文档基本描述了，服务器使用的框架和基本技术。

1, 基地

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

基地的结构为：

{

name: string 主键,

description:option 相关描述信息是个文档{}，

# stuff:{} base中不保存职员的缩略信息， 职员信息由<base>/stuff 获取

获取所有基地下的职员信息url: /stuff

}

* + 1. 操作

1）更新：

客户端做不做表单的验证都可。

更新的时候，服务端不会记录用户当前更新基地的name, 如果客户端能修改name的话（name参数以明文的形式post到服务器）， 数据库将执行db .base.update({“name”:request.form.name}, {info})的时候可能会产生让人愤怒的事情：

数据库有记录 {name:”a”, …} {name:”b”, …}

用户不小心name:”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：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”,]}， 那么可以可以解决一部分的更新覆盖问题。

另外参考了一下豆瓣修改帖子的方式： [http://www.douban.com/group/topic/](http://www.douban.com/group/topic/28571151/)28571151/，

主题id是可以修改。但是只能修改该用户可以修改的帖子，不能修改的会报权限错误。

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, # 常用的email，主键

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

c)查看某基地下指定的员工:

/<base>/<entity:stuff>/<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 ， 不使用/<base>/stuff/<operator> 的原因是添加更新的form中会有基地的信息，另外一般不允许变更员工所属基地。其中更新不允许修改主键name (email)的值。原因与基地部分相同。如果录入出错，只能删除后添加新email，不再允许更新主键。 后面大致都是一样的处理。

f) 查看指定员工的信息:

/stuff/<name>

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。 这个权限的检测是登录用户权限和查看用户的一些信息的匹配，不如权限中基地是否包含用户所在的基地等等吧。

3， 飞机航班

1） 属性描述

airway =

{

aircraft: 机型,

name: 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 = {

base: 基地，

stuff: 任务当前有那名员工处理,

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

discuss:string， 对问题的讨论和说明，

timeline:[stuff1, stuff2, …] 记录任务经过那些人

}

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

url: /task/move

2）任务的说明添加归结与对任务的update

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， mode：简单的，数据和操作在一起的类。现在还在学习如何在flask引入mode，

实际上，这一块可能会被放弃。

2，数据库的链接和管理的模块（这一块可能会加入mode的一些东西）。

3， 开始使用flaskext.wtf改写过去的与form相关的代码，解决服务器端对form的验证方式。

4， flaskext插件将被使用。

5，客户端jquery及其插件，bootstrap等吧