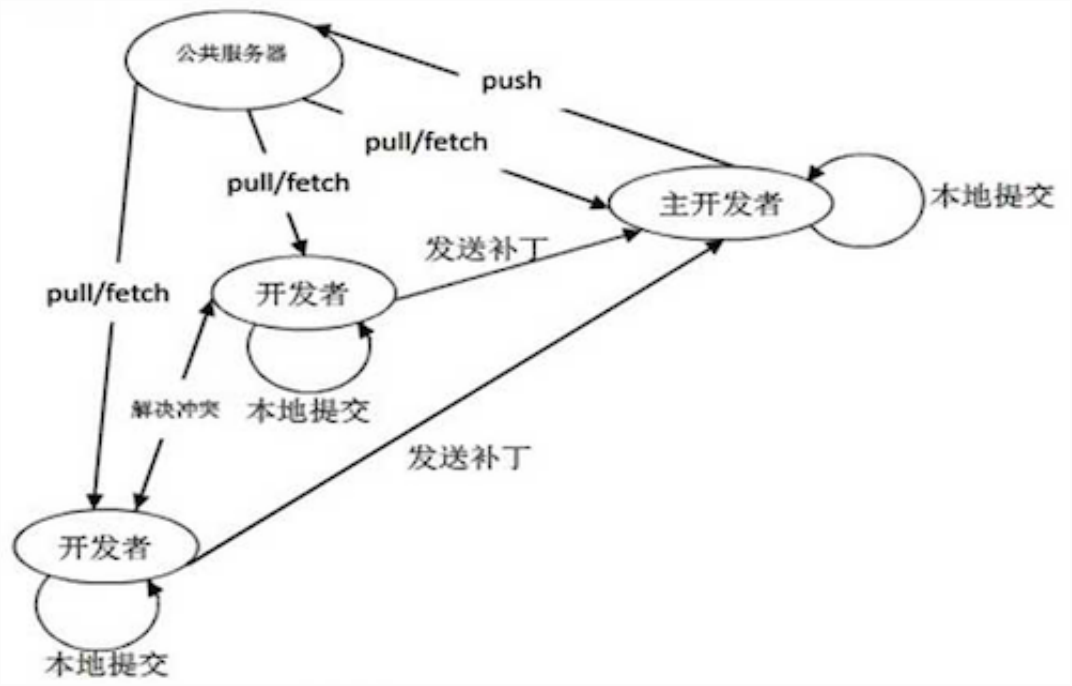
工作中的项目开发流程

1. **立项**：市场可行性分析，立项报告，立项会议
    1. 如果你去的公司，没有立项，直接开始，就要小心了。
    2. 推动立项
    3. 呆的越久越浪费时间，赶紧思考后路
2. 需求：讨论并明确大的需求，大家达成**理解一致**，项目**目标一致**，各自分工准备
3. 概要设计：产品人员进行原型设计，提出详细的开发需求，同步技术人员做概要设计：数据结构、数据库、基于团队的经验和能力选定框架
    1. 实验性项目：尝试一些新技术：PostgreSQL
    2. 有经验的架构师确定框架和基础库
4. 需求设计：详细需求讨论会，产品出原型图
5. 工作分工：分解需求，大体确定工期，阶段计划，迭代方式(Scrum)，迭代节奏，有详细设计文档
6. 以下进入迭代周期（Scrum流程，2-3周）：

1. PO：**确定**本期详细需求：这次迭代要完成哪些工作，需求不再变化，保证大家**理解一致**，宁可多花时间，达成一致
2. 每日站会：
    1. Master：组织：
    2. 每个人看白板分工：todo、doing、done
        1. **每日站会要说清楚三个事情：**
            1. 昨天我做了什么：
                1. 需求讨论、分析
                2. 设计：美术、技术
                3. 接口定义
                4. 搭建测试环境
                    1. 数据库搭建
                    2. 测试web环境搭建
                5. 框架搭建
                6. 模块设计
                7. **模块编码**
                8. 联调、测试
            2. 今天我计划做什么
            3. 我需要的外部支援，团队一起帮助我协商解决：
                1. 前置条件：等图、等需求、需要讨论确定
                2. 难度过大，自己搞不定，需要外部支持
                3. 有个前置条件还没弄清楚
                4. 顺利工作需要的外部条件：测试环境、需要xxx角色配合测试
    3. 燃尽图：未完成功能点的指示图
    4. 设计人员进行 UI、原画等绘制工作；同步技术做前期技术设计准备：框架、数据结构，分到具体模块的人思考一下如何设计
    5. 前端人员接收各种图形元素
    6. 前后端人员协作，做好接口标准约定，写**接口设计文档**，编写接口文档，后端主导写接口
    7. **工时最长的编码部分**：前后端同时开始开发各自功能
    8. 联调：前后端联合调试
    9. 部署到测试环境，测试人员测试，开发内部兼任测试
3. 上线部署(、服务重启)、新功能发布
4. **总结本次迭代优缺点**
5. 开始下一个迭代需求讨论
6. 可参考的流程方式：

    1. 瀑布式：需求分析、设计、编码（4-5个月）、测试、上线、完成
        1. 周期比较长，比如3-6月，6月-12个月
    2. XP：极限编程，特点是结对编程，对学习型任务，提高成员水平，帮助极大
    3. **Scrum**：目前主流采用的方法，上面讲的基本就是Scrum的迭代流程
        1. 周期缩短，每个短周期交付一小部分工作
        2. PO：product owner
        3. Master：团队迭代的管理者，tech leader 兼任
    4. Kanban：新兴方法
7. 合理项目的研发流程的时间占比：
    1. 需求分析、设计：1/3
    2. 编码：1/3
    3. 联调、测试，复盘：1/3
8. 不合理的项目流程：不具有可持续性
    1. 需求：一个人说了算，不占时间
    2. 编码：80%，自己挤出时间设计，写必要的文档
    3. 测试：20%
9. 交付的标准
    1. **订好验收标准**
        1. 交付的不是自己简单完成功能的模块
        2. 而是多方验证无误，达成需求的，健壮的模块
    2. 这会影响别人对你的预期，靠谱的预期
        1. 所以我们要**管理别人对我们的预期**
        2. 这取决于我们自己的行动
            1. 不要给别人过高的承诺
                1. 不要什么任务都接
            2. 承诺了就要尽力达到
            3. 尽力达成比别人的标准更高一些
10. 实际上，项目时间都是弹性的
    1. 技术是能**争取来时间做分析和设计**的
11. 面试题：
    1. 你们怎么开发协作?

GIT 命令回顾



- `init` : 在本地创建一个新的库，与 clone 互斥
- `clone` : 从服务器克隆代码到本地 (将所有代码下载)，与 init 互斥
- `status` : 查看当前代码库的状态
- `add` : 将本地文件添加到暂存区
- `commit` : 将代码提交到本地仓库
- `push` : 将本地代码推送到远程仓库
- `pull` : 将远程仓库的代码拉取到本地 (只更新与本地不一样的代码)
- `branch` : 分支管理
- `checkout` : 切换分支 / 代码回滚 / 代码还原
- `merge` : 合并分支
- `log` : 查看提交历史
- `diff` : 差异对比
- `remote` : 远程库管理
- `.gitignore` : 一个特殊文件, 用来记录需要忽略哪些文件
- ssh-key 的使用

1. github操作
    1. 登录 github.com
        1. 注册、验证邮箱
    2. 点 + 号，创建新项目
        1. public
        2. gitignore：python
        3. license: MIT
        4. create Readme.md
    3. copy 项目地址：
2. 本地操作
    1. 打开命令行窗口
    2. cd到自己的工作目录

3. git **clone** https://github.com/yuzebin/bj-py1903.git
    1. gcl
4. 修改readme.md文件，增加一点内容
5. git add .
6. git commit -m "some change"
7. git push
8. git branch
9. git branch --create develop
10. git checkout develop
11. 修改 readme.md，增加在分支中的内容
12. git add .
13. git commit -m "branch changes"
14. ? git push
    1. fatal: 当前分支 develop 没有对应的上游分支。 为推送当前分支并建立与远程上游的跟踪，使用

       git push --set-upstream origin develop
15. git push --set-upstream origin develop
16. git push
17. git status
    1. gst

3. 组织操作
    1. 打开命令行窗口
    2. cd到自己的工作目录
    3. git clone 本小组的代码
    4. 修改readme.md文件，增加一点内容
    5. git branch # 查看本小组代码分支
    6. **git branch --create branch_name** # 创建新分支
    7. **git checkout branch_name**
    8. 修改 readme.md，增加在分支中的内容
    9. git add .
    10. git commit -m "branch branch_name changes"
    11. ? git push
        1. fatal: 当前分支 develop 没有对应的上游分支。 为推送当前分支并建立与远程上游的跟踪，使用

```
git push --set-upstream origin branch_name
```

12. git push --set-upstream origin branch_name

4. Github 操作
    1. 到 pull request 页新建 pull request
    2. 选好目的分支，源分支
    3. 选好审核人
    4. 新建 pull request
    5. 审核人去审核，合并
5. 面试题：
    1. 代码冲突怎么办?
    2. merge还是rebase?

GITHUB 常用操作

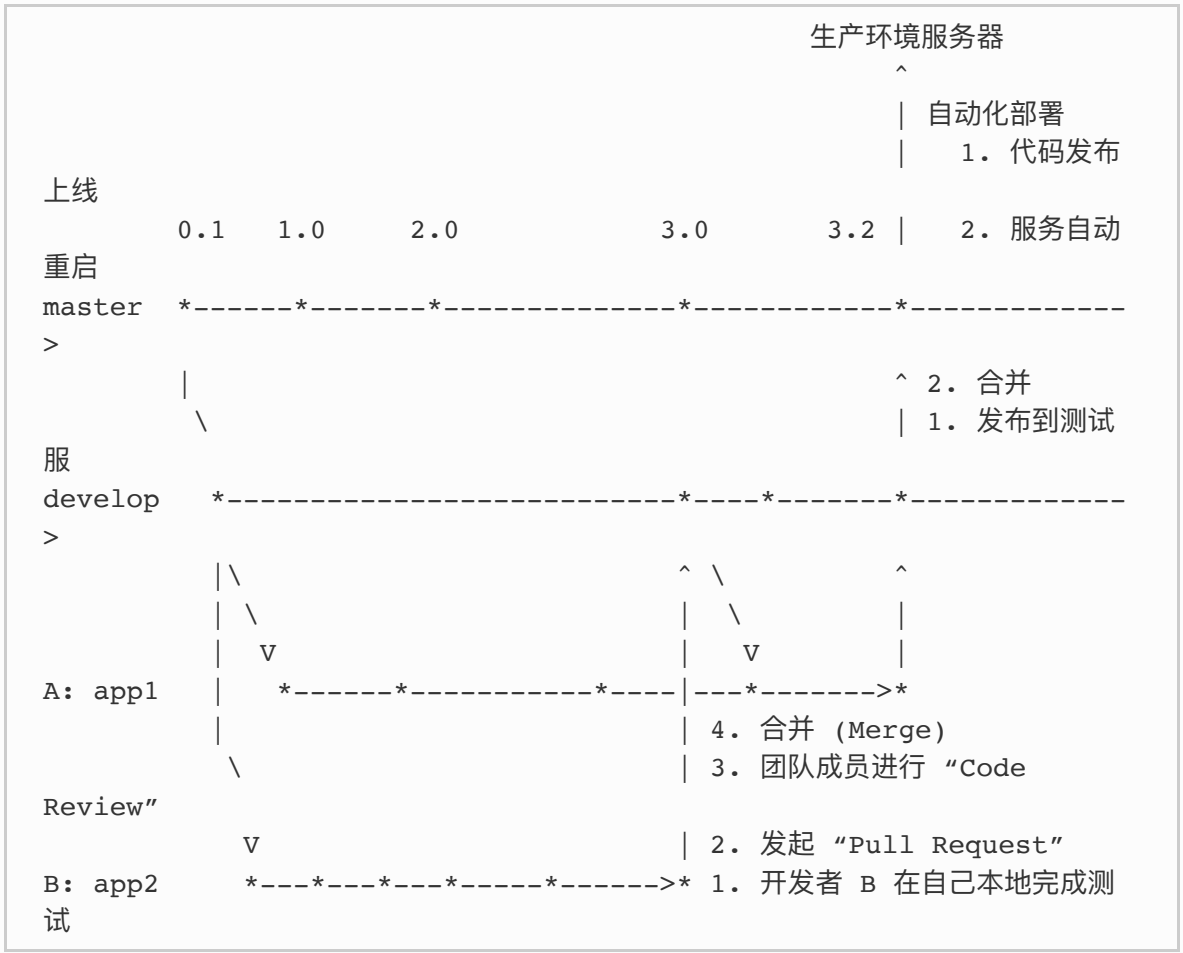
- 注册
- 登录
- 创建项目
- 私有、公开
- gitignore
- license
- 常规操作
    - wiki
    - issue
    - release
- 创建组织
    - 加入组织

GIT 分支管理简介

- master
- develop
- 常见分支风格
    - git flow
    - github flow
    - gitlab flow

GITHUB FLOW

1. 版本控制及代码管理
    - 分支类型
        - master: 主干分支，代码经过严格测试，最稳定，可以随时上线
        - develop: 开发分支，合并了各个开发者最新完成的功能，经过了初步测试，没有明显 BUG
        - feature: 功能分支，开发中的状态，代码最不稳定，开发完成后需要合并到 develop 分支
    - Pull Request: 拉取请求
        - 开发者自己提交 Pull Request 通知团队成员来拉取合并自己提交的代码。
        - 通过此方式可以将合并过程暴露给团队成员，让代码在合并之前可以被团队其他成员审核，保证代码质量。
    - Code Review: 代码审核
        - 代码逻辑问题
        - 算法问题
        - 错误的使用方式
        - 代码风格及规范化问题
        - **学习其他人的优秀代码**
2. 上线流程介绍



作业：项目阶段开发流程及要求

1. 两人一组，结组编程，每组不要超过三人
2. 每组选一人作为组长，由组长在 Github 上创建自己的组和项目
3. 组长分配任务，各自开发自己的功能
4. 开发过程中注意编码规范：PEP8，力求做到 "团队代码如同一人编写"
5. 每个人为接到的功能创建一个独立的分支
6. 开发、提交、审核、合并、上线