2020.09.13

1. 基于github的协作流程简介
    a. **目前**的管理方式并不足够稳定，可能会随软件质量与改进课程推进发生变化
    b. 之前的流程中存在的问题：
        i. 缺少分支管理和版本控制
            1. 只有一个分支，或者只有一个master一个dev
            2. 语义不清晰，无法进行需求跟踪和状态管理
            3. 不利于容错和回滚
        ii. 粒度过细的分支（如以用例为单位的分支组织）
            1. 频繁的、无意义的分支操作带来的损耗
            2. 开发人员的厌倦感和抵触情绪
        iii. 不规范的提交信息和issue
            1. 不利于快速发现并解决问题
            2. 不利于跟踪
        iv. 没有利用pr，多人共用同一仓库（乃至同一分支）
            1. 增加冲突的可能性，影响开发效率
                a. 需要时不时地pull，检查自身的工作是否与可能已被他人修改的基线冲突
            2. 无法量化跟踪项目进度
                a. 功能是否完成?
            3. 缺少必要的质量保障手段
                a. 不符合规范的代码也可以轻易地并入基线，没有额外检查
    c. 主仓库的目的应当是充当开发的基线
    d. 状态跟踪
        i. project：整体状态跟踪
        ii. milestone：每个项目内部的工作计划
        iii. 可能存在冗余?
    e. issue
        i. issue templates
    f. pull request
        i. fork后在自己的仓库进行开发
        ii. review
2. CI/CD流程简介
    a. 基本流程
        i. 测试
        ii. 构建
        iii. 打包
        iv. 部署
    b. 目前的challenge
        i. mono repo || multi repo?
            1. 更复杂的CI/CD || 更复杂的项目管理
            2. 目前是打算先用mono repo
                a. 通过分支控制，当然这意味着大量的脚本编写工作

    b. 后面如果有问题（如，构建时间过长，项目结构不清晰）再重构
3. 现阶段目标：最小化构建、最小化交付
    a. 总时间：8个sprint
    b. 预计完成全部目标周期：2 – 3个sprint
        i. 理想情况是能在10月4日左右完成（即第2个sprint结束时）
        ii. 最坏情况是在10月18日之前**必须**完成（即第3个sprint结束时）
        iii. 一般情况是在期间完成
            1. 最坏情况下可能会影响后续进度
            2. 现阶段目标只是完成一个最小化的构建，类似于原型开发
        iv. 可能的影响因素：
            1. 中秋、国庆假期（可能无法顺利在第2个sprint时完成）
            2. 下周本人有事，会影响第1个sprint的进度
    c. 简化后的任务
        i. 数据处理：
            1. 先不考虑数据异构和同名消歧
                a. 此简化仅影响数据表现，可以后期清洗
                b. 先进行**单数据源**、**不消歧**的数据开发
                    i. 但是需要**针对不同的实体服务提供不同的数据**
                    ii. 先熟悉数据处理相关任务的开发方式和常见坑点
                    iii. 可以继承此前软工三的IEEE爬虫代码
                    iv. 可以咨询zwq
                c. 项目结构视情况而定，可以后期重构
                d. **现阶段**不太需要CI / CD
                    i. 不一定有按需导入数据的需求，只需要定时向实体服务注入符合要求的数据
                    ii. 不一定需要封装成服务
                    iii. 定时任务（crontab等）即可，或者手动注入
                e. 数据规模：
                    i. 领域：软件工程
                    ii. 2个A级会议及2个A级期刊
                        1. TSE（IEEE，期刊）
                        2. TOSEM（ACM，期刊）
                        3. ASE（IEEE / ACM，会议）
                        4. ICSE（IEEE / ACM，会议）
                    iii. 近10年
                    iv. 优先选择 **IEEE** 的数据
        ii. 展示层：
            1. 简化界面
                a. 简单的中后台数据面板即可，并不需要太多交互操作
                b. **暂时**不考虑搜索等额外功能
                    i. 如果有查询需求，先用URL访问
                c. 快速原型开发
                d. **暂时**不考虑人机交互的易用性
                    i. 随着人机交互课程推进继续改进
        iii. 服务库存：
            1. **自顶向下**的构建方式
                a. 基于流程的构建方式
                b. 利于对需求的理解
                c. 利于变更（现阶段需求还不够稳定）

    d. 敏捷交付：每次sprint结束（或者某个阶段结束时），对现有的设计和服务进行回顾
  2. 领域（同时也是基础的实体服务）：
    a. 学者
    b. 论文
    c. 关键词
    d. 机构
    e. 会议
    f. 期刊
  3. 任务服务（待定）
  4. 对外接口（待定）
4. 架构设计确认
  a. 目标是最小化构建、最小化交付，暂不考虑QoS层和服务发现、服务治理
    i. 先进行点对点调用
  b. 前后端分离
    i. ~~实验性想法：多个spring MVC组合？然后在网关层进行路由隔离？~~
  c. 展示层
  d. facade层（网关层）
  e. 任务服务层
  f. 实体服务层
  g. 数据层
5. 开始编码
  a. 尽可能多关注github，任务和项目进度会在上面以issue和pr的形式进行管理
  b. 站立会议，每天或每两天交流一次进度和遇到的问题
  c. **冲**
6. 最后的废话
  a. 要复用，更要合理复用
    i. 基本原则是“领域”：这两个东西是一个领域的吗？不是就不要复用，哪怕长得完全一样；因为无法预料他在未来会不会变成一个完全不同的东西。
  b. 适当的冗余是可以接受的
    i. 但不代表放弃复用
  c. 不要过度设计
    i. 俗称想太多，考虑超出范围（能力、项目等）的内容
    ii. 完成现阶段的任务是第一目标
    iii. 但不代表完全不思考直接搬砖，如果发现不对（或者只是感觉不对）请立刻反馈，以便在后续迭代进行修复；“越早发现和修复问题，成本越低”。
  d. 不要图一时方便而做出过于短视的设计（和修复）
    i. 重构和下一迭代的时候倒霉的是自己
    ii. 典型行为：不写（必要的）注释、不写（必要的）文档