本文档描述了，服务器基本的数据结构和使用到技术等。

0 用户登录

用户的名字， 选择的基地编号都将放到session。

1, 基地

1.1）获取基地内的实体

url: <base>/<entity> base: 基地名字； entity:职员，飞机等。

基地内的实体信息存放到单独的collection， 每一个实体都有一个“base”字段表明属于那个基地

基地的结构为：

{

code: 编号（主键），

name: string,

description:option 相关描述信息可能是个文档，

address: 基地地址

…其他的信息

}

1.2)操作

1）更新：

说明：用户选择基地后，基地的code存session的原因：更新的时候，服务端如果不记录用户登录的基地code, 服务器端在更新数据的时候，就不得不以来客户段传来的code，在更新数据；如果客户端修改code的话（code参数以明文的形式post到服务器）， 数据库将执行db .base.update({“code”:request.form.code}, {info})的时候可能会产生数据被覆盖的现象：数据库有记录 {code:”a”, …} {code:”b”, …}.

用户code:”a”的时候把“a”改成了“b”， 那么最后更新后，“a”的记录将不会被更新，而“b”的记录将会被更新覆盖掉。 为了避免这种事情：约定只能修改登录基地的信息，而不能修改其他基地的信息。另外管理员的权限将被分为show\_priv, add\_priv, update\_priv三类权限字典(参考点mysql 用户管理)。例如update\_pri = {“base”:[“a”,]}，表示该用户可以更新code是“a”的基地的基本信息。实际URL路由中尽量将code作为URL的一部分发送到服务器。

2，员工

* 1. 管理员， 管理单独拿出来一个collection。

管理员可能每个部门只有一个或两个人，单独管理感觉比较安全和维护一些。

管理员的除了是员工外，还有权限的管理部分，普通的员工全部是普通的权限--查看。

1） 属性

name: email, # 使用email作为用户名字

userNumber：员工的编号,

email: 邮箱，#常用的email， 和name的区别是email的值是可以修改的。

base: 所属基地名字，

password: 密码,

begin: 入职时间,

end： 离职时间,

role: 职位，

telephone: 手机电话，

department：部门，

description: 描述，

status: 状态（离职，在职等等），枚举类型。

}

2） 操作

a）查看所有的员工:

/stuff

b) 查看某基地下的所有员工：

/<base>/stuff = /stuff/<base>

c)查看某基地下指定的员工:

/<base>/<entity:stuff>/<name> = /stuff/<base>/<name>

d)添加员工, 更新员工:

# /<base>/<entity:stuff>/<name>/<operator>

/stuff/<operator> 现阶段有添加 /stuff/add， 更新/stuff/update，没有删除 ，

这里的操作，都不允许跨基地的操作， 比如只能查看登录基地的员工信息。与更新基地信息相同的原因，更新不允许修改主键（但不保证该值不会被修改）name (email)的值。

在保证用户登录安全性的条件下，服务器段将完全相信客户端的修改，更新信息，因此数据录入“正确性” 的保证与用户有关。比如name。如果输入错误的了name了，需要删除重新建立一个账户。

备注:这个操作需要权限的检测的。登录者需要跨基地的检索权限。

3， 飞机航班

1） 属性描述

airway =

{

aircraft: 机型,

code: 航班号码（航空公司规定的号），

seats: {} 座位信息，

base: 基地名字，

depart: 起飞时间，

depart\_airport: 起飞机场编码，

arrive：降落时间，

arrive\_airport: 降落机场编码,

schedule：班期,

from: 离开城市的名字,

to: 到达城市的名字，

stops: [], 停靠城市

}

2） 操作与stuff相同。

3， 任务

* + 1. 属性

task = {

id: 任务的ID

base: 基地，

owner: 任务当前有那名员工处理,

createDate: 该任务创建的日期，

start: 开始时间

endtime：结束时间

task:具体任务描述，

history:[{stuff:..., start:..., endtime:...}, {}] 任务历史记录

contents:[{“username”:xxxx, “content”:xxxx}, …]} 对问题的讨论和说明

}

2）操作

添加删除更新操作于前面类似。

1), 任务的分配和转接, 任务的评论

url: /task/<base>/<taskid> method:get(查看), post（更新和局部更新）

5） 设备：

1） 属性：

{

“name”: 设备的名字或编码，

“airway”: 所属于航班，

describe:[1, 2,3 ] 设备描述，设备可能使用时间很长因此这个描述可能会很长

history :[h1, h2...] 设备的历史信息，历史信息保存时限。

“status'”: [], 状态，（状态是人为填写？设备主动向服务器注册状态变化？）

“slave“：依附与主体的附件

}

2 工具和插件：

1，数据库的链接和管理的模块（这一块可能会加入modell的一些东西）。

2， flaskext.wtf来处理form，数据的表单验证交与form处理。

3，客户端jquery及其插件，bootstrap等吧

3, 几个模块：

1， modell： 数据定义，数据的一些基本操作。

2， 多线程同步控制装饰器， 和thread-local 的数据modell（防止数据覆盖）， 这些基本上都参考了一些werkzeug的东西。

3， 简单的数据库加载模块, 完成数据库的连接， 还没有数据库的管理部分。

4 部署：

nginx + fcgi + flup. 还没有使用memcache技术。配置部分根据需要另外出文档吧。

对Mongodb数据库的配置和管理没有做太多考虑， 直接采用默认配置。

5， 存在的问题和不足：

1，route部分改了几次，感觉越该越乱了。

2，部分模块的封装简单，处于比较初级阶段。

3， 。。。。。。想起来再补充。