Post-print: Web.py-Based Interface Implementation for the MUSER Software System

Authors: Cai Nengjian, Liu Donghao, Deng Hui, Wei Shoulin, Wang Feng, Mei Ying, Dai Wei, Liu Yingbo, Wu Jingping

Date: 2017-09-26T00:00:00+00:00

Abstract

The Mingantu Ultrawide SpEctral Radioheliograph (MUSER) has completed its hardware construction, and its accompanying data processing system has also been preliminarily developed. To enhance usability and simplify operational complexity, the data processing system provides both traditional command-line user interfaces and web-based interaction methods. This work briefly introduces the fundamental concepts of Web.py and the interaction workflow of its internal components, discusses the methods and procedures for web development based on Web.py, and implements the web interface interaction for the data processing system utilizing the Web.py framework. The achieved results have been applied in practice, improving the usability of the radioheliograph data processing system and providing valuable reference for the development of other astronomical software systems.

Full Text

Abstract

The Mingantu Ultrawide Spectral Radioheliograph (MUSER) has completed its hardware construction, and its accompanying data processing system has also been initially developed. To improve user operation usability and simplify operational difficulty, the data processing system provides both traditional command-line user interfaces and web-based interaction methods. This paper briefly introduces the methods and processes of web development and utilizes the web.py framework to implement the web interface interaction for the data processing system. The results have been applied, improving the usability of the radioheliograph data processing system and providing valuable reference for the development of other astronomical software systems.

Keywords: Astronomical software systems; web.py; User interface

1. Introduction

To promote the development of domestic solar radio astronomy observation technology, Chinese astronomers proposed the construction of the Mingantu Ultrawide Spectral Radioheliograph (MUSER), which has been fully completed [1]. This instrument can simultaneously observe the Sun with high spatial, temporal, and frequency resolution in the centimeter and decimeter wavebands. The project was constructed in two phases: MUSER-I (0.4–2 GHz) and MUSER-II (2–15 GHz). The data from this facility can better facilitate research on solar activities.

The data processing system for MUSER is a key component of the entire project. Convenient data processing software has become an important support for realizing the telescope' s capabilities. Currently, browser-based applications have become mainstream in software development. Such software features high portability, which is very beneficial for long-term maintenance. From a long-term perspective, implementing a user-friendly web interface that does not require software deployment and can meet the requirements of both on-site observation and remote data processing is essential.

Many technologies are available for developing web applications, with the most common being based on [other frameworks]. However, such software suffers from complex deployment environments and long development cycles. Astronomical software development mostly adopts Python, which can integrate and interact well with data processing systems. The MUSER data processing system is also based on Python, and thus the web user interface for MUSER similarly adopts the web.py framework. This paper first introduces the architecture of the data processing system and the functions of its operation interface. It then discusses the lightweight web framework web.py and its internal component interaction flow, focusing on methods for rapid development using the web.py framework to implement web interface and function encapsulation for MUSER.

2. MUSER Data Processing System Architecture

With the significant improvement of telescope performance, the volume of generated data has multiplied and the processing procedures have become increasingly complex. Computer software systems have become indispensable tools in astronomical data processing. Astronomical software systems can be divided into two categories: data processing and data release. Data processing software is generally deployed locally, using standalone or computer cluster methods to process observation data, including the most common IDL, CASA, etc. Data release software is generally based on web applications and Representational State Transfer (REST), a standard network service release technology such as the cone search in the Virtual Observatory.

In daily MUSER data processing, there are two requirements: one is the processing of historical data, and the other is real-time processing of observation data sampled every second. To meet the needs of high-speed data processing, a

high-performance computing system deployed at the Mingantu Observatory has been developed and implemented [2]. At the same time, to meet the needs of astronomers for offline processing, a client software that can run on a standalone machine has been developed with similar functionality. The overall functional structure of the MUSER software system is shown in Figure 1 [Figure 1: see original paper].

The system is divided into three layers: core data processing, user interface for interaction, and database. The core data processing module has been developed based on OpenCluster [4-5], supporting distributed task scheduling and other functions. It uses CLEAN, FFT, IFFT, etc., to achieve allocation and scheduling of computing resources using Python. The database stores several key parameters in data processing, such as optical fiber transmission delays and meteorological data [6]. The software system uses MySQL as the primary database and also supports exporting data to the Redis in-memory database to improve query speed. Functions such as antenna status maintenance, automatic weather station integration, and satellite data correction configuration have been developed. MUSER has implemented a command-line interface similar to CASA [7], but the command-line approach has shortcomings such as numerous commands and non-intuitive presentation. To further improve the efficiency of the data processing system, a more user-friendly interactive interface is needed.

3. Web.py Framework

Web.py [8] is a pure Python-based lightweight web application development framework. The current version is 0.38. Web.py can run as a standalone daemon process or with the help of the Web Server Gateway Interface (WSGI) on any compatible server. It follows the Model-View-Controller (MVC) architectural style for web services, mapping URL classes to request handling objects. It natively supports compilation and provides a minimal web server. It can be installed by downloading the official binary package (web.py-0.38.tar.gz), decompressing it, and executing the command `python setup.py install`, or through easy_{install}. Web.py does not depend on other third-party libraries.

The web.py framework execution process is mainly defined in several files: application.py, wsgi.py, httpserver.py, and wsgiserver.py. Application.py is the entry point for a web application; wsgi.py runs the web application on a built-in simple web server or a standard server (FastCGI); httpserver.py is responsible for creating the server and receiving client requests; wsgiserver.py is the built-in web server. The internal execution flow from browser access is shown in Figure 2 [Figure 2: see original paper].

3.1 URL Design and Processing

For any web application, URL organization is the most important part. A well-designed URL can clearly express the control logic of the entire web appli-

cation software. Web.py supports RESTful design, which has advantages such as clear structure and easy extension. Therefore, the web interaction interface adopts RESTful design. RESTful contains two important concepts: first, resources, which represent the objects of operation and are usually nouns, with naming conventions generally corresponding to database table names; second, state transfer, which represents operations on resources that cause state changes, corresponding to four data operations: Create, Read, Update, Delete (CURD). In HTTP protocols, these are the four verbs: GET, POST, PUT, DELETE. GET requests are used to retrieve data, POST for submitting new forms when inserting or modifying data. Web.py only requires defining functions with the same names in the corresponding processing classes.

Table 1 lists the mappings between some classes and resources in the web interaction interface. Mapping of classes to resources in MUSER web design

URL Path	Python Class
/antenna	Antenna
/antenna/position	AntennaPosition
/antenna/flag	AntennaFlag
/antenna/delay	AntennaDelay
/parameter	Parameter
/job	Job

The control mode is simple. When the web application starts, all URLs used by the system must be defined in a tuple list in web.py. The tuple format is: (“URL path” , “Python class”). For example, if the path used is “/antenna” and the processing class is “Antenna” , then the tuple is defined as (“/antenna” , “Antenna”). For instance, to query the status of a computing task, assuming a task ID needs to be passed, the path can be defined as “/job/(.+)” . The content after /job/ will be captured and used as a parameter in the Job processing class. When querying “/job/10” , the specific processing class can obtain this task ID, execute the specific query, and return the result.

3.2 Database Operations

Web.py supports multiple database operations and encapsulates various database operations, providing a simple and easy-to-use API. Specific low-level database operations require corresponding database operation modules such as Python’ s MySQLdb or pymssql for SQL Server. The database-related code in web.py is in db.py. The entry function is database(), which looks up the implementation class for the corresponding database type in MySQL, initializes this database implementation class with remaining parameters, and returns a specific database operation object. This object is used to perform operations on database tables. Table 2 shows the database operation methods in web.py.

Method	Description
db.insert(self, ···)	Insert operation
db.query(self, ···)	Query operation
db.update(self, ···)	Update operation
db.delete(self, ···)	Delete operation

The data processing system uses MySQL as the primary database, supporting read and write operations. Redis is mainly used in distributed environments, and the system regularly exports data from MySQL to Redis. Redis provides read-only operations in the data processing system to ensure data consistency. Redis is an in-memory key-value storage system that supports high-speed, high-concurrency read and write operations, providing guarantee for high-speed access to meteorological and antenna-related parameters in MUSER' s distributed processing environment. Redis is not a relational database, and its access and operations require Python' s redis module. SQLite is used in the standalone version of the processing system.

3.3 Template Rendering

Web frameworks provide template technology to implement complex web pages. Web.py' s template functionality is implemented in template.py. The template language allows embedding Python code in template files. Template files must begin with `$def with()` to define all variables used in the file. These variable names must be consistent with those returned in the Python code, and then can be directly used in the template files. The power of web.py templates lies not only in variable reference but also in supporting Python process control statements such as `for`, `while`, `if`, `elif`, which need to be prefixed with `$`. For example, part of the task processing list template in MUSER is as follows:

```
$for job in jobs:
    // jobs is the task list variable
    // Outputting the status of individual tasks
    $job.status
```

Web.py supports rendering template files in Python code using `render.weather.html`, or direct rendering. Figure 3 [Figure 3: see original paper] shows the template rendering specification in web.py.

MUSER uses a dedicated template folder 'templates' to store all template files. Web.py searches for template files from this directory, matches the corresponding template file, renders it, and returns it to the browser. Web.py supports template nesting and specifying master templates.

4. System Deployment and Results

The developed data processing system user interface has been basically completed and applied to actual data processing. The system interface is shown in Figure 4 [Figure 4: see original paper]. The system implements configuration of system parameters, providing convenience for querying and usage. The web interface provides configuration of data processing parameters, facilitating the submission of complex tasks. Compared with the command-line interface, it has better interactivity and usability. Built with web.py and Bootstrap, it constructs responsive pages. This paper introduces the process of developing the MUSER data processing system user interface based on the web.py framework. The lightweight framework provides core modules that can quickly develop a stable and effective web application according to requirements. Web.py has the characteristics of being lightweight, simple, and having clean code design style, meeting the needs of user interface development in the radioheliograph data processing system. The developed program is stable and reliable, and the system has been initially built. However, there are still shortcomings, such as more reliable data error verification and providing more real-time interactive functions, which will continue to be studied and explored in future work.

References

[1] Yan Yihua, Zhang Jian, Chen Zhijun, et al. Progress on Chinese radioheliograph in cm-dm wavebands. Astronomical Research & Technology Publications of National Astronomical Observatories of China, 91-98.

[2] Ma Rongting, Liu Fei, Deng Hui, et al. The design and implementation of a data-receiving subsystem of the CSRH-I. Astronomical Research & Technology, 63-69.

[3] Wei Shoulin, Wang Feng, Deng Hui, et al. OpenCluster: a flexible distributed computing framework for astronomical data processing. Publications of the Astronomical Society of the Pacific, 383-396.

[4] Mei Ying, Liu Donghao, Wang Feng, et al. A study of the FITS-IDI format for the Chinese Spectral Radio Heliograph. Astronomical Research & Technology Publications of National Astronomical Observatories of China, 388-395.

[5] Wei Shoulin, Shi Congming, Gao Jiaojiao, et al. A study on the real-time collection of the Vantage Pro weather station and the application in the MUSER. Astronomical Research & Technology, 117-123.

[6] Chen Meng, Wang Feng, Deng Hui, et al. Design and implementation of an astronomical command line interface system based on the Python. Astronomical Research & Technology, 196-203.

[7] Swartz A. Web.py Install Guide [EB/OL]. http://webpy.org/install.

[8] Cai Nengjian, Liu Donghao, Deng Hui, et al. The Application Method of Web.py in Developing MUSER Software System. Astronomical Research &

Technology, 229-235.

Note: Figure translations are in progress. See original paper for figures.