

A Real-Time Data Analyzing System

Supervisor: Prof. David ROSSITER

JIAN Xun: chineshboy@hotmail.com

Contents

1	Introduction	2
1.1	<i>Background</i>	2
1.2	<i>Overview</i>	2
2	Design	3
2.1	<i>Framework</i>	3
2.2	<i>Basic Elements</i>	4
2.2.1	<i>Data Type</i>	4
2.2.2	<i>Data</i>	4
2.2.3	<i>Function</i>	5
2.2.4	<i>Task</i>	9
2.3	<i>Database and DAO</i>	10
2.4	<i>Data Center</i>	12
2.5	<i>Task Center</i>	14
2.6	<i>API</i>	15
3	Implementation	16
3.1	<i>Environment</i>	16
3.2	<i>Third Party Libraries and Frameworks</i>	16
4	Demonstration	17
5	Conclusion	20
6	Appendix.....	21
6.1	<i>References</i>	21
6.2	<i>Minutes of Meetings</i>	21
6.2.1	<i>Minutes of the 1st meeting</i>	21
6.2.2	<i>Minutes of the 2nd meeting</i>	21
6.2.3	<i>Minutes of the 3rd meeting</i>	22
6.2.4	<i>Minutes of the 4th meeting</i>	22
6.2.5	<i>Minutes of the 5th meeting</i>	23

1 Introduction

1.1 Background

A software system can generate a large number of log data every day. These data need to be analyzed in time to reveal essential knowledge and even problems inside the system. In my working experiences, there were sometime that I needed to do some analyze on some daily-generated log data. I tried many ways to do it, but the costs were very high, especially when I need to change the way of analyzing.

To do it with a relatively low cost, we need a special designed system. Many companies use Apache Storm to do this job, but it needs some programming, which is not very easy to use. Then I came about this idea, to make a common system which is easy to use and powerful for normal analyzing.

1.2 Overview

This project designed and implemented a system which provides varies options to analyze input data streams. A web page is provided to do configuration and to view the data. To make the system flexible and easy to use as well, the input data structure is customized so that this system can handle different kinds of data and users of this system only need to know some basic knowledge of how to apply a function and build a formula. All the objects used in the system are described with JSON, a widely used description language. With those features, I believe our system can can handle normal analyzing jobs in an efficient way.

2 Design

2.1 Framework

The system has three main parts: Controller, Data Center and Task Center. The Controller handles all the requests, including querying a web page, configuring the system, and sending the data. The Data Center is responsible for data storage, data registration, data verification, data querying, and data transmission. The Task Center is responsible for task storage, task registration, task querying, and task execution.

The MySQL is at the bottom of this system. Only Data Center and Task Center can directly read and write the data, which means the Controller must get data by querying from Data Center and Task Center.

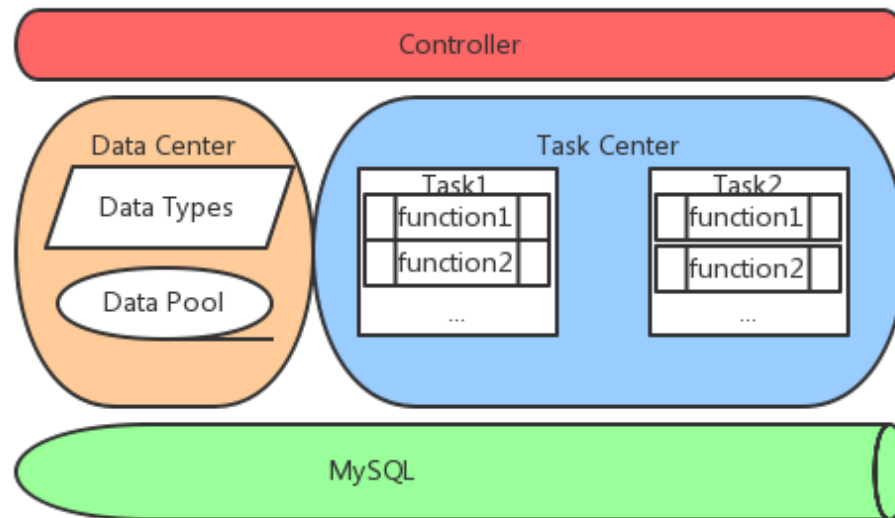


Figure 2.1.1 Framework

2.2 Basic Elements

2.2.1 Data Type

A data type defines the structure of a kind of data. I use JSON to describe a data type like the following:

```
{
  "title": "the title of the data type",
  "attrs": {
    "name": "string",
    "value1": "int",
    "value2": "string"
  }
}
```

The “title” is a label given by the user, which is a string contains at most 100 characters.

The “attrs” is a map contains all the attributes in this data type. Each attribute is described as a key-value pair, in which the key is a string indicates the name of the attribute, and the value is also a string indicates the basic data type of the attribute. So far I support 8 basic data types, which are:

int, long, float, string, bool, list, dict, set

Those basic data types have the same meanings as they have in Python.

2.2.2 Data

The data is the main object to be calculated and stored. It is also described by JSON:

```
{
  "data_type_id": 5243212,
  "attrs": {
    "name": "Bob",
    "value": 12.59
  },
  "time": "2015-09-01 10:50:39,423"
}
```

The “data_type_id” is an integer, indicates the id of a data type. The system will use the description in the specified data type to read and check the attributes in the data.

The “attrs” is similar to the “attrs” in the data type, while the value here is the real value of the attribute other than its type.

The “time” is a string indicates the sending time of the data, which should match the time format “YYYY-mm-dd HH:MM:SS”.

2.2.3 Function

A function is a simple operation on the data. This system has 9 customized functions. In function “self_calc”, user can use all the built-in functions in “math” module in Python, so it can support more than 20 functions now.

Different functions have different attributes, but they all need attribute “name” to specify which function it is. For some functions that has “time_range” attribute, this attribute is always inherited from the description of the task (we will see it in the Task part), so there is no need to put it in function description in practice. The value of the “tag” attribute is always the name of the result to put in the output data.

The details of each customized function are as follows.

- **count**

```
{
  "name": "count",
  "distinct": [
    "function",
    "cost"
  ],
  "tag": "count_function_cost",
  "time_range": "1m"
}
```

This function can count the number of data comes in each time period.

The “distinct” is a list of attribute names. All the attributes here act like the primary key in a relational database. Data shares the same attributes in the list would not be counted twice.

The “time_range” is a string represents the time period. it is a positive integer followed by a time unit. it supports 4 different time units, which are “d” (day), “h” (hour), “m” (minute) and “s” (second). In the sample JSON, the function would count the data in every minute.

The result of this function is an integer—the number of (distinct) data comes in each time period.

- **average**

```
{
  "name": "average",
  "target": "cost",
  "tag": "avg_function_cost",
  "time_range": "1m"
}
```

This function can calculate the average value of the specified attribute in the data in each time period.

The “target” is the name of the attribute to be calculated. The attribute must have type int or long or float.

The result of this function is a float—the average of the attribute in each time period.

- **percentile**

```
{
  "name": "percentage",
  "target": "cost",
  "percentage": 95,
  "time_range": "1m"
}
```

This function can calculate the specified percentile of an attribute (in increasing order) in each time period, and return the whole corresponding data.

The “target” is the name of the attribute to be calculated. The attribute must have type int or long or float.

The “percentage” is a positive integer. In the sample JSON, the function would pick the value in the 95th percentile. If no data is exactly in this percentile, the nearest smaller one would be picked.

The result of this function has the same data type as the input data’s.

- **top**

```
{
  "name": "top",
  "tag": "top_cost",
  "target": "cost",
  "num": 3,
  "time_range": "1m"
}
```

This function gives the top values of an attribute in each time period.

The “target” is the name of the attribute to be calculated. The attribute must have type int or long or float.

The “num” specifies the number of top values to be kept.

The result of this function is a list contains the top values, if no enough data comes, the last one would be repeatedly put in the result.

- **sum**

```
{
  "name": "average",
  "tag": "avg_cost",
  "target": "cost",
  "time_range": "1m"
}
```

This function is similar to the average function. the only difference is that this one gives the sum of the attribute instead of average.

- **standard_deviation**

```
{
  "name": "standard_deviation",
  "tag": "sd_cost",
  "target": "cost",
  "time_range": "1m"
}
```

This function is similar to the average function, but this one gives the standard deviation of the target attribute.

- **filter**

```
{
  "name": "filter",
  "condition": "{function}==\"add\" and {cost}/10>30",
  "attr_remain": [
    "cost",
    "function"
  ]
}
```

This function can filter out the data that doesn't meet the condition.

The "condition" is a string that represents the filter condition. It must be a expression that has either "True" or "False" as its value. To use the attributes in the expression, just enclose the name with "{" and "}". In the sample JSON, the attribute "function" and "cost" would be used in the expression. Only the data which makes the expression have value "True" can pass the filter.

The "attr_remain" is a list of names of attributes. Attributes in this list would be put in the result data, and other attributes would be dropped.

The output data of this function contains only the attributes in "attr_remain" from the input data.

- **collect**

```
{
  "name": "collect",
  "target": [
    "function",
    "cost"
  ],
  "time_range": "1m"
}
```

This function can collect attributes in separate lists in each time period.

The "target" is a list, contains the names of the attributes to be collected. In the sample JSON, the attributes "function" and "cost" would be collected in two lists with the same names, and other attributes would be dropped.

•self_calc

```
{
  "name": "self_calc",
  "attr_remain": [
    "cost",
    "function"
  ],
  "expression": "log({cost}, 2)*10",
  "tag": "log_cost"
}
```

This function allows user to use the built-in functions and operators in “math” module in Python.

The “expression” is a string represents an expression in Python. Similar to the “condition” in filter function, user can use operators and functions in this expression, and enclose the attributes in brackets. The type of the result must be one of the 8 basic types.

The “attr_remain” is the same as it is in the filter function.

2.2.4 Task

A task in this system is a combination of several functions. Since a function is very simple, a task is used to perform a complex calculation on a kind of data. Unlike functions, the result data of a task would be stored and can be viewed by user and be used by other tasks.

Generally, a task would be defined as follows:

```
{
  "input_data_type_id": 12,
  "time-range": "1m",
  "functions": {
    "1": {... },
    "2": {... },
    ...
  }
}
```

The “input_data_type_id” is an integer, indicates the id of the data type of the input data.

The “time_range” is talked in the Function part. For some tasks, this attribute is not necessary, but should be always filled with value.

The “functions” is a key-value map, in which the key is the order of the function, starting from 1, represented in string, and the value is an object describe the function, as in the Function part.

Every task in the system has another attribute “result_data_type_id”, which indicates the data type of the result of the task. This attribute is automatically generated by the system, and return to the user after registration.

There are no limits of the number of functions in a single task, but considering the storage in database, the total length of the JSON string of “functions” cannot be larger than 20000.

A task would run as a thread in the Task Center. When a data comes, it will firstly come into the function with order “1”, and the result will com into the function with order “2”, and so on. The result of the last function would be the result of the task, and be put into the Data Center.

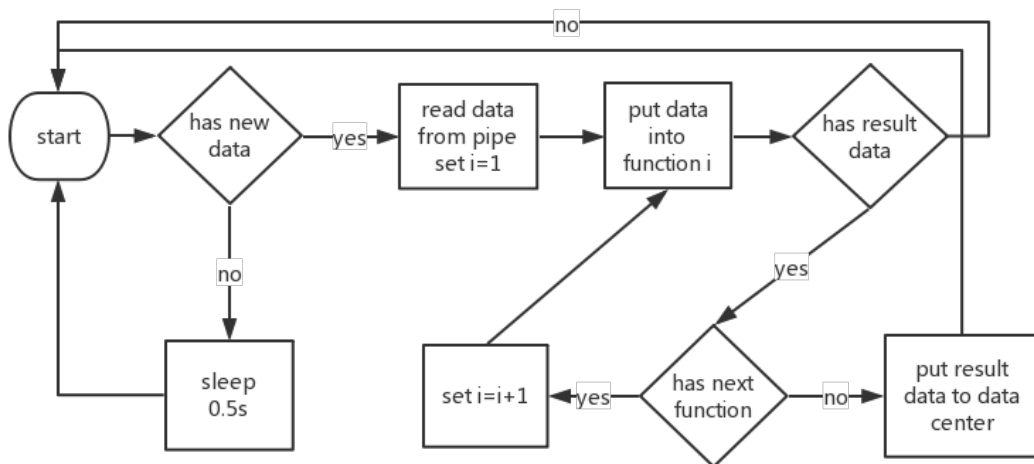


Figure 2.2.4.1 Task work flow

2.3 Database and DAO

There are 3 tables in the database in total. Each table has its corresponding object, which is built using SQLAlchemy, to abstract the operations on database.

Table “datatypeinfo” stores the information of data types, including the description and id.

Table 2.3.1 datatypeinfo

Column	Type
id	int(32)
title	varchar(100)
description	varchar(2000)

The “id” is generated automatically by the database, and the “description” is a JSON string describing the attributes of the data type.

Table “taskinfo” stores the information of tasks.

Table 2.3.2 taskinfo

Column	Type
id	int(32)
title	varchar(100)
input_data_type_id	int(32)
result_data_type_id	int(32)
functions	varchar(20000)
time_range	varchar(10)

The “id” is generated automatically by the database, and the “functions” is a JSON string describing the functions and orders of the task.

Table “datainfo” stores the data sent by the user or generated by a task.

Table 2.3.3 datainfo

Column	Type
id	int(32)
data_type_id	int(32)
attrs	varchar(2000)
time	datetime

The “id” is also generated by the database, the “attrs” is a JSON string describing the values of attributes of the data.

2.4 Data Center

The Data Center handles the data management and data type management. It consists of a map stores all data types, a queue stores new data, and a worker thread to handle the new data. The reason to use the worker thread is to handle the data asynchronously, so that the Controller thread wouldn't be blocked.

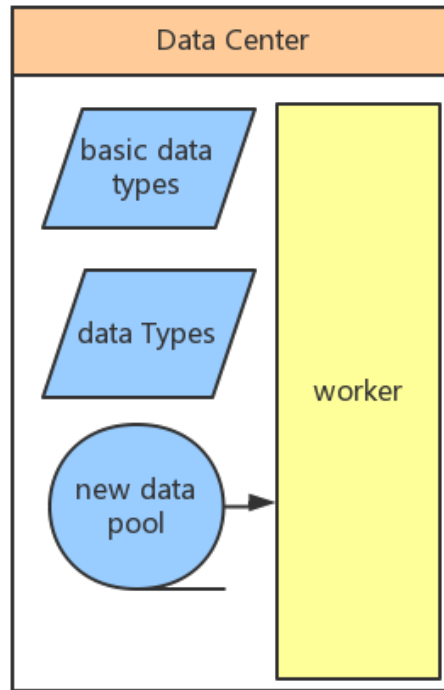


Figure 2.4.1 Data Center structure

When new data type comes (data type registration), the Data Center firstly check the validity of the data type description. If it is valid, the Data Center would store the information in the database, and return an id of this data type.



Figure 2.4.2 Data type registration work flow

When new data comes, the Data Center firstly check the validation of the data, using the data type description. Invalid data would be dropped, and valid data would be put in a queue. Another thread would read and handle those data.

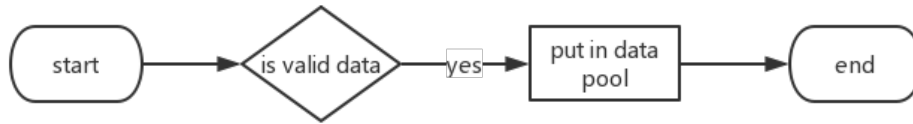


Figure 2.4.3 New data work flow

There is a worker thread keeping look at the data pool. As soon as there is new data in the data pool, the worker would store the data into database, and deliver the data to task center for calculation.

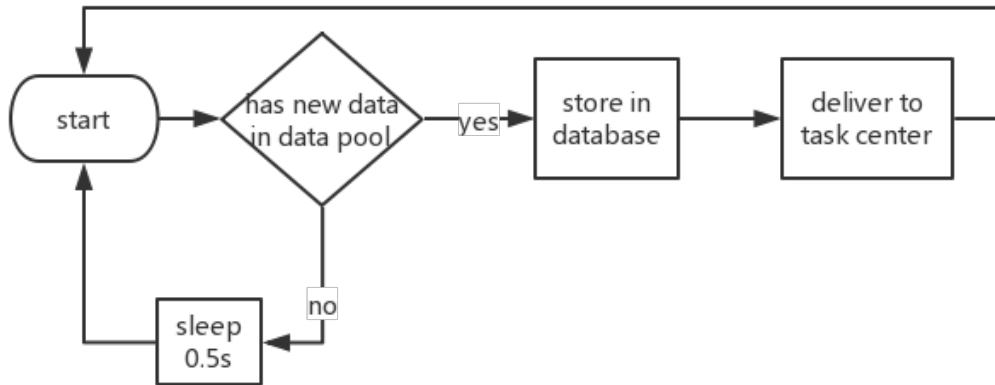


Figure 2.4.4 Data delivery work flow

2.5 Task Center

The Task Center is responsible for the task management. It consists of a task list, a data distribution map. Similar to the worker thread in the Data Center, each task is running as an independent thread. The data distribution map maintains the relationship of data and tasks.

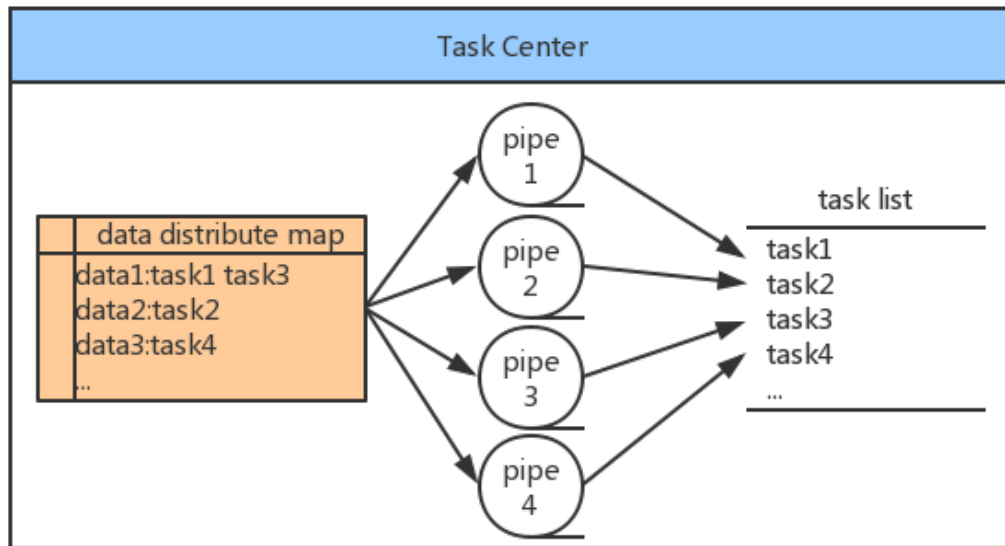


Figure 2.5.1 Task Center structure

When a new task comes, the Task Center firstly checks if the description of the task is valid, then register the result data type, stores it in the database, updates the data distribute map, creates a thread in task list running this task.

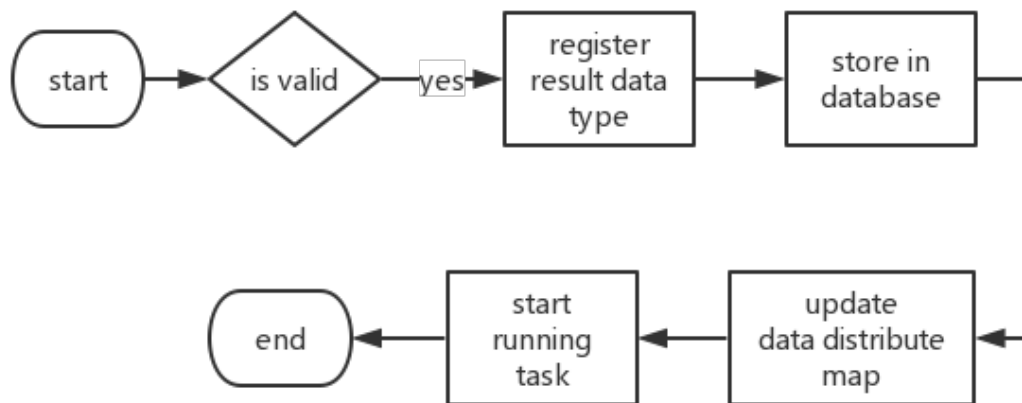


Figure 2.5.2 Task registration work flow

When new data comes, it will be put into some pipes according to the data distribute map, then the task threads will read from their own pipe and do the calculation.



Figure 2.5.3 New data work flow

2.6 API

The system provides RESTful API for users to use.

- GET /home
Returns the home page of this system. User can register data types and tasks, or view data from this web page.
- POST /datatype
Register a new data type. User should put the description of the data type in a JSON string in parameter "desc".
- GET /datatypes
Returns all the data types registered in the system in a JSON string.
- GET /basictypes
Returns all supported basic data types in a JSON string.
- POST /task
Register a new task. User should put the description of the task in a JSON string in parameter "desc".
- DELETE/ task
Delete an existing task. User should put the id of the task in parameter "id".
- POST /data
Send a new data to the system. User should put the description of the data in a JSON string in parameter "data".

3 Implementation

3.1 Environment

- Language: Python 2.7
- IDE: PyCharm CE 5 [1]
- Database: MySQL 5.6.27 [2]

3.2 Third Party Libraries and Frameworks

- Flask 0.10.1 [3]

Flask is a micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed.

In this system, Flask is used to dispatch requests, and rendering pages.

- SQLAlchemy 1.0.8 [4]

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

- jQuery [5]

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is free, open-source software licensed under the MIT License.

The web page of this system uses jQuery to send request, handle response, and render the page.

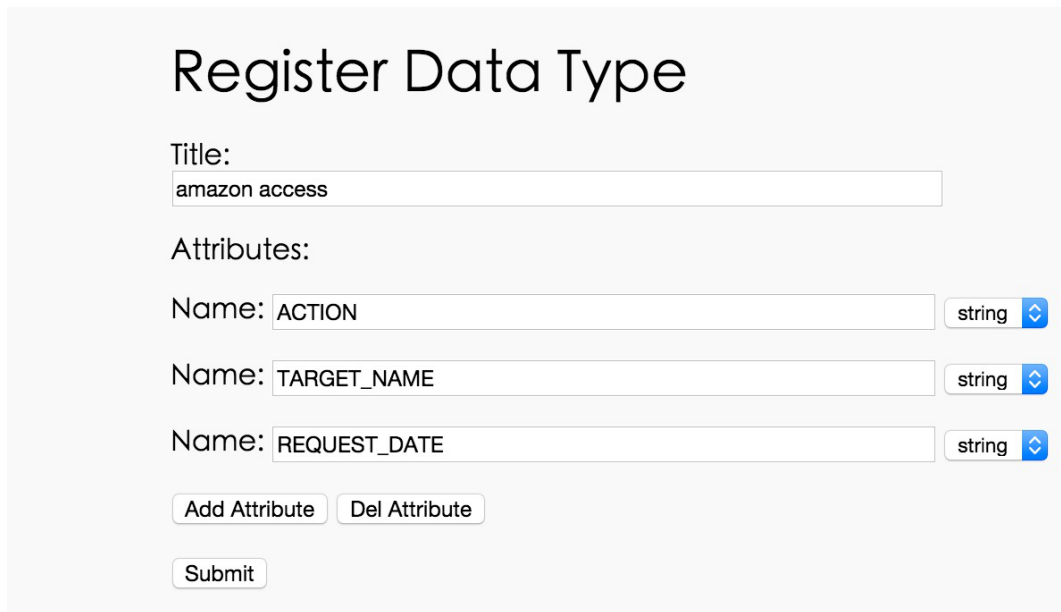
- Highcharts 4.1.9 [6]

Highcharts is a charting library written in JavaScript. Highcharts is free for non-commercial and personal use.

The web page of this system uses Highcharts to show the data in a line chart.

4 Demonstration

Here is a simple demonstration of using this system to analyze the access data of amazon. The screenshots below show how to get the top 3 access frequencies of single items.




The screenshot shows a web form titled "Register Data Type". It contains the following elements:

- Title:** A text input field containing "amazon access".
- Attributes:** A section containing three attribute entries, each with a "Name:" label, a text input field, and a dropdown menu.
 - Attribute 1: Name: ACTION, dropdown: string
 - Attribute 2: Name: TARGET_NAME, dropdown: string
 - Attribute 3: Name: REQUEST_DATE, dropdown: string
- Buttons:** Two buttons labeled "Add Attribute" and "Del Attribute" are positioned below the attribute list. A "Submit" button is located at the bottom of the form.


Figure 3.2.1 Register data type

Figure 3.2.1 shows how to register a data type on the page. Go to the home page of the system, choose the "RegisterDataType" tab, enter some information, and then click "submit" to register the data type. You can click "Add Attribute" to add more attributes.

Task Title:

Input Data Type ID: 

Desc:

Time Range: 

Function 1:

Function 2:

Figure 3.2.2 Register task

Figure 3.22 shows how to register a task. Go to the “RegisterTask” tab, enter a title, choose the input data type (you will see the data type description below), enter an appropriate time range and the description of the functions. In this task, I use “filter”, “collect”, “self_calc” functions. Finally click “submit” to register the task.

After registering the task, you can send the data through the system APIs.

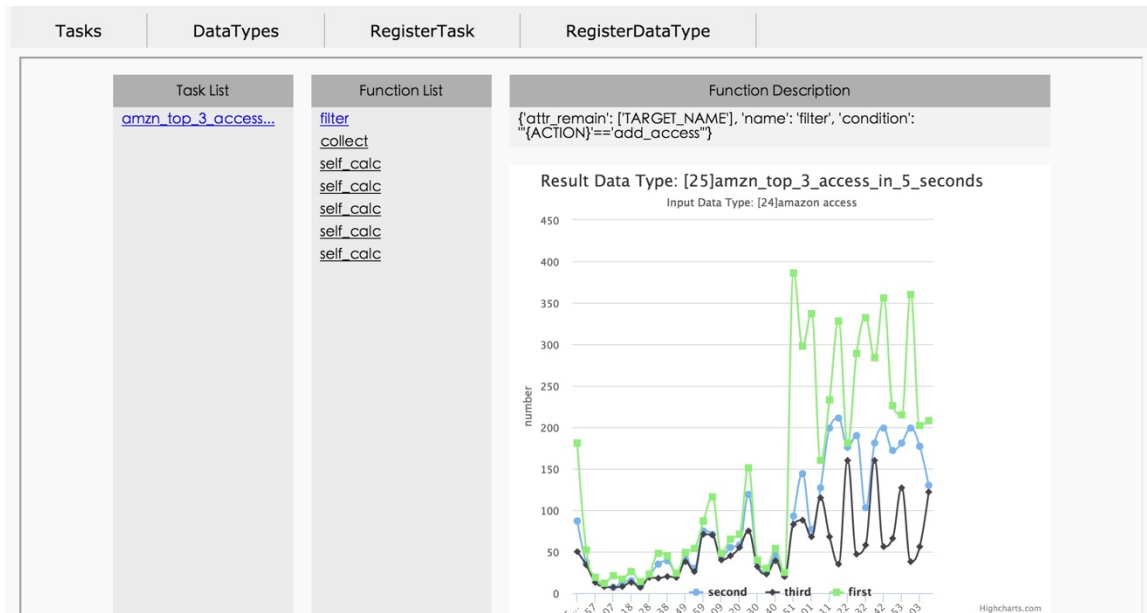


Figure 3.2.3 View data

The result of the task can be seen on the home page immediately.

In this case, we can see the top 3 access frequencies of single items every 5 seconds.

5 Conclusion

This system is implemented by myself starting from zero. Doing this project gives me a deeper understanding of system robustness and maintainability. I also learned much knowledge of simple web page development.

As we can see in the demonstration, this system is flexible to use and has the ability to deal with different types of data. Actually what I'm trying to do is to make the system easy to use and upgrade as well. When implementing the system, I firstly complete a prototype, and then improve each parts to make it stronger. During this process, some bad design was fixed, and some extra works were made to reduce the cost of upgrading.

Three months' time is not enough for me to make this project perfect. I can still see some parts that can become better. However, the main functions of this system are all good and can really show the advantages. That's because Prof. David Rossiter gave me lots of advices of how to focus on the core functions of the system and how to manage time and plan. Without his supervision, this project cannot be as good as it is now.

6 Appendix

6.1 References

- [1] PyCharm: <https://www.jetbrains.com/pycharm/>
- [2] MySQL: <http://dev.mysql.com/downloads/mysql/>
- [3] Flask: <http://flask.pocoo.org/>
- [4] SQLAlchemy: <http://www.sqlalchemy.org/>
- [5] jQuery: <https://jquery.com/>
- [6] Highcharts: <http://www.highcharts.com/>

6.2 Minutes of Meetings

6.2.1 Minutes of the 1st meeting

- Date: Wednesday, 16 September 2015
- Time: 1:30 pm
- Place: Room 3512
- Attending: Prof. Rossiter, JIAN Xun
- Absent: None
- Recorder: JIAN Xun
- Approval of minutes: N/A
- Discussion items and things to do:
 - ◆ The initial project idea
 - ◆ Simplify the project idea so that the workloads are reasonable
- Meeting adjournment and next meeting:
 - ◆ No adjournment
 - ◆ Next meeting on 22 September

6.2.2 Minutes of the 2nd meeting

- Date: Tuesday, 22 September 2015
- Time: 2:00 pm
- Place: Room 3512
- Attending: Prof. Rossiter, JIAN Xun
- Absent: None

- Recorder: JIAN Xun
- Approval of minutes: approved
- Discussion items and things to do:
 - ◆ The simplified project idea
 - ◆ Get to know some real-world data
 - ◆ Starting doing project
- Meeting adjournment and next meeting:
 - ◆ No adjournment
 - ◆ Next meeting on 13 October

6.2.3 Minutes of the 3rd meeting

- Date: Tuesday, 13 October 2015
- Time: 10:50 am
- Place: Room 3512
- Attending: Prof. Rossiter, JIAN Xun
- Absent: None
- Recorder: JIAN Xun
- Approval of minutes: approved
- Discussion items and things to do:
 - ◆ System framework
 - ◆ Design details of Data Center, Task Center, and functions
 - ◆ To get the system running
 - ◆ To create a web page
- Meeting adjournment and next meeting:
 - ◆ No adjournment
 - ◆ Next meeting on 2 November

6.2.4 Minutes of the 4th meeting

- Date: Monday, 2 November 2015
- Time: 2:00 pm
- Place: Room 3512
- Attending: Prof. Rossiter, JIAN Xun
- Absent: None
- Recorder: JIAN Xun
- Approval of minutes: approved
- Discussion items and things to do:
 - ◆ A working prototype with one function
 - ◆ Enrich the kinds of functions
- Meeting adjournment and next meeting:
 - ◆ No adjournment

- ◆ Next meeting on 17 November

6.2.5 Minutes of the 5th meeting

- Date: Tuesday, 17 November 2015
- Time: 2:25 pm
- Place: Room 3512
- Attending: Prof. Rossiter, JIAN Xun
- Absent: None
- Recorder: JIAN Xun
- Approval of minutes: approved
- Discussion items and things to do:
 - ◆ An improved powerful system
 - ◆ Starting doing report
 - ◆ Prepare a good demo
- Meeting adjournment and next meeting:
 - ◆ No adjournment
 - ◆ No next meeting