

# Map-walker 相关模块设计与实现 1

包含模块：*datab, log, router*

运行环境：*python 3.5*

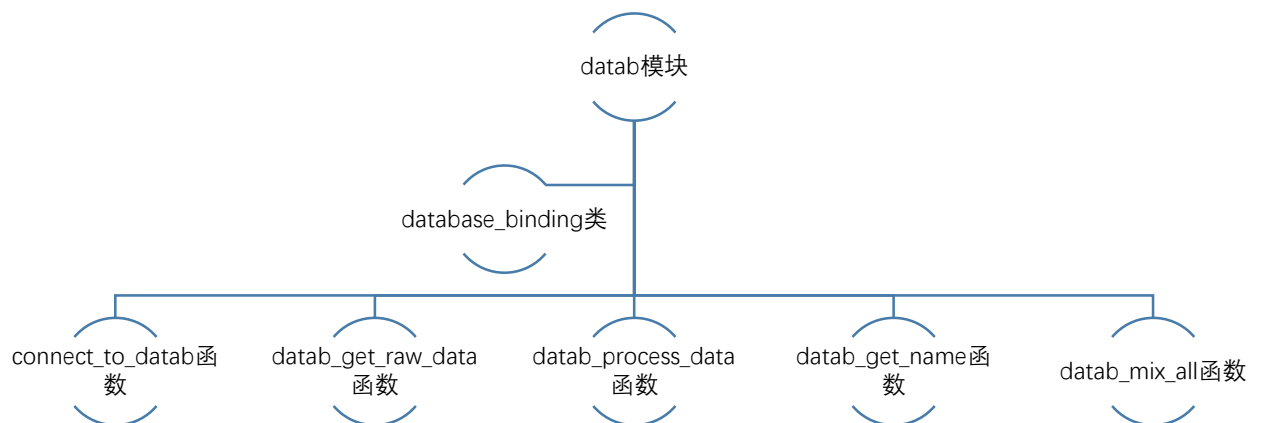
日期：*2016/4/16*

## ➤ datab 模块

### ■ 概述

该模块内通过一系列函数实现了对数据库的连接、读取以及处理等基础操作，将这些函数在一个类内进行统一调用，由 core 模块进行实例化后便可或得经过处理的数据库信息，方便后期核心算法调用。

### ■ 函数与类具体设计



整个模块内函数均如上图所示，下面我将对各个函数及类一一做具体说明。

◆ connect\_to\_datab 函数

- 传入参数：无
- 功能：连接到项目根目录下 data/data.db 数据库文件
- 返回值：指定数据库对象
- 使用样例：sql = connect\_to\_datab()

◆ datab\_get\_raw\_data 函数

- 传入参数：(数据库对象)
- 功能：从数据库中执行查询语句获得指定飞机、火车、汽车三个表的内容矩阵
- 返回值：(飞机数据矩阵, 火车数据矩阵, 汽车数据矩阵)
- 使用样例：(raw\_data\_flight, raw\_data\_train, raw\_data\_bus) = datab\_get\_raw\_data(sql)

◆ datab\_process\_data 函数

- 传入参数：(飞机、火车或汽车其中一个原生数据矩阵)
- 功能：对于 datab\_get\_raw\_data 函数得到的原生数据矩阵其中一个进行处理提取关键信息得到字典列表，内包含时间、价格两个矩阵
- 返回值：(输入矩阵处理过后的字典)
- 使用样例：data\_flight = datab\_process\_data(raw\_data\_flight)

◆ datab\_get\_name 函数

- 传入参数：(飞机、火车或汽车其中一个原生数据矩阵)
- 功能：获取所有城市 ID 编号对应城市中文名

- 返回值：(线性检索的城市中文名 list)
- 使用样例：data\_name = datab\_get\_name(raw\_data\_train)

#### ◆ datab\_mix\_all 函数

- 传入参数：(处理过的飞机字典，火车字典，汽车字典)
- 功能：将三种交通方式的数据字典进行组合得到新的字典
- 返回值：(组合有三种交通方式的大字典)
- 使用样例：

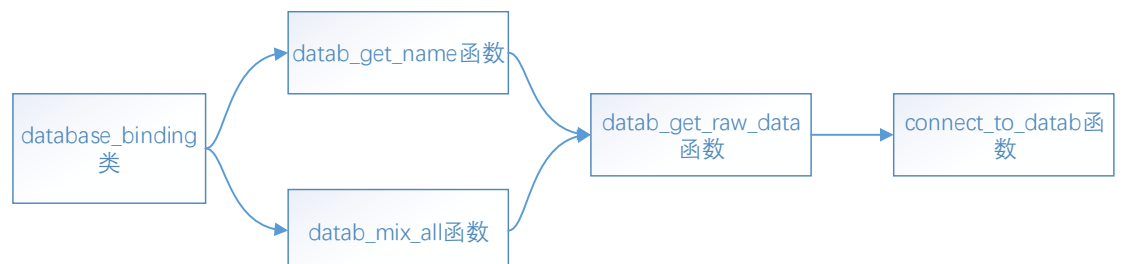
```
data_all = datab_mix_all(data_flight, data_train, data_bus)
```

#### ◆ database\_binding 类

- 传入参数：无
- 包含函数功能：\_\_init\_\_ 初始化函数用于组织及调用上述各个数据库处理函数，从而获得综合有三种交通方式的大字典和城市中文名 list 两个关键数据。
- 使用样例：database = database\_binding()

## ■ 调用说明及样例

该模块内所有函数调用均在 database\_binding 类的 \_\_init\_\_ 初始化函数中进行，交叉引用如下图所示：



## ■ 存在的问题及后期规划

目前数据库处理方面缺乏差错处理等，后期还在继续完善中。

### ➤ Log 模块

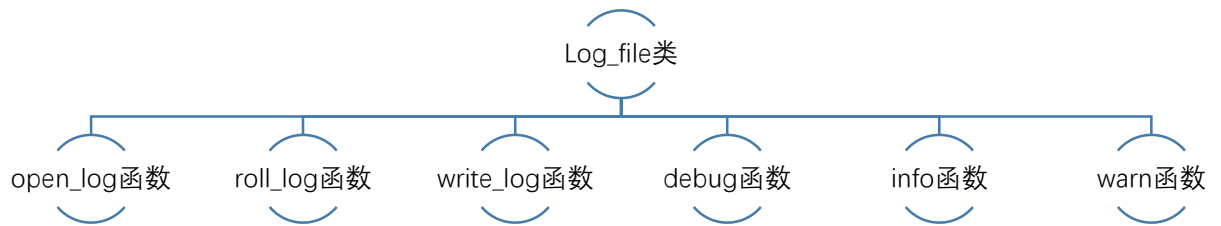
#### ■ 概述

该模块旨在实现日志文件的写入及存储管理，其中日志文件分为两种：log.txt 和 log\_test.txt，第一个为常规的日志包含数据库的读写信息、客户端请求以及程序运行中的常规事件；第二个为调试日志，其中将包含更加详细地信息，服务端运行过程中的任何一次函数调用以及数据都将被记录在案，方便出现问题后开发者及时处理。

存储管理则是该模块中一个亮点，考虑到服务端运行时可能会接受到大量地客户端请求以及长时间地运行情况，若不进行日志存储管理，单个地日志文件将变得越来越多臃肿，对于查看日志造成了十分大的障碍。

因此该模块将在初始化时对于日志文件进行检查，若文件超过一定行数就将对当前日志进行备份为.bak 文件，同时清空该日志，之后再继续进行正常写入工作。

## ■ 函数与类具体设计



### ◆ `__init__` 函数

- 传入参数：无
- 功能：实现类成员变量初始化并调用 `open_log` 函数打开日志文件

### ◆ `open_log` 函数

- 传入参数：`self`
- 功能：实现打开文件根目录下 `/data/log.txt` 文件并判断文件行数
- 返回值或类成员变量：修改类成员变量—日志文件对象
- 使用样例：`self.open_log()`

### ◆ `roll_log` 函数

- 传入参数：`self`
- 功能：将当前日志文件重命名后转移到指定目录
- 返回值：无
- 使用样例：`self.roll_log()`

### ◆ `write_log` 函数

- 传入参数：`(self, mode, fmt, *msg)`，其中 `mode` 用于选择写入模式，`fmt` 用于指定输入的格式，`msg` 可以输入一个 `list`。

- 功能：实现将指定消息按照固定格式写入日志文件中
- 返回值：无
- 使用样例：`self.write_log('DEBUG', '%d %s' , 21, 'hello' )`

◆ debug 函数

- 传入参数：`(self, fmt, *msg)`
- 功能：实现 `write_log` 模式中的 debug 模式，写入日志文件后改行会有 debug 标记
- 返回值：无
- 使用样例：`self.debug( '%d %s' , 21, 'hello' )`

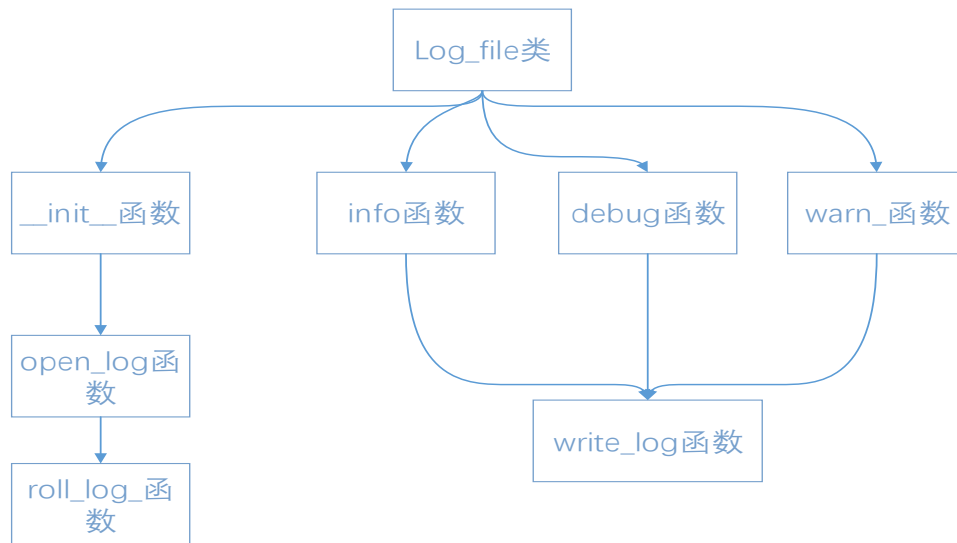
◆ info 函数

- 传入参数：`(self, fmt, *msg)`
- 功能：实现 `write_log` 模式中的 info 模式，写入日志文件后改行会有 info 标记
- 返回值：无
- 使用样例：`self.info( '%d %s' , 21, 'hello' )`

◆ warn 函数

- 传入参数：`(self, fmt, *msg)`
- 功能：实现 `write_log` 模式中的 warn 模式，写入日志文件后改行会有 warn 标记
- 返回值：无
- 使用样例：`self.warn( '%d %s' , 21, 'hello' )`

## ■ 调用说明及样例



该类中实例化一个 Log\_file 类的对象 log\_file，因此可以通过在其他模块中直接引用 log\_file 对象进行日志文件的写入。

## ■ 存在的问题及后期规划

目前程序中只加入了普通日志文件的写入，还较为基础，后期规划中将会引入 log\_test 的详细设计以及对于 log\_test 的自动分析从而使得系统可以简单判断服务器出现的问题。

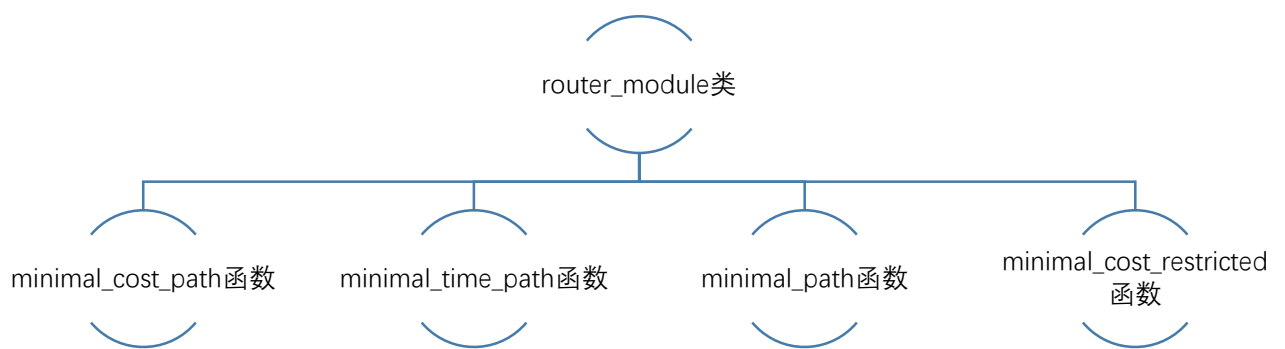
### ➤ Router 模块

#### ■ 概述

该模块设计用于对于三种“路由”策略的实现。模块内采用 OOP 的编程方法，将算法函数均封装到 router\_module 类中，从而在 core 模块中进行统一调用。

其中前两种策略 :时间最短和花费最少均采用相同的改造 Dijkstra 算法进行, 通过传入 mode 参数判断当前计算时间还是话费。该改造 Dijkstra 算法当是本模块一大特色, 改造后可以计算一个起点到若干终点的最短路径, 实现了旅行模拟系统的要求。

## ■ 函数与类具体设计



### ◆ \_\_init\_\_ 函数

- 传入参数 : (self, database\_binding) => 传入数据库管理对象
- 功能 : 实现从 database\_binding 对象中读取综合数据库字典 data\_all
- 返回值或类成员变量 : 获取 data\_all

### ◆ minimal\_path 函数

- 传入参数 : (self, datab\_link, source, destination, mode)
- 功能 : 通过 datab\_link 综合数据库字典利用改造 dijkstra 算法计算从 source 城市到 destination 目的地列表所有城市的最短时间路径或者最少花费路径
- 返回值 : (path, dis) => 其中 path 列表代表路径以及交通方式, dis



值代表最短时间或最少花费

- 使用样例：`(path_cost, min_cost) = minimal_path(data_all, 0, [9, 2, 1, 6], 'price')`

#### ◆ `minimal_cost_path` 函数

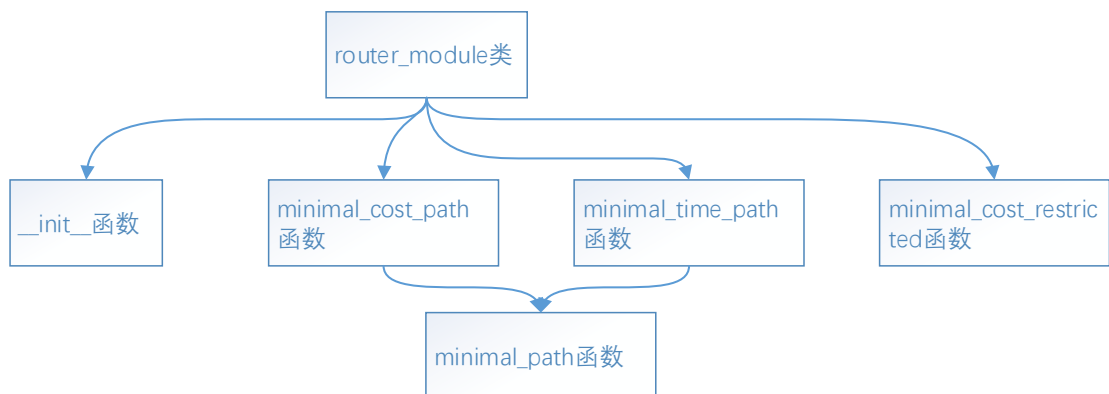
- 传入参数：`(self, id_src, id_dest)`
- 功能：调用 `minimal_path` 函数直接计算最少花费
- 返回值：`self.minimal_path(self.data_all, id_src, id_dest, 'price')`
- 使用样例：`(path_cost, min_cost) = minimal_cost_path(1, [2, 0, 9])`

#### ◆ `minimal_time_path` 函数

- 传入参数：`(self, id_src, id_dest)`
- 功能：调用 `minimal_path` 函数直接计算最少时间
- 返回值：`self.minimal_path(self.data_all, id_src, id_dest, 'time')`
- 使用样例：`(path_time, min_time) = minimal_cost_path(1, [2, 0, 9])`

#### ◆ `minimal_cost_restricted` 函数

### ■ 调用说明及样例



通过实例化 `router_module` 类后便可利用该对象调用 `minimal_cost_path` 和

`minimal_time_path` 函数直接计算最少花费和最少时间路径。

## ■ 存在的问题及后期规划

目前模块内计算最短路径还没有加上保存城市之间航班号、列车号或汽车号的数组，后期规划还需要加上不同交通方式的发车时间匹配，以更加逼真地模拟真实旅游系统。