20190912 第二天学习总结

一、TAF平台发布流程

1、登录平台

2、运维工具 =》服务管理 =》部署申请

- 选择应用、填写服务名、选择服务类型
  - 服务名称必须与打包的项目的目录名称一致，否则发布时会出现以下错误"verify failed"错误
- 提交后填写服务节点信息
  - 端口需要自动生成，否则容易冲突，同时也可以端口校验填写的端口是否有冲突

3、服务管理 =》应用/服务 =》发布管理

- 上传发布包，选择打包好的项目tgz压缩包，填写描述，提交发布

4、服务管理 =》应用/服务 =》服务管理

- 点击节点中的IP地址打开Logview本地日志，点击提交查询按钮查看服务是否启动正常；
- 可以访问 `http://ip:port` 测试是否可以打开；

5、业务配置 =》Base配置 =》测试域名配置

- 点击添加，填写自定义的域名，必须是 `test.whup.com` 下的子域名，IP、端口则是配置服务节点时填写的IP和端口；
- 配置完成，过个几分钟即可通过域名访问发布的服务，比如：`http://stdbao.test.whup.com`

6、注意事项

- 每次上传发布包后，需要重启服务：服务管理=》应用/服务=》服务管理
- jce文件中 `module` 对应服务管理中的应用名称
- 安装 `jce2node` 工具

```
# 下载 up 开发工具包
npm i -g @up/oem-cli
# 编辑 .profile 文件，添加别名
alias jce2node="up jce2node "
# 生成服务端
jce2node Hello.jce --server
# 生成客户端
jce2node Hello.jce --client
```

- 配置的应用名、服务名、servant要与项目中的内容保存一致；

二、nodejs学习一

1、第一章 Node简介

1.1、web服务器高性能的几个要点：事件驱动、非阻塞I/O；

1.2、有些服务器使用了阻塞I/O的库，所有达不到见到更有效；

1.3、选择javascript的理由

- 开发门槛低
- 没有后端市场，导入非阻塞I/O没有压力，没有什么影响；
- 浏览器中广泛使用事件驱动；

1.4、单线程弱点

- 无法利用多核CPU
- 错误会导致整个应用退出
- 大量计算应用占用CPU导致无法调用异步I/O
- 解决方案：
    - Google的gears：启用单独进程，负责计算程序执行，得出结果后通过事件传递给主线程；
    - HTML5的WebWorkers：创建工作线程计算，通过消息传递结果，保持应用的简单低依赖；
    - node采用同H5思路的child_process

1.5、node应用场景

- I/O密集型，利用事件循环的处理能力，非一个线程一个请求，减少资源占用；
- CPU密集型，借助V8引擎，同样也可以处理CPU密集型业务；

2、第二章 模块机制

2.1、CommonJS规范

- JavaScript规范薄弱：没有模块系统、标准库较少、没有标准接口、缺乏包管理系统；
- node弥补了JavaScript的缺陷的同时，可以编写：服务端应用、命令行工具、桌面应用、混合应用等；
- commonjs规范：模块引用、模块定义、模块标识

2.2、node模块实现

- require原理：路径分析、文件定位、编译执行
- 模块分类：核心模块、文件模块，核心模块优先于文件模块；
- require对相同模块二次加载都采用缓存（编译执行后的对象）优先的方式；
- 模块路径的生成规则
    - 当前目录下的node_modules目录
    - 父目录下的node_modules目录
    - 父目录下父目录下的node_modules目录
    - 沿路径向上逐级递归，知道根目录下的node_modules目录
- 文件定位：除js外指定扩展名、同步配合缓存、require是指定目录下的文件

2.3、模块编译

- 每个模块文件中存在：require、exports、module三个变量，以及`__filename`、`__dirname`两个变量；
- 编译时，node对文件内容头部包装：`(function(exports,require,module,__filename,__dirname){/* code */})`

2.4、主要包描述package.json

- name包名、description包描述、version包版本、dependencies依赖包列表（require导入的依赖）、devDependencies、scripts执行npm run命令的地方

2.5、npm命令

- npm init初始化项目、npm install安装依赖、npm run执行package.json中scripts配置的命令（npm钩子命令）、npm config配置npm

3、第三章 异步I/O

3.1、异步I/O解决的问题

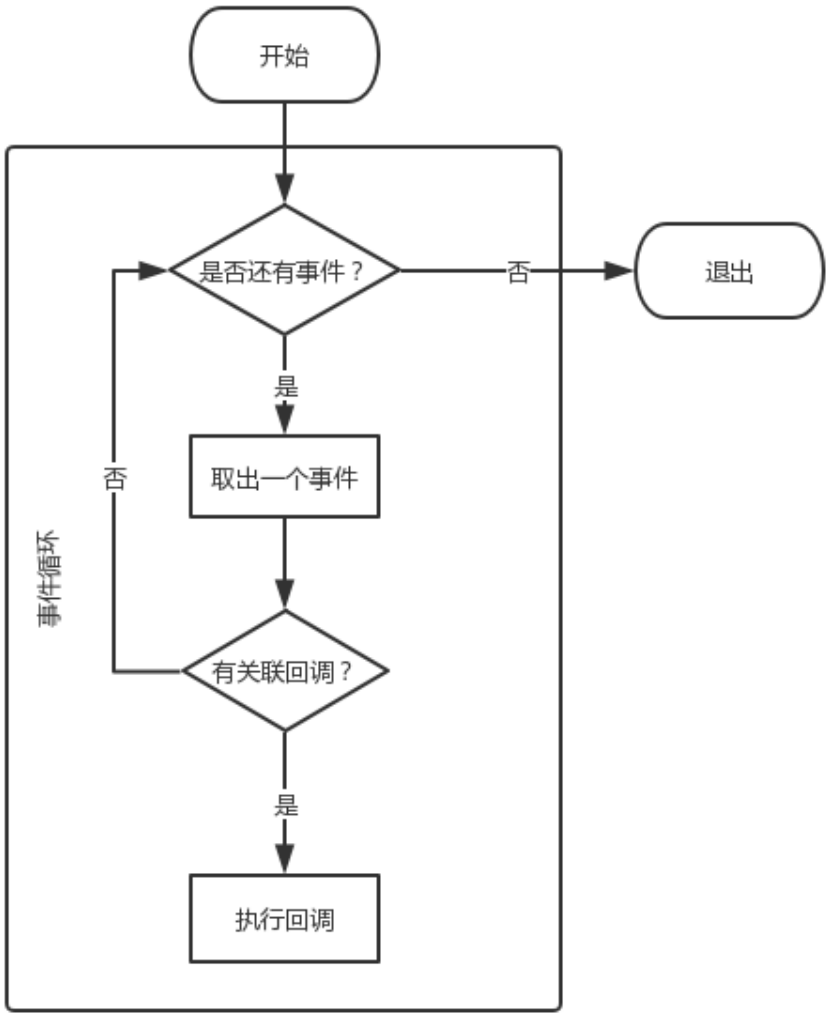
- 用户体验：浏览器js执行同ui渲染一个线程，脚本执行超过100ms，页面出现卡顿感觉；异步I/O则异步执行脚本，不影响UI渲染；
- 资源分配：单线程阻塞I/O资源得不到更优利用，多线程死锁、状态同步问题；

3.2、异步I/O实现

- 非阻塞I/O轮询实现：
    - read：重复调用检查、CPU等待，对原始、性能最低
    - select：在read基础上添加文件描述符上的事件状态判断，1024长度数组存储状态
    - poll：对select改进，采用链表的方式避免数组长度限制，可以避免不必要的检查，但当文件描述符较多的情况下，性能低下；
    - epoll：Linux下高效率的I/O事件通知机制，轮询时没有检查到I/O事件则进入休眠，知道事件发生将其唤醒，利用了事件通知、执行回调的方式，而不是遍历查询；
    - kqueue：与epoll类似，FreeBSD系统下存在
    - aio解决休眠期间CPU限制的问题，但只Linux下有，无法利用系统缓存

3.3、node的异步I/O

- 事件循环



- 观察者：每个事件循环中存在一个或多个观察者，而判断是否有事件要处理的过程就是想这些观察者询问是否有要处理的事件；
- 事件循环是一个典型的生产者-消费者模型；

3.4、非I/O的异步API

- setTimeout/setInterval定时器：某一此循环占用时间长，则下次循环可能超时了；
- process.nextTick：解决定时器精确度不够、使用了红黑树的问题，相对比较轻量；
- setImmediate：与process.nextTick类似，process.nextTick回调函数存放在数组中，setImmediate回调存放在俩表中；每次轮询nextTick执行所有回调，setImmediate执行一个回调；

三、第三天（20190916）学习计划

1、nodejs链接pdf书 4、7、8章