**Data access layer**

主要是对非原始数据（数据库或者文本文件等存放数据的形式）的操作层，而不是指原始数据，也就是说，是对数据库的操作，而不是数据，具体为业务逻辑层或表示层提供数据服务。

1. 读取csv

1.1 读取文件的所有内容

**Business Logic Layer**

主要是针对具体的问题的操作，也可以理解成对数据层的操作，对数据业务逻辑处理，如果说数据层是积木，那逻辑层就是对这些积木的搭建。

**~~1. Read~~**

~~1.1 生成文件路径列表[seq.csv setting.ini courseInfo.csv courseInfo.csv courseInfo.csv]~~

~~1.1.1 interior~~

~~[setting.ini courseInfo.csv courseInfo.csv courseInfo.csv]~~

~~1.1.2 persistence~~

~~[seq.csv detail.csv randomdetail.csv sum.csv]~~

~~1.2 读取动态文件名 (filename, teacher\_id, course\_id) [detail randomdetail.csv sum]~~

~~1.3 读取固定文件名 (filename)~~

**~~2. Write~~**

~~2.1 写入动态文件名 (data, filename, teacher\_id, course\_id)~~

~~2.2 写入固定文件名 (data, filename)~~

**~~3. Append~~**

**Csvkit Tutorial**

https://docs.python.org/2.7/library/csv.html#module-csv

https://docs.python.org/3.6/library/csv.html#module-csv

1. import csvkit

1.1

**PRE**

1. isSucc三种状态

针对学生考勤提交的特征信息内容，通过系统内部的考勤识别机制可根据其识别、认证的结果成功与否来响应本次学生提交的特征信息是否有效

1）true: 认证成功，即进行3，并修改isSucc状态

2）false: 认证失败，不可进行3，仅修改isSucc状态

3）空: 未参与

2. 出勤有效时间界定机制(starttime, checkinTime)

~~1）分类~~

~~2）聚类~~

3）2分钟内

3. 考勤结果算法

在学生考勤时，根据late checkinResult和late checkinTime，计算最终的checkinResult

根据2的出勤有效阶段判定机制: ^在出勤有效时间范围内 ^^不在

Detail.csv中学生的考勤状态: 最新的状态

1）出勤: ^keep ^^迟到

2）缺勤: ^迟到 ^^keep

3）迟到: ^keep ^^早退

4）早退: ^迟到 ^^keep

5） 空: ^keep ^^迟到

~~6）请假~~

**因此，学生考勤的过程为 1 -> 3**

**0. 辅助**

0.1 获取模块

0.1.1 获取本次考勤次序号 (course\_id）

0.1.2 获取teacher\_id (wechat\_id)

0.1.3 获取teacher\_id (course\_id)

0.1.4 获取student\_id (wechat\_Id)

0.1.5 获取学生所有课程号 (wechat\_id)

0.1.6 获取教师所有课程号 (wechat\_id)

0.1.7 获取班级名列表（course\_id）

0.1.8 获取timewindow (url)

0.1.10 获取当前时间 ()

0.2 初始化detail.csv (course\_id)

获取当前课程的所有学号，checkintype为auto，其余的为空

0.3 初始化randomdetail.csv(course\_id, random\_list)

1）若没有此teacherid\_random.csv，则根据0.1.2来创建新文件，添加header信息

2）**清空除header之外的信息（仅用一次，结束后需更新至detail.csv）**

3）**根据random\_list为stuid赋值，将checkintype置auto，其余为空**

工号\_课程号\_考勤序号\_randomdetail.csv

[StuID,checkinTime,ProofPath,checkinType,IsSucc,checkinResult]

0.4 空太多（course\_id）

1）读取detail.csv

2）若checkinResult的空状态的数量超过总体的百分之80，则此次考勤无效。

0.5 TIMER计算考勤结果（wechat\_id, course\_id，timer\_list）

以detail.csv为主线，按学号依次计算

1）检验：4.7（此处特殊，存在才能继续）

2）若0.4，删除该detail.csv，random.csv ，停止。

3）0.7

**4）提取detail.csv的checkinresult，结果更新至sum.csv （方便查询）**

0.6 计算考勤结果（course\_id）

针对该course\_id课程：

1）检验：4.7 至少存在一次考勤记录 是否有请假记录

2）遍历teacherid\_lea.csv的内容，若存在与course\_id相关的请假记录，若教师给予请假，并将请假状态更新至相应的detail.csv文件，若不给予请假则不更新

3）遍历course\_id下的所有detail.csv，并将考勤结果写入sum.csv

4）若教师忽略批假则将对应学生的考勤状态置在sum.csv中置为“空”，直到教师处理该学生的假条

0.7 合并考勤结果 (course\_id)

1）获取源url与目标url格式[0.1.3 course\_id 0.1.1]

2） 根据randomdetail.csv更新detail文件的内容，若checkinresult不同，则以random.csv为主，即新的checkinresult覆盖detail.csv

0.8 生成random列表(nums)

0.9 格式化统计结果(course\_id) ,打印出此课程对应每个人的考勤状况，每节课出勤率，平均出勤率

**6. 外部系统**

6.1 请假 (student\_id, course\_id, seq\_id，lea)

1）假条识别系统（lea）

2）系统成功识别后，将【student\_id, course\_id, seq\_id】存入teacherid\_lea.csv

**1. 检验**

1.1 课程号检验（wechat\_id, course\_id）

1.2 抽点人数检验 (num)

1.3 考勤状态检验（type）

1.4 考勤次序号检验（seq） 是否存在

1.5 课程号学号检验（stu\_id）

1.6 教师微信号检验 (wechat\_id)

1.7 学生微信号检验 (wechat\_id)

1.8 学生是否可进行考勤 (wechat\_id)

通过0.1.5获得学生所有的course\_id调用4.9

**2. 教师**

2.0 查看我的course\_id (wechat\_id)

1）检验：1.6

2）0.1.6

2.1 自助考勤（wechat\_id，course\_id，~~timer\_list~~） 教师入队是互斥的--threading.RLock

1）检验：1.1 1.6

2）互斥的进入4.5

3）维护detail.csv

4）维护seq.csv

2.2 抽点（wechat\_id,  ~~timer\_list~~） **（在时间窗口自动检测courseid且支持多次开启抽点）**

1）检验： 1.6 4.7 当前时间小于4.8-int(5\*60)

2）若教师再次开启抽点，则0.7(0.1.3)

3）输入抽点人数1.2 初始化0.3(5.0, 生成random列表)

4）维护seq.csv

2.3 手工（wechat\_id，course\_id）

1）检验：1.1 1.6 非4.7

2）维护detail.csv，默认全部出勤

3）教师可输入学号1.5和状态1.3，

4）更新detail 自动获取seq 调用0.1 维护seq.csv

2.4 修改（wechat\_id，course\_id，seq\_id）

1）检验：1.1 1.4 1.6 非4.7

2）输入学号1.5+状态1.3，

3）更新detail.csv

~~3）更新sum.csv~~

2.5 汇总 (wechat\_id, course\_id)

1）检验：1.6 非4.7

2）0.6 ~~想要汇总某个课程，必须先把所有课程假条处理完毕~~

3）0.9

2.6 查看最近 (wechat\_id) 传入后，显示

1）检验：1.6

2）直接seq中的信息，获取相关url得到detail

3）统计：出勤率，缺勤率等

**3. 学生**

3.0 查看我的course\_id (wechat\_id)

1）检验：1.7

2）0.1.5

3.1 考勤 (wechat\_id)

1）检验：1.6 1.7 1.8

2）上传特征信息

3）参考开头的pre

3.2 请假 (wechat\_id) [注]可以帮忙请假,因此此处的wechat\_id不具有完全的充分性

1）检验：1.6 1.7 1.8

2）上假条lea

3）6.1

3.3 查看正在进行的考勤 (wechat\_id, timer\_list)

1）检验： 1.7 1.8

2）根据timer\_list中对应的course\_Id, detail.csv、randomdetail.csv(优先)相关内容

3.4 查看历史考勤 (wechat\_id，course\_id)

1）检验：1.7

2）根据course\_Id, 查询sum.csv相关内容

**4. Timer**  
 TCB=[wechat\_id ,course\_id, sectime, endtime,class\_list[]]

Timer\_list=[] 存放TCB **~~类的全局变量~~ 顶层的全局变量**

4.1 获取sectime (url)上课区间包括开始和结束

1）获取当前时间，获取sectime列表。

2）若当前时间，小于sectime列表第一个或大于其最后一个，返回sec1

3）根据sec1,2,3匹配，若起始sec<=若当前时间<结束sec, 返回对应secn。 否则：若当时间小于最后一节课的起始sec，若并且当前时间大于等于secn的结束时间并且小于secn+1的开始时间，返回secn+1 否则，返回sec\_last

4.3 初始化tcb(wechat\_id, course\_id)

[wechat\_id , course\_id, 4.1, 0.1.6]

4.4 进队检测与处理（~~wechat\_id ,~~ course\_id）

1）若timer\_list空，返回true

2）根据course\_id获取班级名列表（4.3）, 获取new\_sectime（4.1）

3）获取的班级名列表和timer\_list中的每个TCB[]中的class\_list[]依次进行查重操作

Set(list1) & Set(list2),若没有重复，返回true。 否则，获取出现重复班级名的sectime与new\_sectime对比，若相同，返回false并提示失败原因。若不同，返回true并且获取出现重复班级名的course\_id, TCB出队(4.6)

4.5 进队（wechat\_id, course\_id）

1）若可进（4.4），

2）4.3

3）TCB进队，threading.Timer(0.1.7, 4.6, ("msg1","msg2"))

[注] threading.**Timer**(1, self.depart\_queque, (*wechat\_id*, ))

4.6 出队（wechat\_id, timer\_list）

1）若4.7真，继续

2）TCB出队

4.7 根据wechat检测是否在队列（wechat）

1）true or false

4.8 获取endtime(wechat\_id)

4.9 根据course\_id检测是否在队列（course\_id）

if not a:

print("List is empty")

5.0 根据wechat获取教师课程号

**5. 管理员**

Import\_file + format\_check

**10. 信号量**

[**https://docs.python.org/3.1/library/threading.html**](https://docs.python.org/3.1/library/threading.html)

### 16.2.5.1. [Semaphore](https://docs.python.org/3.1/library/threading.html#threading.Semaphore) Example

Semaphores are often used to guard resources with limited capacity, for example, a database server. In any situation where the size of the resource is fixed, you should use a bounded semaphore. Before spawning any worker threads, your main thread would initialize the semaphore:

maxconnections = 5

...

pool\_sema = BoundedSemaphore(value=maxconnections)

Once spawned, worker threads call the semaphore’s acquire and release methods when they need to connect to the server:

pool\_sema.acquire()

...

en\_queuer()

...

pool\_sema.release()

## 16.2.8. Using locks, conditions, and semaphores in the [with](https://docs.python.org/3.1/reference/compound_stmts.html#with) statement

All of the objects provided by this module that have **acquire()** and **release()** methods can be used as context managers for a [**with**](https://docs.python.org/3.1/reference/compound_stmts.html#with) statement. The **acquire()** method will be called when the block is entered, and **release()** will be called when the block is exited.

Currently, [**Lock**](https://docs.python.org/3.1/library/threading.html#threading.Lock), [**RLock**](https://docs.python.org/3.1/library/threading.html#threading.RLock), [**Condition**](https://docs.python.org/3.1/library/threading.html#threading.Condition), [**Semaphore**](https://docs.python.org/3.1/library/threading.html#threading.Semaphore), and [**BoundedSemaphore**](https://docs.python.org/3.1/library/threading.html#threading.BoundedSemaphore) objects may be used as [**with**](https://docs.python.org/3.1/reference/compound_stmts.html#with) statement context managers. For example:

**import** **threading**

some\_rlock = threading.RLock()

**with** some\_rlock:

print("some\_rlock is locked while this executes")

**User Interface Layer**

**Function provided for users**

**Harvest**

1. 分层：不是我的事，我不操心

1.1下层：若有下层，我只要下层的结果，他对我是透明的

1.2上层：若有上层，我为她服务

2. Mutex

教师进入时间窗口队列的整个过程（包括检测是否可进）是互斥的