

Report

by Hang Zuo

FILE	17711_HANG_ZUO_REPORT_254686_682112731.DOCX (347.37K)		
TIME SUBMITTED	14-MAY-2020 04:41PM (UTC+0100)	WORD COUNT	5164
SUBMISSION ID	127621961	CHARACTER COUNT	28864

Best Practice for RESTful Web Service
L7 – Web Service (ACCA7005)
L7 – Web Development (ACCA7004)

Student Name: Hang Zuo

Student Number: 1904646

Guide Teacher:

Dr Tim Bashford

Dr Michael Dacey

Date: 14/May/2020

I. CONTENTS

<i>I.</i>	<i>CONTENTS</i>	2
<i>II.</i>	<i>ABSTRACT</i>	3
<i>III.</i>	<i>INTRODUCTION</i>	4
<i>IV.</i>	<i>FOUNDATION</i>	4
<i>V.</i>	<i>BEST PRACTICES FOR RESTFUL WEB SERVICES</i>	5
<i>A.</i>	<i>Identification of Resources</i>	6
<i>B.</i>	<i>Use JSON</i>	6
<i>C.</i>	<i>Use plural nouns to name collections</i> <i>Order Table</i>	6
<i>D.</i>	<i>Interaction with Resources</i>	7
<i>E.</i>	<i>Caching</i>	7
<i>F.</i>	<i>Error handling</i>	8
<i>G.</i>	<i>Versioning</i>	8
<i>VI.</i>	<i>SUMMARY AND OUTLOOK</i>	8
<i>VII.</i>	<i>REFERENCES</i>	8
	<i>APPENDIX</i>	9

II. ABSTRACT

In this article, I propose best practices for RESTful web services, and illustrate the best practices with my own completed implementation. In this article, I have solved the problem and put forward seven best practices by referring to relevant materials, what we learned in class and our own practical understanding.

This study found that although there are no fixed rules for restful web services, theoretical knowledge should be combined in practice, which is very helpful for building an excellent web service.

Keywords-HTTP; Web; Service; RESTful; JSON (key words)

III. INTRODUCTION

Over the years, more and more REST-based web services^[1-3] have become widely available. Compared with traditional web services with Simple Object Access Protocol, REST service attracts more and more developers. Also a large number of famous companies, such as Amazon, Facebook and Twitter, are using REST for their services.

Although this trend is becoming more and more obvious, there is no clear de standard or guidelines on how to develop restful web services.

In this article, I first raised the question how to develop best restful web service. Then we will introduce web service and rest service respectively from the definition and analyze their similarities and differences at the same time. Finally, I will combine the implementation of online sales system I have done to answer this question.

IV. FOUNDATION

REST is design methods of online application, which was first proposed by fielding in his paper.

So, let me first introduce what is web service and what is REST service.

Web services are Web-based applications with platform independence, loosely coupled, self-contained, programmable features. Open XML^[5] standards can be used to describe, publish, discover, coordinate, and configure these applications, which are used to develop distributed and interoperable applications.

It has following main features:

(1) platform independent

First, the application that can be called Web Service has platform independence, so-called platform independence, which can be used in Windows, Linux, Unix platforms and so on.

(2) loosely coupled and self-contained

Second, web service applications are loosely coupled and self-contained. The so-called loose coupling means that the dependencies and constraints between modules are relatively small. Changing one module will not have a greater impact on other modules. To put it plainly, you do not need to care about how the modules are implemented inside. Give you standard interfaces, you can use the technology that everyone uses.

(3) Distributed Interoperability

Finally, distributed interoperability is easy to understand. To have interoperability with other systems, the system is not isolated. Open interfaces can be invoked on any platform without dependency.

However, with the rise of Internet technology, XML is becoming more and more criticized, XML packets become heavier, SOAP protocol is less convenient and flexible, advocated by large Internet companies such as Yahoo, Google and Facebook, REST (Representational State Transfer) has been widely accepted in the Web field, based on SOAP and Web Services Description. A simpler alternative to Language's Web services.

And what is REST?

REST -- Representational State Transfer

Rest is a design and development method for network application, which can reduce the complexity of development and improve the scalability of the system.

Rest is a design and development method, which has the characteristics of network application. Rest also has many outstanding advantages, for example, it can reduce the difficulty of development and improve the scalability of the system.

Features of REST

- (1) Client and server architecture
- (2) Connection protocol is stateless
- (3) Capable of leveraging Cache mechanisms to improve performance
- (4) Hierarchical System

REST Service is lighter than Web Service's recommended standards and can even be invoked using JavaScript, making it easier, more efficient, and easier to use.

The REST architecture follows the CRUD^[7-10] principle, which requires only four behaviors for a resource: Create, Read, Update, and Delete to complete its operation and processing.

Conclusion

REST looks at the entire network from a resource perspective, and the resources distributed everywhere are determined by URI. Client applications use URI to obtain the representation of resources. Obtaining these representations transforms the state of these applications. With the constant acquisition of representations of resources, client applications are constantly changing their state. That is what we called Representational State Transfer.

Through the above analysis, rest service has all the characteristics of Web Service: platform independence, loose coupling and interoperability. And rest is lighter and simpler. It can be said that rest service is an implementation of web service, not a replacement of web service.

V. BEST PRACTICES FOR RESTFUL WEB SERVICES

Through the analysis of last section, I will summarize my understanding of REST.

1).What does REST do?

The URL locates the resource and describes the operation with HTTP verbs (get, post, delete.).

2).Why we choose RESTful API?

We all know that "ancient" web pages are front-end and back-end^[4,6,9], such as PHP, JSP and so on. In the previous desktop era, the problem is not big, but in recent years, with the development of mobile Internet, various types of clients emerge in endlessly. Restful can provide services for web, IOS and Android through a set of unified interface. In addition, for a wide range of platforms, such as Facebook platform, Weibo open platform, Wechat public platform, etc., they do not need an explicit front-end, only a set of service interface, so restful is their best choice. Under the restful architecture:

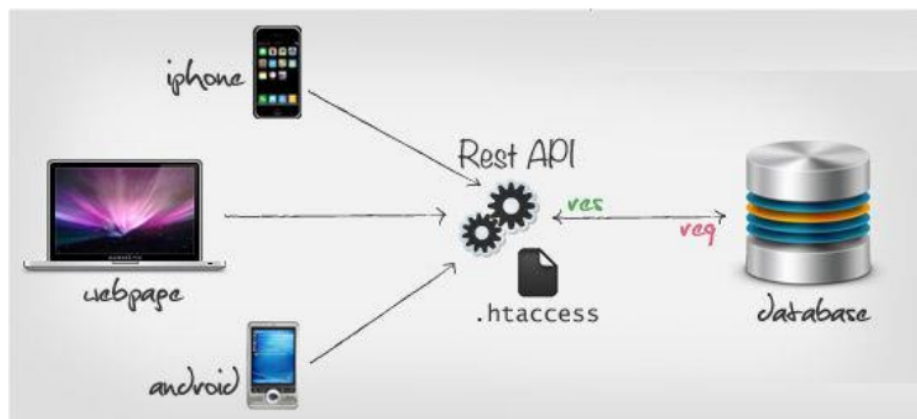


Figure 1. RESTful architecture

So Use REST we can achieve:

- 1) If you look at URL, you know what you want.
- 2) See HTTP method to know what to do.
- 3) Look at the HTTP status code to see the results.

We can see it clearly that REST plays an important role in developing the web. In this part, I will focus on the best practices of developing restful web services in combination with my completed online sales system.

A. Identification of Resources

URI is an important part of web services because it is difficult to change, once the service is published and used by the client.

So when we design the URI, we should follow the following best practices:

- 1) URI design should be as concise as possible.
- 2) Easy to remember and clear definition without ambiguity.
- 3) URI consists of nouns only.
- 4) Generally speaking, a resource consists of at most two URIs, in which the first URI address represents the specific state of a specific resource and another specific resource.

```
1. @DeleteMapping("/goods/{id}")
2. public Integer deleteGoodsLists(@PathVariable("id") Long id) {
3.     return goodsService.deleteGoods(id);
4. }
```

Figure 2. Example for URI design

B. Use JSON

You can know that rest supports different output formats, such as plain text, JSON, XML, and even HTML, by consulting relevant materials. Of course, it may depend on the circumstances. We should choose JSON^[6] as the output format of communication, because if you have a public oriented service, you definitely want to access it through the rest API. More importantly, application / JSON is a common MIME type, making it a useful use.

```
1. //Request method
2. type: "GET",
3. //Requested media type
4. contentType: "application/json; charset=UTF-8",
5. xhrFields: {'Access-Control-Allow-Origin': '*'},
6. dataType: "json",
7. //Request Address, baseUrl=localhost/8080
8. url: `http://${baseUrl}/goodsFi/${detailId}`,
```

Figure 3. Example for Use JSON

C. Use plural nouns to name collectionsOrder Table

Plural nouns should be used when developing the REST API. This is not only a good habit, but it

also makes it easier for people to understand that something is actually a collection.

We use goods rather than good in the figure below.

```
1. @PostMapping("goods")
2. public Integer saveGoods(@RequestBody Goods goods) {
3.     return goodsService.save(goods);
4. }
```

Figure 4. Example for Use plural nouns

D. Interaction with Resources

Through the previous chapters, the essence of RESTful Web Service is to use URL to locate resources, and then use HTTP verbs(POST, GET, PUT, DELETE) to complete operations on resources.

In the process of building a RESTful web service, the interaction between the client and the resource is not simply obtained directly. The interaction between client and server is based on the application layer Protocol HTTP, which plays an important role in communication. When interacting using the HTTP protocol, I recommend the following best practices:

- 1) When we use HTTP methods, we should follow the definition in the official HTTP specification.
- 2) The support of http-options is very important, especially when we need to transfer a large amount of data.
- 3) While developing HTTP-based service, we should consider conditional GET method.

```
1. @GetMapping("/goods/{id}")
2. @PostMapping("goods")
3. @DeleteMapping("/goods/{id}")
```

Figure 5. Example of using HTTP methods

The table below shows the characteristics of most used HTTP methods.

TABLE 1.Characteristics of HTTP methods

HTTP method	safe	idempotent
POST	NO	NO
GET	YES	YES
PUT	NO	YES
DELETE	NO	YES

E. Caching

The server should only cache successful get requests in order to achieve good performance. In addition, it is recommended that we can cahce in multiple layers.

```
1. // Add methods to `ListCache`.
2. ListCache.prototype.clear = listCacheClear;
3. ListCache.prototype['delete'] = listCacheDelete;
4. ListCache.prototype.get = listCacheGet;
5. ListCache.prototype.has = listCacheHas;
6. ListCache.prototype.set = listCacheSet;
```

Figure 6. Example for caching

F. Error handling

People must be annoyed when they see something wrong. So we can provide information about the error, let them know what happened, why the error occurred, and how to solve it. Therefore, correct handling of errors is an important practice in the development of RESTful web service.

```
1. public HttpRetryException(String detail, int code) {  
2.     super(detail);  
3.     responseCode = code;  
4. }
```

Figure 7. Example for error handling

The figure above gives an example showing how to deal with the error^[3,4].

Table 2. Implication of HTTP status code

HTTP status code	implication
200	Request successful
201	Request has been created
202	Request has been accepted
400	Bad request
401	Unauthorized request

G. Versioning

Versioning the RESTful API is very important, which allows me to do specific operations or change the structure of the data as I want, even if they are broken. However, sometimes, there may be multiple versions of API. For many users, it will take a long time to adapt.

VI. SUMMARY AND OUTLOOK

In this paper, I put forward the best practice for restful web design by consulting the relevant literature, and I combined the online sales system I completed to illustrate the specific use of the best practice.

In my future study, I intend to study best practices for improving development speed, and evaluate the practicability of restful web services best practices through some indicators.

I know it's very difficult for me, but the hard work is challenging, which can stimulate my morale and hope to gain more knowledge in the future study.

Finally, I would like to thank the teacher for his teaching, the wonderful lectures in the class, and the answers to our questions.

VIII. REFERENCES

- [1] A. Navarro and A. da Silva, 'A metamodel-based definition of a conversion mechanism between SOAP and RESTful web services.', *Computer Standards & Interfaces*, vol. 48, pp. 49–71, Nov. 2016, doi: 10.1016/j.csi.2016.03.004.
- [2] P. Giessler, M. Gebhart, D. Sarancin, R. Steinegger, and S. Abeck, 'Best Practices for the Design of RESTful Web Services', p. 7, 2015.

- [3] C. Zolotas, T. Diamantopoulos, K. Chatzidimitriou, and A. Symeonidis, 'From requirements to source code: a Model-Driven Engineering approach for RESTful web services', *Automated Software Engineering*, vol. 24, no. 4, pp. 791–838, 2017, doi: 10.1007/s10515-016-0206-x.
- [4] P. Mestre, R. Madureira, P. Melo-Pinto, and C. Serodio, 'Multiple JSON Web Tokens for Mobile Distributed Applications.', *Engineering Letters*, vol. 26, no. 2, pp. 281–287, Jun. 2018.
- [5] Adi Nugroho and Khabib Mustofa, 'PERBANDINGAN ANTARA "BIG" WEB SERVICE DENGAN RESTFUL WEB SERVICE UNTUK INTEGRASI DATA BERFORMAT GML', *Jurnal Informatika*, vol. 11, no. 1, pp. 8–16, 2012, doi: 10.9744/informatika.11.1.8-16.
- [6] O. DOSPINESCU, C. STRÎMBEI, R.-M. STRAINU, and A. NISTOR, 'REST SOA Orchestration and BPM Platforms.', *Informatica Economica*, vol. 21, no. 1, pp. 30–43, Mar. 2017, doi: 10.12948/issn14531305/21.1.2017.03.
- [7] R. Casella, 'RESTful Web Services at BNL', 2011, Accessed: May 13, 2020. [Online]. Available: <http://digital.library.unt.edu/ark:/67531/metadc829571/>.
- [8] M. Arroqui, C. Mateos, C. Machado, and A. Zunino, 'RESTful Web Services improve the efficiency of data transfer of a whole-farm simulator accessed by Android smartphones', *Computers & Electronics in Agriculture*, vol. 87, pp. 14–19, Sep. 2012, doi: 10.1016/j.compag.2012.05.016.
- [9] S. Kennedy, R. Stewart, P. Jacob, and O. Molloy, 'StoRHm: a protocol adapter for mapping SOAP based Web Services to RESTful HTTP format', *Electronic Commerce Research*, vol. 11, no. 3, pp. 245–269, 2011, doi: 10.1007/s10660-011-9075-3.
- [10] M. Islam, F. Belqasmi, R. Glitho, and F. Khendek, 'The design and implementation of OMA restful location services in wireless sensor environments.', *IEEE Communications Magazine*, vol. 51, no. 4, pp. 122–132, Apr. 2013, doi: 10.1109/MCOM.2013.6495771.
- [11] M. P. Papazoglou, *Web services & SOA: principles and technology*, 2nd ed. Harlow: Pearson, 2012.
- [12] M. Breeding, 'Web Services in the Library Environment.', *Library Technology Reports*, vol. 42, no. 3, pp. 36–47, May 2006.

APPENDIX

Since I have made some simple introductions on the best practices of restful web services in the previous section, in the appendix section, I only attach the relevant codes of the restful interface in the back end.

GoodsController.java

```

1. import java.util.List;
2. @RestController
3. @CrossOrigin(origins = "*", maxAge = 3600)
4. public class GoodsController {
5.     @Autowired
6.     private IUserService userService;
7.     @Autowired
8.     private IGoodsService goodsService;
9.     /**
10.      * check user information
11.      *
12.      * @param user

```

```

13.     * @return
14.     */
15.     @PostMapping("/user")
16.     public User checkUserPermission(@RequestBody User user) {
17.         return userService.checkUserPermission(user);
18.     }
19.     @PostMapping("/goods/search")
20.     public Page query(@RequestBody(required = false) SearchCondition searchC
ondition) {
21.         return goodsService.query(searchCondition);
22.     }
23.     @GetMapping("/goods/{id}")
24.     public Goods queryById(@PathVariable("id") Long id) {
25.         return goodsService.selectById(id);
26.     }
27.     @PostMapping("/goods")
28.     public Integer saveGoods(@RequestBody Goods goods) {
29.         return goodsService.save(goods);
30.     }
31.     @DeleteMapping("/goods/{id}")
32.     public Integer deleteGoodsLists(@PathVariable("id") Long id) {
33.         return goodsService.deleteGoods(id);
34.     }
35.     @PostMapping("/goods/buy")
36.     public String buyArticle(@RequestBody PurchaseParam purchaseParam) {
37.         return goodsService.purchaseGoods(purchaseParam);
38.     }
39.     @PutMapping("/goods")
40.     public Integer updateById(@RequestBody Goods goods) {
41.         return goodsService.updateById(goods);
42.     }
43. }

```

FileController.java

```

1. @RestController
2. @CrossOrigin(origins = "*", maxAge = 3600)
3. public class FileController {
4.     @PostMapping(value = "/uploadFile")
5.     public String uploadFile(@RequestParam("file") MultipartFile multipartFi
le) {
6.         String fileName = "";
7.         try {
8.             String uuid = UUID.randomUUID().toString();

```

```

9.          // file's location in server
10.         int index = multipartFile.getOriginalFilename().lastIndexOf(".")
        ;
11.         if (index < 0) {
12.             return "upload error";
13.         } else {
14.             fileName = uuid + "." + multipartFile.getOriginalFilename().
            substring(index + 1);
15.             System.out.println(fileName);
16.         }
17.         String rootPath = this.getClass().getResource("/").getPath() + "
        static/img";
18.         File dir = new File(rootPath);
19.         if (!dir.exists()) {
20.             dir.mkdirs();
21.         }
22.         // upload the file to the server
23.         String path = dir.getAbsolutePath() + File.separator + fileName;

24.         File serverFile = new File(path);
25.         System.out.println("path is " + path);
26.         multipartFile.transferTo(serverFile);
27.     } catch (Exception e) {
28.         e.printStackTrace();
29.     }
30.     return fileName;
31. }
32. }

```

“Serverless” technologies and Circuit breaker

L7 – Web Service (ACCA7005)

L7 – Web Development (ACCA7004)

Student Name: Hang Zuo

Student Number: 1904646

Guide Teacher:

Dr Tim Bashford

Dr Michael Dacey

Date: 14/May/2020

I. CONTENTS

<i>I. CONTENTS</i>	<i>13</i>
<i>II. ABSTRACT</i>	<i>14</i>
<i>III. SERVERLESS TECHNOLOGY</i>	<i>15</i>
<i>A. Framework</i>	<i>15</i>
<i>B. Web IDE</i>	<i>16</i>
<i>C. Command line tools</i>	<i>16</i>
<i>D. Application scenario</i>	<i>16</i>
<i>E. Comparison of different serverless services</i>	<i>16</i>
<i>F. Serverless and it's relevance to Web hosting</i>	<i>17</i>
<i>IV. THE CIRCUIT BREAKER PATTERN</i>	<i>18</i>
<i>V. REFERENCES</i>	<i>22</i>

II. ABSTRACT

In this article, I first introduced what is Serverless technology, and explained the technology from the use scenarios and other aspects. Then, I illustrates the connection between serverless technology and web hosting. In the second part of the article I introduced the Circuit Breaker design pattern, explained its principle, and finally demonstrated its important application in web development through an example. My main research methods are literature research and practice. By looking up relevant materials, I have learned something I can't learn in class, and I have a deeper understanding of my major.

Keywords-Serverless; Web; Circuit Breaker pattern; Web hosting(key words)

III. SERVERLESS TECHNOLOGY

Serverless technology refers to the concept of building and running applications that do not require server management.

For example, CDN^[1-4] (Content Delivery Network) play the role of escorts and accelerators, enabling them to quickly trigger information and reach every user, resulting in a more extreme experience. when we publish static resources to a CDN, we don't need to care about how many nodes the CDN has, how the nodes are distributed, how it balances the load, how it achieves network acceleration, so CDN is Serverless for the front end.

Of course, we still need a more precise definition. Technically, Serverless is the combination of FaaS and BaaS.

Serverless = FaaS + BaaS.

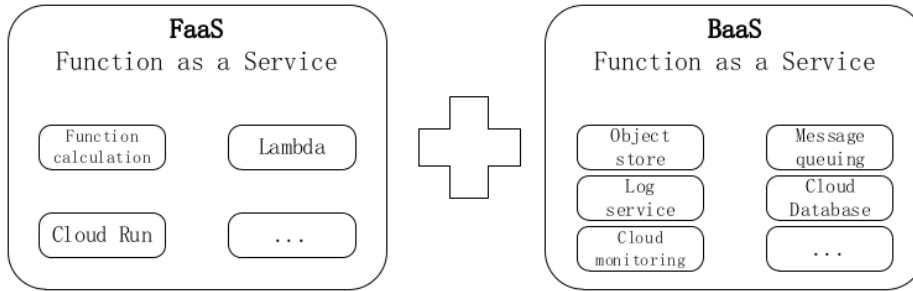


Figure 1. Composition of serverless

In short, FaaS^[2,3] (function as a service) is a platform for running functions, such as Alibaba cloud's function computing and AWS's lambda.

However, BaaS (backend as a service) is some back-end cloud services, such as cloud database, object storage, message queuing, etc. Using baas can greatly simplify the difficulty of our application development.

Main features of Serverless:

- 1) event driven.
- 2) Functions in the FAAS platform need to drive function execution through a series of events.
- 3) Stateless.
- 4) Because each time a function is executed, it may use a different container and cannot share memory or data. If you want to share data, you can only use third-party services, such as redis.
- 5) No operation and maintenance .
- 6) With serverless, we don't need to care about the server or the operation and maintenance. This is also the core of serverless.
- 7) low cost.
- 8) The cost of using serverless is low because we only need to pay for each function run. If the function does not run, it will not cost money or waste server resources.

I'll detail Serverless in the following ways:

H. Framework

Now there is no uniform serverless standard. The serverless services provided by different cloud computing platforms are likely to be different, which leads to our code cannot be smoothly migrated.

One of the main functions of the serverless framework is to simplify the development and deployment process of serverless. The other is to shield the differences among different serverless services, so that our functions can run in other serverless services without changing or only changing a small part.

Common Serverless frameworks are Serverless Framework, ZEIT Now, Apex, and so on.

I. Web IDE

The web ide closely related to serverless is also the web ide of each cloud computing platform. With web IDE, we can easily develop and debug functions in the cloud, and directly deploy them to the corresponding FAAS platform. This advantage is to avoid installing various development tools and configuring various environments locally. Common web ide include Cloud 9 of AWS^[3,4,5], functional computing web ide of Alicloud, and cloud studio of Tencent cloud. From experience, AWS cloud9 is the best.

J. Command line tools

Of course, at present, the most important way of development is to develop locally. Therefore, it is necessary to develop the command line tools^[4] of Serverless locally.

Here are a few examples of command-line tools.:

Create

```
# serverless
$ serverless create --template aws-nodejs --path myService
# fun
$ fun init -n qcondemo helloworld-nodejs8
```

Deploy

```
# serverless
$ serverless deploy
# fun
$ fun deploy
```

Debugging

```
# serverless
$ serverless invoke [local] --function functionName
# fun
$ fun local invoke functionName
```

K. Application scenario

On the upper layer of development tools, there are some vertical application scenarios of serverless. In addition to the traditional server-side development, there are currently small program development using serverless technology, and the Internet of things (IOT)^[6] may be designed in the future.

L. Comparison of different serverless services

The table below compares different Serverless services in terms of supporting languages, triggers, and so on. You can see the differences and similarities.

For example, almost all Serverless services support languages such as Node.js/Python/Java.

However, Triggers are also related to the platform's own back-end services, such as Ali Cloud's Object Storage Trigger, which is triggered based on events such as access to Ali Cloud's OSS products; while

AWS's Object Storage Trigger is triggered based on AWS's S3 events, the two platforms are not universal. This is also a problem Serverless is currently facing, that is, standards are not uniform.

Table 1. Comparison of different serverless services

Features	Alibaba cloud functional computing	AWS Lambda	Azure Functions
Language support	Node.js/ Python/ PHP/ Java/ C#	Node.js/ Java/ Go/ PowerShell/ C#/ Python/ Ruby	Node.js/ C#/ F#/ Java/ Python
Trigger	OSS/ API gateway/ HTTP/ CDN/ Log/ RDS/ MNS	S3/ DynamoDB/ SNS/ SES/ SQS/ Cognito/ CloudFormation	Blob Storage/ Cosmos DB/ Event Grid/ Event Hubs/ HTTP & Webhooks
Edge calculation	IoT Edge	CloudFront	IoT Edge
Http(s) call	API gateway / HTTP trigger	API gateway	HTTP trigger
Function group	Support	Not support	Not support
Development Tool	Fun/ fdi/ Web IDE	AWS SAM CLI/ Cloud9 IDE	Azure CLI



M. Serverless and it's relevance to Web hosting

In the previous section, I compared the differences between Alibaba cloud functional computing, AWS Lambda, and Azure^[1] Functions. In this section, I'll give you a more detailed description of AWS Lambda and its relevance to Web hosting.

AWS Lambda is a Function-as-Servcie computing service.In short, functions, functions, and services are written directly by developers and run on the cloud.We don't need to focus on the basic environment configurations of operating systems, operating environments, gateways. We just need to focus on writing our business code.

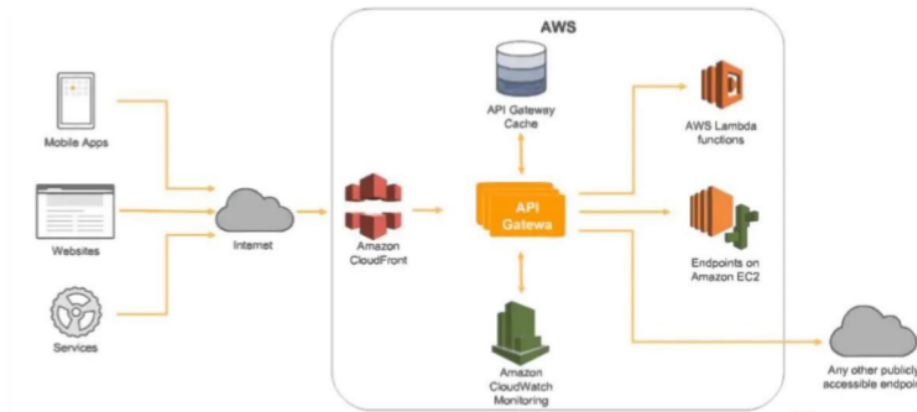


Figure 2 . How API Gateway works

Amazon API Gateway is a fully hosted API gateway service that helps developers easily create, publish, maintain, monitor, and protect APIs of any size. It integrates many API gateway functions, such

as caching, user authentication, and so on. It also supports configuration through HAML and Waggon, which allows you to configure the API with a code management system.

Amazon API Gateway can pass the requested data to different resources for processing based on different RESTful API access points.

When a request accesses an application through a domain name, the application forwards the HTTP request to the CDN (CloudFront).

CloudFront forwards the corresponding API request to the API Gateway according to the forwarding rules.

API Gateway handles requests for access points and content to the corresponding AWS Lambda or EC2 services, or to other accessible services.

The result of the request will be returned to the client after processing is completed. When returned, the API Gateway can also process the returned content through Lambda.

This integration of API Gateway and Lambda provides lighter-weight services than traditional Web services. By planning API access and completing function development, the team can quickly build the simplest service, reducing the time to build the micro-service infrastructure from a few weeks to a few hours. In addition, it greatly improves the efficiency and stability of service architecture development.

Conclusion

In this chapter, I first introduce what Serverless technology is, and then explain in detail the framework, command line tools, scenarios, and so on. Finally, the connection between Serverless and Web hosting is illustrated in conjunction with AWS Lambda. Finally, I conclude that Serverless greatly reduces the threshold for infrastructure configuration and maintenance, but it also presents new challenges, such as the management of a large number of functions, the testing of infrastructure integration, and so on. These problems are also worth pondering.

IV. THE CIRCUIT BREAKER PATTERN

At the beginning of this chapter, I'll start by explaining what The Retry pattern^[5-7] is. Let's look at the figure below.

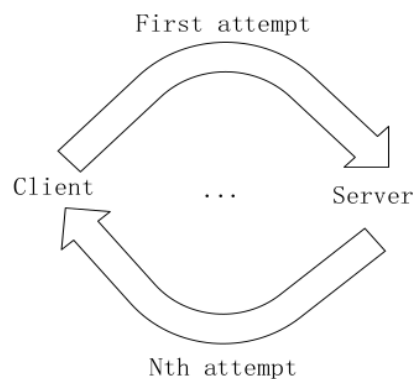


Figure 3. The Retry Pattern

The Retry pattern can handle some expected temporary errors when calling remote services or network resources by repeating previously failed operations. The Retry^[8,9] pattern can improve the stability of the application.

When a request is sent to an external service (such as a database, Webservice), if there are

some network problems, we send the request again until the request succeeds or the maximum number of attempts has been made.

The figure below shows how the Circuit Breaker pattern works. We can see it clearly that the Circuit Breaker pattern is an improvement on Retry mode. In Retry mode, when something goes wrong, we constantly try to send requests again. If external services cannot be restored in a short time, this is undoubtedly a waste of effort and resources. So Circuit Breaker pattern takes a break (that is, stop trying, Sleep for a few seconds) after a certain number of failures, and after a while, try again, if it fails, Sleep again, and try again until it succeeds.

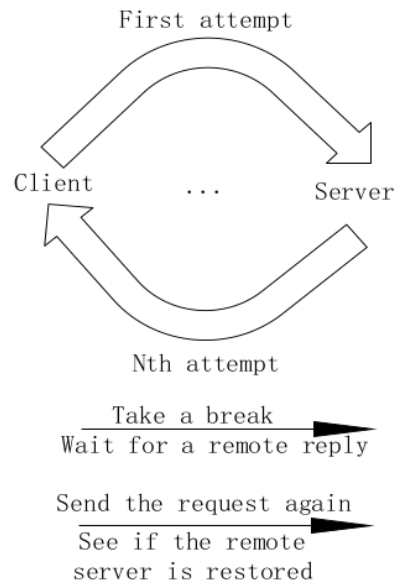


Figure 4. The Circuit Breaker Pattern

The Circuit Breaker pattern plays an important role in fault tolerance. In different microservices, it can effectively prevent further error propagation.

There are also many benefits to using the circuit breaker mode:

- 1) Ability to block operations that are doomed to fail.
- 2) Handle errors quickly.
- 3) Avoid large requests for services.
- 4) Callers do not have to handle failures themselves.

Next, I will further introduce the Circuit Breaker design pattern, and explain its rationale and how to use it in web development.

As we can see the figure below, I built two APIs, ServiceA and ServiceB, where /data depends on ServiceB and /data2 does not depend on external services. Through this model, I will explain the principle of Circuit Breaker design pattern and its application in Web development.

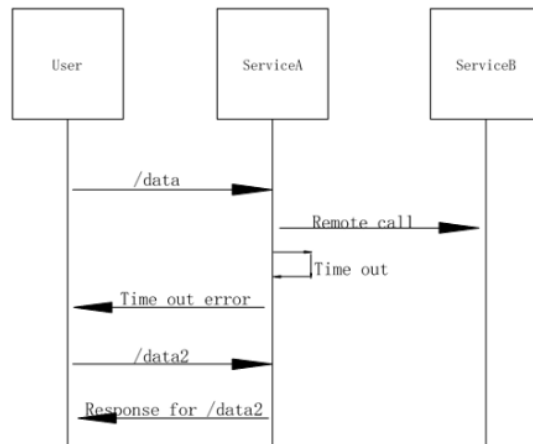


Figure 5. Example for Circuit Breaker pattern

ServiceB is implemented as follows API delays response by 5 seconds to requests for the first five minutes.

```

1. server.route({
2.   method: 'GET',
3.   path: '/flakycall',
4.   handler: async (request, h) => {
5.     const currentTime = Date.now();
6.     if ((currentTime - serverStartTime) < (1000 * 60 * 5)) {
7.       const result = await new Promise((resolve) => {
8.         setTimeout(() => {
9.           resolve('This is a delayed repsonse');
10.        }, 5000);
11.      });
12.      return h.response(result);
13.    }
14.    return h.response('This is immediate response');
15.  },
16. });
  
```

Figure 6. Implementation of Service B

When I use serviceA to make an HTTP request to serviceB.

```

1. server.route({
2.   method: 'GET',
3.   path: '/data2',
4.   handler: (request, h) => {
5.     try {
6.       return h.response('data2');
7.     } catch (err) {
  
```

```

8.     throw Boom.clientTimeout(err);
9.   }
10. },
11. });
12. server.route({
13.   method: 'GET',
14.   path: '/data',
15.   handler: async (request, h) => {
16.     try {
17.       const response = await axios({
18.         url: 'http://0.0.0.0:8000/flakycall',
19.         timeout: 6000,
20.         method: 'get',
21.       });
22.       return h.response(response.data);
23.     } catch (err) {
24.       throw Boom.clientTimeout(err);
25.     }
26.   },
27. });

```

Figure 7. HTTP request sent to ServiceB

We will use *jMeter* to simulate the load. Within seconds, serviceA will run out of resources. All requests are waiting for the HTTP request^[2,3,6,7,8] to complete. The first API will start throwing errors, and it will eventually crash because it will reach the maximum heap limit.

Now we have two services that don't work, not one. This will upgrade the entire system, causing the entire infrastructure to collapse.

If our serviceB fails, serviceA should still try to recover from it and try to do one of the following:

- 1) Custom fallback
- 2) Fast failure
- 3) Other APIs continue to work and automatically repair

That's why we need to use the Circuit Breaker pattern. In fact, in this model, the idea of Circuit Breaker design is very simple:

- 1) Once ServiceA "knows" that ServiceB fails, there is no need to request ServiceB. ServiceA should return cached data or timeout errors as soon as possible.
- 2) Once ServiceA "knows" ServiceB, we can turn off the circuit so that ServiceB can make a request again.
- 3) Make regular new calls to serviceb to see if the results are returned successfully. This state is half-open.

Let's implement a method for circuitBreaker to make GET http calls.

```

1. class CircuitBreaker {
2.   constructor(timeout, failureThreshold, retryTimePeriod) {
3.     // We start in a closed state hoping that everything is fine

```

```

4.     this.state = 'CLOSED';
5.     // Number of failures we receive from the depended service before we cha
       nge the state to 'OPEN'
6.     this.failureThreshold = failureThreshold;
7.     // Timeout for the API request.
8.     this.timeout = timeout;
9.     // Time period after which a fresh request be made to the dependent
10.    // service to check if service is up.
11.    this.retryTimePeriod = retryTimePeriod;
12.    this.lastFailureTime = null;
13.    this.failureCount = 0;
14.  }
15. }

```

Figure 8. A method for circuitBreaker

Then modify ServiceA and serviceb respectively. When we run our JMeter test again, we can see that our ServiceA has not crashed, and our error rate has decreased significantly.

Conclusion

In this section, I went from retry design pattern to Circuit Breaker design pattern and explained what is Circuit Breaker design pattern. Then through an example, the principle of Circuit Breaker design pattern and its important role in web development are illustrated.

V. REFERENCES

- [1]D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, and G. Moltó, 'Accelerated serverless computing based on GPU virtualization.', *Journal of Parallel & Distributed Computing*, vol. 139, pp. 32–43, May 2020, doi: 10.1016/j.jpdc.2020.01.004.
- [2]P. A. Witte, M. Louboutin, H. Modzelewski, C. Jones, J. Selvage, and F. J. Herrmann, 'An Event-Driven Approach to Serverless Seismic Imaging in the Cloud.', *IEEE Transactions on Parallel & Distributed Systems*, vol. 31, no. 9, pp. 2032–2050, Sep. 2020, doi: 10.1109/TPDS.2020.2982626.
- [3]J. Dean, A. Harrison, R. N. Lass, J. Macker, D. Millar, and I. Taylor, 'Client/server messaging protocols in serverless environments', *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1366–1379, 2011, doi: 10.1016/j.jnca.2011.03.021.
- [4]V. Malcher, 'Design Patterns in Cloud Computing', 2015, pp. 32–35, doi: 10.1109/3PGCIC.2015.142.
- [5]S. Ivanova, 'Designing an Educational Game for a Web Design and Development Course.', *eLearning & Software for Education*, vol. 1, pp. 287–293, Jan. 2018, doi: 10.12753/2066-026X-18-039.
- [6]Chao Fu, Jungang Zhou, Xiaoyan Zhang, and Chunyan Cui, 'Electromagnetic Compatibility Design of Intelligent Circuit Breaker', 2011, vol. 2, pp. 19–21, doi: 10.1109/ICMTMA.2011.292.
- [7]J. Enes, R. R. Expósito, and J. Touriño, 'Real-time resource scaling platform for Big Data workloads on serverless environments.', *Future Generation Computer Systems*, vol. 105, pp. 361–380, Apr. 2020, doi: 10.1016/j.future.2019.11.037.
- [8]M. Asay, 'Serverless computing may kill Google Cloud Platform', *Computerworld Hong Kong*, 2017.
- [9]Y. Kim and J. Lin, 'Serverless Data Analytics with Flint', *arXiv.org*, 2018, Accessed: May 14, 2020. [Online]. Available: <http://search.proquest.com/docview/2071828788/?pq-origsite=primo>.

[10]C. R. Jaimez-González and M. Castillo-Cortes, 'Web Application to Support the Learning of Programming Through the Graphic Visualization of Programs.', *International Journal of Emerging Technologies in Learning*, vol. 15, no. 6, pp. 33–50, Jun. 2020, doi: 10.3991/ijet.v15i06.12157.

[11]C. Preimesberger, 'Why Serverless May Become a Dominant Programming Paradigm.', *eWeek*, p. N.PAG, Nov. 2019.

Report

GRADEMARK REPORT

FINAL GRADE

88/100

GENERAL COMMENTS

Instructor

Implementation (50): 43
REST (25): 22

Serverless (25): 23

Feedback

Please do not use superscripts for reference.

Good discussion of REST, would liked to have seen more analysis of its benefits and drawbacks.

Very nice overview of FaaS and circuit breaker.

Very good implementation demonstrating a good understanding of both the webservice and AJAX client side.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7



Comment 1

...and this means??

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17



Comment 2

Reference

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23
