The Design of Embedded Web System based on REST Architecture

Yunwei Zhao ,Xin Wan
School of Information and Communication Engineering, Communication University of China
Beijing, China
15300233105@163.com, wallin82@cuc.edu.cn

***Abstract*—Aiming at the problems of high coupling, large resource waste and low expansibility in traditional embedded WEB system development mode, this paper presents an embedded WEB system development design based on REST style. In this paper, data of the system is abstracted as a resource. The front-end initiates a resource request by the improved AJAX request, and calls the RESTful API to realize resource transmission, and the server processes the client requests through the CGI program. The implementation of this scheme improves the versatility and expansibility of embedded WEB system, and reduces the memory cost of server. It also provides a low-cost design method for embedded systems with limited storage resources.**

***Keywords*—REST; CGI; AJAX; JSON; Embedded web system**

I. INTRODUCTION

With the rapid development of network technology, embedded web system development has become an important branch of information technology. The integration of web technology makes embedded system become more intelligent. People can not only observe the changes of device data, but also control and manage it remotely, which greatly improves the usability of embedded devices. In order to make full use of the limited resources, reduce the memory costs and improve the performance of the embedded system because of its' characteristics of small size and low resources. The design of the system should minimize the burden of embedded Web server in storage space and processing capacity as far as possible[1].

The traditional development mode of embedded web system has a high coupling degree between the front-end and back-end. Because the back-end also handles front-end pages besides data processing. So there are some problems such as high memory consumption, long development cycle, high costs for maintenance and poor expansion performance. In order to solve these problems, this paper proposes an embedded web system design scheme based on REST architecture. It abstracts system data into resources, adopts the front and rear separation structure, so that the front-end is only used for the business logic and pages, and the back-end is only used for resources processing. Using a RESTful API interface to transfer resources between front-end and back-end. Using this scheme, we can reduce the coupling, improve the performance and the expansibility of the system, and also ensure the maintenance is convenient.

II. TRADITIONAL DEVELOPMENT MODE

Early embedded web system development is composed of HTML FORM method, web server and CGI[2]as shown in Fig. 1. Due to the limited resources of embedded devices, we usually choose Boa as the web server, and use CGI to develop WEB service. Both of them have the characteristics of low resource consumption and high execution efficiency[3]. However, there still have some problems in this mode.

(1) High coupling: in this development model, besides data processing,CGI process also need to output HTML pages. CGI pro grams are usually developed in C, while front-end pages are developed in HTML. The combination of the two languages leads to high code coupling, and the functions of front-end and back-end cannot be separated, which makes the development cycle longer and the later maintenance inconvenient[4].

(2) The page response is not timely: using FORM method to request data, the update of the data needs to refresh the entire page[4].In the case of the page response is not timely, it will bring a bad interactive experience to the user.

(3) High memory consumption: because the traditional development mode is to insert the HTML language into CGI program to output HTML page directly. In the case of the front-end page content is rich, the number of pages is large, it will consume a lot of storage memory, waste equipment resources.

(4) Low versatility: due to the variety of client types, the development modes are different. It is inconvenient to develop multiple system to adapt varied types.

(5) Low expansibility: due to the tight coupling between the front-end and back-end, the later functional expansion of the system needs a large amount of work.

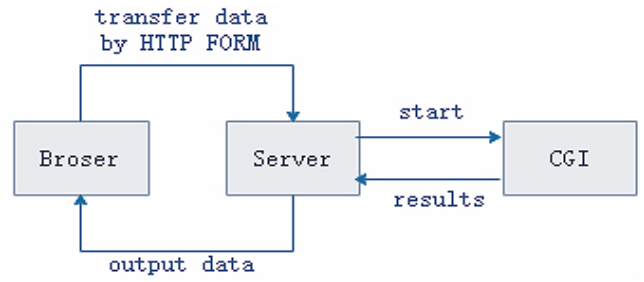

Fig. 1. Traditional embedded web system.

Then, with the development of AJAX technology, the problem of untimely page response was improved. AJAX

technology uses asynchronous data transmission between the browser and the Web server to make the Web page request a small amount of information from the server instead of the entire page. So that it can realize the local refresh of the page and improve the page's response speed[5]. They also separate the CGI program from HTML. In papers of WANG You-shun[4] and LYU Guo-yong[5], CGI is only used for data processing, and it will return data to HTML page in XML format, in order to realize web dynamic interaction.

Although AJAX technology has improved the previously existing code chaos and page-related problems, the system still has problems of poor expansibility, high resource consumption, and high coupling between front-end and back-end. To solve these problems, the REST architectural style caught our attention.

III. MAIN TECHNICAL PRINCIPLES

A. CGI

CGI (Common Gateway Interface) is an Interface standard between Web server and external application, which is often used for embedded Web development. It uses HTML FORM to send information to server, and exchange data with server by environment variables. In embedded system, only GET and POST methods are supported[6]:

(1) GET: Transfer data by URL. The server gets data from the environment variable QUERY_STRING. Due to the limitation of the length of the URL, the size of data should not exceed 1024 bytes;

(2) POST: Carry data by HTTP BODY. The server first obtains the length of data from the environment variable CONTENT_LENGTH, then gets the data from the standard input STDIN.

B. REST

REST[7] (Representational State Transfer) is an architectural style or design principle first proposed by Roy Fielding. It consists of a series of principle specifications to guide the web architecture design. The principle of REST is to abstract all the things in the system (such as data, video, audio, web pages, etc.) into resources, represented by URL, and the client uses HTTP methods (such as GET, POST, PUT, DELETE, etc.) to operate resources. After receiving the requests, the server will take different responses according to different methods (such as POST means modify or DELETE means delete), and finally return data to the client in XML, JSON or other formats[8].

Five principles for REST[9]:

(1) Unified resource identifier: all resources are identified by URL, and each resource has a unique global ID, so that resources can be located by URI.

(2) Manipulate resources with the standard methods: the operation of resources is realized by multiplexing the basic operations of the HTTP protocol, such as GET, POST, PUT, and DELETE which equals to querying, adding, modifying and deleting resources to ensure the versatility of the interface.

(3) Multiple representations of resources: REST supports multiple data formats such as XML, JSON, HTML, etc.

(4) All the resources are linked together: we should try to avoid designing isolated resources. In addition to designing resources themselves, we also need to design the relationship between resources and link resources through hyperlinks.

(5) Stateless communication: the Session State of the communication should be maintained entirely by the client.

C. RESTful API

The RESTful API is an API interface based on REST architecture. The resource is identified and located by URL and operated by HTTP methods (such as POST, GET, PUT, DELETE.etc). It is easy to process resources with RESTful API.

Compared with the front and rear end coupling development scheme, The RESTful API emphasizes the versatility of the interface. The front and rear end separation architecture using RESTful API has the advantages of low coupling, low complexity and high expansibility. However, due to the high resource requirements of the web development based on REST, and the limitation of the CGI on HTTP protocol, it is difficult to directly transplant the web development in REST style to the embedded system, it needs some changes.

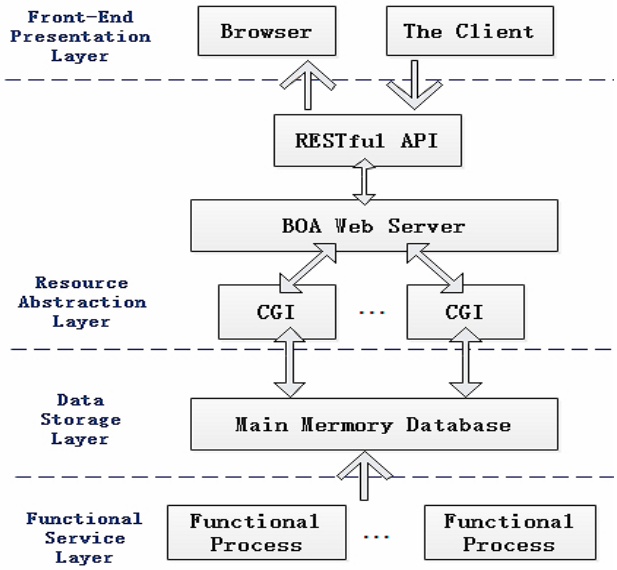

Fig. 2. System Framework Structure.

IV. DEVELOPMENT OF EMBEDDED WEB SYSTEM BASED ON REST ARCHITECTURE

A. System Architecture

According to different functions, the whole embedded WEB system can be divided into four parts which is shown in Fig. 2.

(1) Front-End Presentation Layer. It consists of a browser or a client and is mainly used for data display and user

interaction. Users can view and configure device information through the front-end page.

(2) Resource Abstraction Layer. It consists of RESTful API interface, WEB server and CGI process. It is the core of the whole embedded WEB development scheme. In this paper, the BOA server is used as the WEB server. The resource abstraction layer abstracts all data into different resources, identifies them with URL, and use HTTP methods to define resources' operation. Users can send resource requests to BOA server by calling RESTful API interface. When the server receives requests, it will start a CGI process to analyze requests, then returns the required data to client through API interface.

(3) Data Storage Layer. It consists of a main memory database, which is mainly used to store the data obtained from the functional service layer and return the corresponding data to CGI process according to the request from the resource abstraction layer.

(4) Functional Service Layer. It consists of different functional service modules in embedded devices, it performs actual functions and returns corresponding data. Such as receiving and returning broadcast data, executing algorithms.

B. System Work Flow

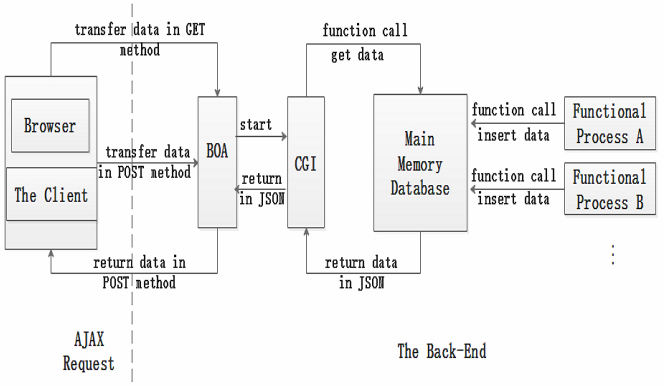

Fig. 3.The Work Flow of Embedded web system.

The work flow of system is shown in Fig. 3.

(1) Users send Ajax requests to WEB server through the browser or the client to call the RESTful API interface.

(2) After receiving requests, the WEB server starts a CGI process to parse the request to determine the required resource and resource operations. At the same time, other functional components of the back-end store data into database which are obtained from internal operation or external chip or sensor independently.

(3) The CGI process accesses, modifies or deletes relevant data from the main memory database by calling different functions according to the parsed resource information. Then returning the results to client in POST request with JSON format.

V. DESIGN AND IMPLEMENTATION OF API INTERFACE

A. RESTful API Interface

1) Resource Identification (URI)

According to the principles of REST, different business types and data can be abstracted as resources and represented by URI. Users can retrieve resources by URL. The URL is mainly composed of the following contents: network protocol (HTTP, HTTPS), server address, URI and parameter list.

URI should be a noun or plural noun, it cannot contain any verbs, such as add, create. And URI must be readable to reflect the meaning of the resource.

Parameter list is used to define the search condition to realize the functions of filtering, sorting, paging and so on.

2) Request Method (GET/POST)

The traditional RESTful API interfaces, use all HTTP methods (GET, POST, PUT, DELETE) to define the resources operations. However, because of the CGI process can only handle GET and POST method, we need some changes. Considering that PUT and DELETE are used to operate data on database. So in order to ensure the data privacy, this paper adopts the method of POST for PUT and DELETE requests. We can distinguish the three request methods of POST, PUT and DELETE by the added field "actions" in the data which carried by POST.

The GET method is used to get resource information, and the database retrieval condition is added in the form of key=value in the URL after '?' parameter list. The design of parameter is shown in Table I.

TABLE I. PARAMETER DESIGN

Parameters	**Meaning**
?uid=1001	Return data with ID=1001
?limit=10	Specify the number of records
?offset=10	Specify the starting position of records
?page=2&per_page=100	Specifies the page number and the number of records per page
?sortby=name&order=asc	Specify sort order

For example:192.168.43.54:8081/cgi-bin/CGI_TEST.cgi/FMDMCard?uid=1001

Meaning:Obtain data with uid of 1001 from FMDMCard data table

Since POST, PUT, and DELETE are all implemented by using the POST method, the data should be carried by HTTP BODY, and in JSON format. You can distinguish three different ways by the added field "actions". The work flow of CGI parses POST request is shown in Fig. 4 and the sample of methods design is shown in Table II.

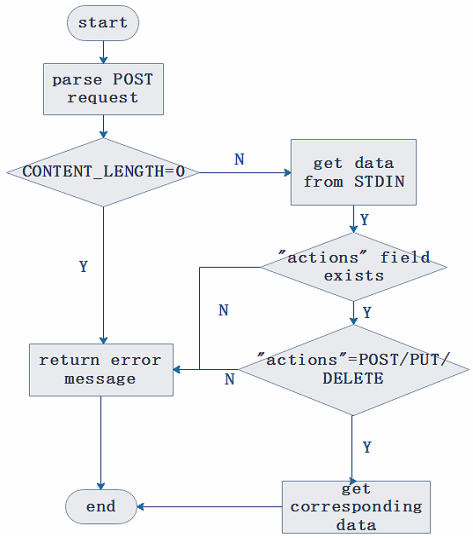

Fig. 4. The work flow of CGI parses POST request.

TABLE II. METHODS DESIGN

Methods	HTTP Requests	"actions" field	Example（JSON）	Meaning
POST	POST	actions=post	{"actions":"post", "DATA":"xxx"}	Add resources
PUT	POST	actions=put	{"actions":"put", "DATA":"xxx"}	Update resources
DELETE	POST	actions=delete	{"actions":"delete", "DATA":"xxx"}	Delete resources

3) Data Representation

In this paper, the JSON data format is used for both requests and responses to data resources. JSON data format is a lightweight data transmission format which represents data as a set of key-value pairs and divides the hierarchy of complex data structures by symbols[10] such as' [', '] ', '{' and'} '. In order to describe the results of the server and the status of the resource, we define HTTP status code to describe it. For example, 200 means the resource request is successful, 4001 means the resource does not exist, and 4002 means the POST format definition error. The design of data representation is shown in Fig.5.

```
//Request
    headers:
        Accept: application/json
        Content-Type: application/json;charset=UTF-8
    body:
        {
            "actions":"POST",
            "User_Name": "xxx",
            "User_Name": "xxx",
            "Sex":"female",
            "Email":"xxx@qq.com",
        }
//Response
    headers:
        Content-Type: application/json;charset=UTF-8
    body:
        {
        "success":"false",
        "error":"4008",
        "msg":"actions wrong"
        }
```

Fig. 5. Data Response Form In JSON.

4) Session Management

Since the HTTP protocol is stateless, it does not determine whether each request initiated is from the same user. Session management is proposed to let the server know which user initiated the HTTP request, so as to determine whether it has the rights to continue the request. Session Management usually use token as the user's identity credential which carried by cookie. In this paper, a main memory database is used to store the user's personal information include user ' s name, password, token, token expiration time and so on. The token is encrypted by user' s name, login time and random numbers.

The token will be stored in database when user login for the first time, and be transferred to browser by cookie. After that, each request initiated by user will take the token through cookie. Then the server will verify whether the token is correct and not expired to make sure user has rights to continue the request, so that it can ensure the security of the system.

B. Front-end data request in AJAX

In order to solve the problem that the delay of page response, the front-end data request adopts AJAX technology to realize the local refresh of the page and improve the response speed of the system. AJAX requests are typically called in three ways: $.ajax(), $.get(), $.post(), where $.ajax is a general way, and the other two are the specific request of the first one. The concrete implementation way is shown in Fig. 6.

```
// $.ajax
$.ajax({
    url:URL,
    type:'POST',
    dataType:"json",
    data:{},
    success : function(data) { }
    error :function(er){ }
});
// $.get
$(selector).get(url,data,success(response,status,xhr),dataType);
// $.post
jQuery.post(url,data,success(data, textStatus, jqXHR),dataType);
```

Fig. 6. Three implementation ways of AJAX Request.

However, because of the POST, PUT, and DELETE

requests are sent as POST, we need to make some changes to the way that AJAX calls. Firstly, the data type of the three calls defaults to JSON format. Secondly, $.ajax has only two modes, GET and POST. Thirdly, the POST method needs to add an "actions" field to in BODY to distinguish .

The improved AJAX request example is shown in Fig.7.

```
$ajax({
    url:"",//the requested URL type:" POST" ,
    //request modes, GET or POST
    dataType:"json",//data format
    data:{ 'actions' = 'put', … },
    success:function(){} // callback function
});
```

Fig. 7. Improved AJAX Request.

VI. PERFORMANCE ANALYSIS

Build Boa web server on the FETMX6UL-C kernel board, the CPU is NXP i.MX6UltraLite. This board is equipped with four cortex-a9 cores of a frequency up to 1.2GHZ for high

computing speed. The memory size is 512MB and the operating system is Linux 3.0.35. The experiment detects server performance and system response time by using Postman to simulate client sending requests to the web server, and then returns the data obtained from the database to the postman. It processed 200, 400, 800, 1000, 2000, 4000 and 6000 requests respectively and used TOP command to check the performance of BOA server as shown in Fig.8.

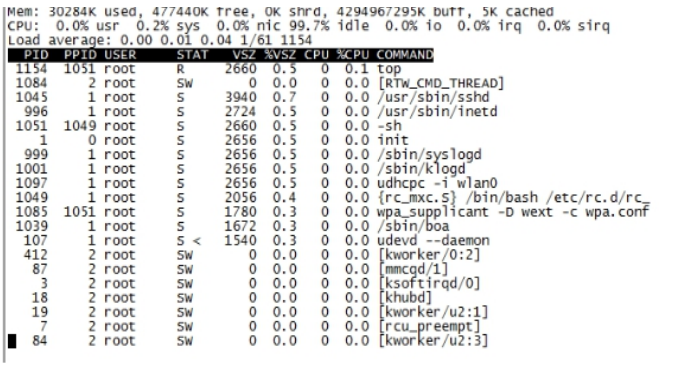

Fig.8. Use TOP Command to check server performance.

Table III shows the changes in server performance during the processing of data requests. According to the data in this table, it can be seen that the proportion of CPU occupied by the BOA server is hardly increased, and the BOA server runs only a small amount of memory resources, which is suitable for used by embedded devices with low resources.

TABLE III. BOA SERVER PERFORMANCE

Requests' number	Minimum memory usage	Maximum memory usage	Average memory usage
200	1.00%	1.31%	1.17%
400	1.01%	1.40%	1.19%
800	1.20%	1.63%	1.24%
1000	1.32%	1.75%	1.28%
2000	1.33%	1.81%	1.43%
4000	1.45%	1.96%	1.56%
6000	1.64%	2.13%	1.78%

TABLE IV. THE RESPONSE TIME OF DATA REQUESTS

Requests' number	Minimum response in system(ms)	Max response in system(ms)	Average response in system(ms)	Average response time in CGI (ms)
200	122	208	136	32
400	124	218	134	33
800	136	196	142	36
1000	132	228	156	37
2000	129	235	171	41
4000	144	248	167	50
6000	139	268	169	52

Table IV shows the change of system response time during the processing of data requests. Due to the difference in time caused by external factors such as network environment, this experiment not only records the overall response time of the system, but also records the running time of CGI program. According to the data in this table, it can be seen that the processing time of CGI program is very short and the response time of whole system is hardly increased. So the system can provide customers with efficient and fast data response.

VII. CONCLUSIONS

This paper proposed an embedded web system development model based on REST architecture style. It adopts the front and rear end separation structure and uses RESTful API interface to exchange data between the front-end and the back-end. In this paper, we focused on the design and implementation of the RESTful API interface.This part mainly uses the resource-oriented features of REST and the HTTP verbs, changes the data request way of AJAX , in order to improve the versatility and expansibility of embedded web systems.

After testing, this kind of development mode reduces the resource cost of the server on embedded device, and improves the response speed of the page. It provides a more convenient development mode of embedded WEB system.

REFERENCES

[1] Zhuo-jin Pan, Qiu-shi Wang, "Study of thin embedded web server based on AJAX and CGI," Computer Engineering and Design (CED). Vol.31, no.20, pp.4372,4374, 2010.

[2] Ya-kun Liu, Xiao-dong Cheng, "Design and Implementation of Embedded Web Server Based on ARM and Linux," 2010 2nd International Conference on Industrial Mechatronics and Automation.

[3] Bao-zhong Wang, Cheng-Ma, "Study and implementation of the embedded dynamic Web based on CGI technology," Electronic Design Engineering (EDE). Vol.20, no.18, pp.161,163, September 2012.

[4] You-shun Wang, Jun-bao Zhang, "The Web Application Research of Embedded Based on CGI and AJAX," Software Guide (SG). Vol.13, no.7, pp.90,92, July 2014.

[5] Guo-yong Lyu, Xiang-long Shi, "Web design of asynchronous interaction based on embedded Linux and Ajax," Journal of Computer Applications (JCA). Vol.33, no.S1, pp.247,251, June 2013.

[6] Jin-Zhu, Juan-juan Lei, Fu-cai Chen, "Design and implementation of embedded Web server based on CGI," Electronic Design Engineering (EDE). Vol.24, no.19, pp.191,193, October 2016.

[7] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," PhD Dissertation, Chapter 5, Dept. of Computer Science, Univ. of California, Irvine, 2000.

[8] Qian-Zhai, "Research on Key Technologies of Embedded Web Service Intelligent Food Safety Monitoring Node," Master Dissertation, Dept. of Information, Univ. of Shanghai Ocean, 2014.

[9] Xiao-hong Li, "Research and Development of Web of Things System based on REST Architecture," 2014 Fifth International Conference on Intelligent Systems Design and Engineering Application.

[10] Bo-ci Lin; Yan-Chen; Xu-Chen; Ying-ying Yu, "Comparision between JSON and XML in Applications on AJAX," 2012 International Conference on Computer Science and Service System.