

Report

by Wenhao Kang

FILE	17875_WENHAO_KANG_REPORT_254686_2035456343.DOCX (200.79K)		
TIME SUBMITTED	15-MAY-2020 04:27PM (UTC+0100)	WORD COUNT	3473
SUBMISSION ID	127807225	CHARACTER COUNT	18797



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David

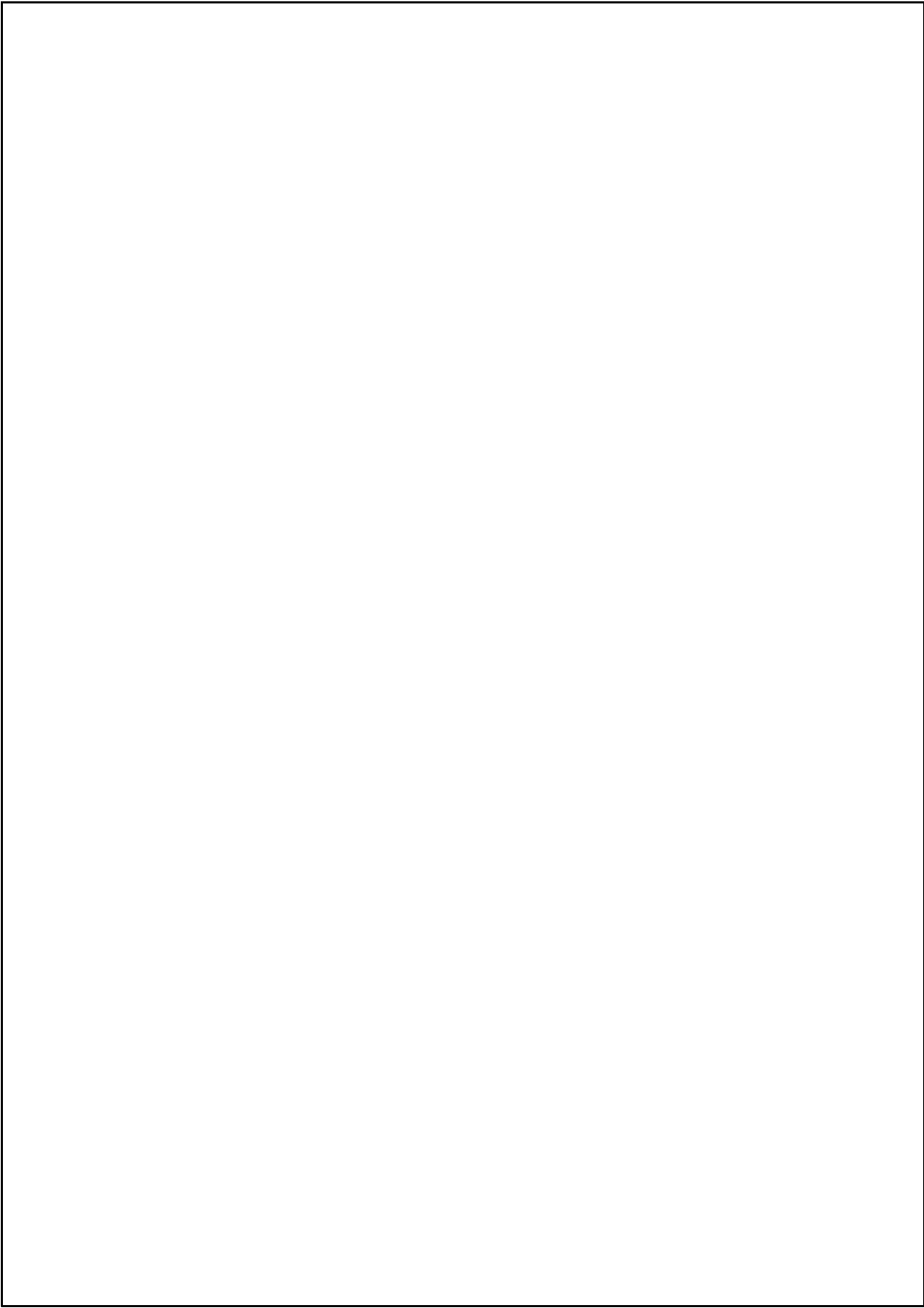
WEB SERVICES (ACCA7005)



REPORT

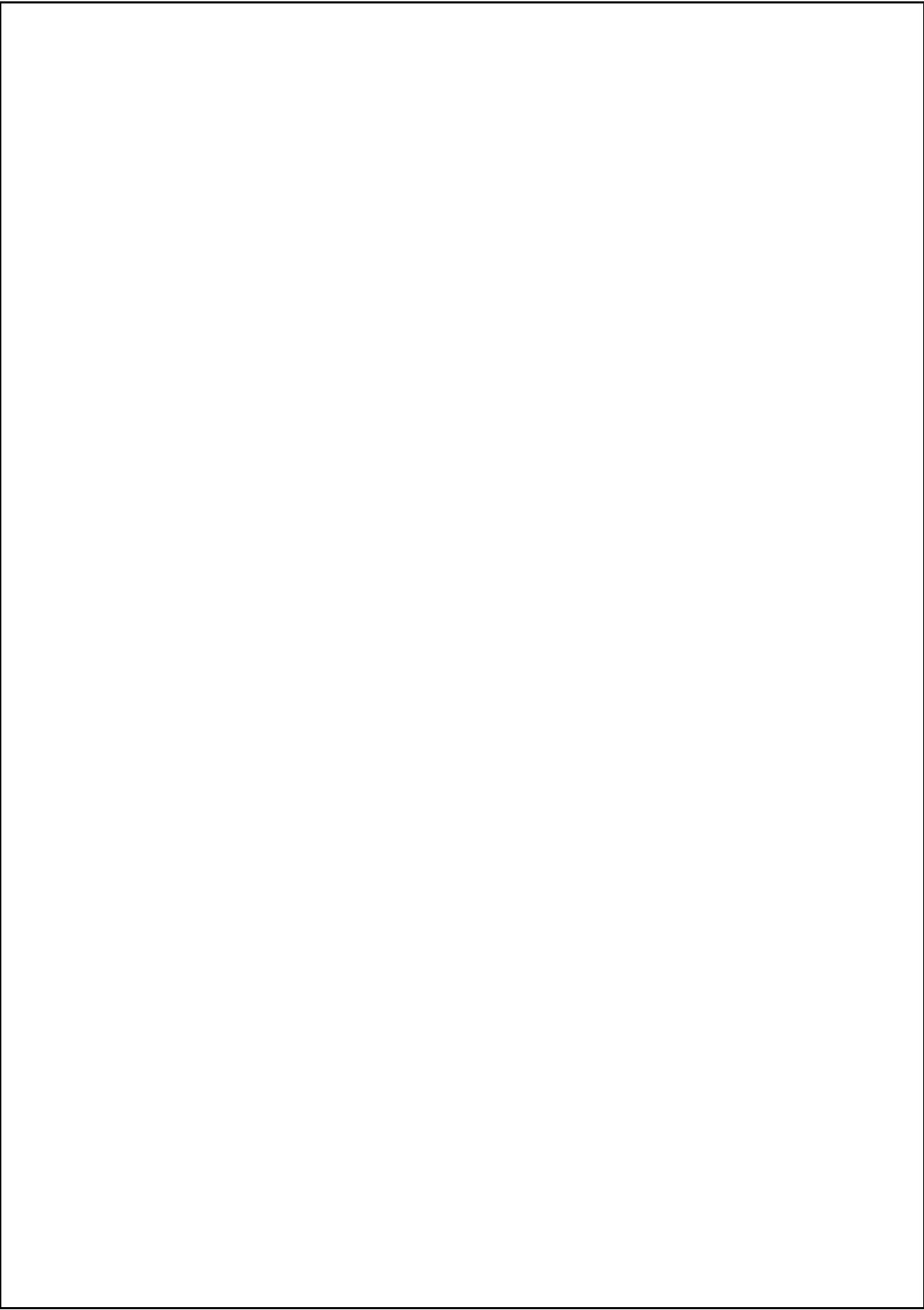
NAME: WENHAO KANG

STUDENT NUMBER: 1904635



Content

ABSTRACT.....	1
1.RESTful best practices.....	2
1.1The RESTful API features	2
1.2 Principles of best practices.....	3
1.3 KANG's store RESTful API practices	4
2.Serverless and web hosting	19
2.1 The definition of a Serverless	19
2.2 Relevance.....	20
3.Circuit breaker pattern	22
3.1 Breaker condition.....	22
3.2 Advantages.....	23
Reference	24



ABSTRACT

This article is divided into three parts. In the first part, it analyzes the characteristics of RESTful architecture style and the standards for implementing RESTful best practices. It also explains how to implement restful best practices in my project in detail, and lists a lot of examples to prove it. The second part explains the definition of serverless technology and its application in web development, and analyzes the correlation between serverless technology and web hosting. The third part explains the meaning of the circuit breaker and the realization principle of the circuit breaker, and proves the importance of the circuit breaker pattern in web development.

Keywords: RESTful, Serverless, Circuit breaker

1.RESTful best practices

1.1The RESTful API features

1.1.1 Source

Based on "resources", data and services are all resources in restful design. The so-called "resource" is an entity on the network, or a specific information on the network. It can be a piece of text, a picture, a song, a service, in short, it is a concrete reality. You can point to it with a URI (uniform resource locator), and each resource corresponds to a specific URI. To get this resource, you can access its URI, so the URI becomes the address or unique identifier of each resource. The so-called "online" is to interact with a series of "resources" on the Internet and call its URI[1].

1.1.2 Stateless

Visiting a website represents an interactive process between client and server. In this process, it is bound to involve changes in data and state. The so-called stateless, that is, all resources can be located by URI, and this location has nothing to do with other resources, and will not change due to the change of other resources. Generally speaking, a call will return a result, and there is no situation similar to "open link - access data - close connection", which depends on the last call[1].

1.1.3 URI

"Resource" is a kind of information entity, which can have many external forms. The concrete form of "resources" is called its "expression layer". A URI represents only the entity of a resource, not its form. Strictly speaking, it is unnecessary to use the ". HTML" suffix at the end of some web addresses, because the suffix represents the format and belongs to the category of "presentation layer", while the URI should only represent the location of "resource". Its specific representation should be specified in the HTTP request header information with the accept and content type fields, which are the description of the "presentation layer". Generally,

each resource has at least one URI corresponding to it, and the most typical URI is the URL.

1.1.4 Unified interface

The restful architecture style specifies the meta operation of data, namely CRUD (create, read, update and delete, That is to say, data addition, deletion, query and modification) operations correspond to HTTP methods respectively: get is used to obtain resources, post is used to create new resources (or update resources), put is used to update resources, and delete is used to delete resources. Thus, the interface of data operation is unified. Only through HTTP method, all data addition, deletion, query and modification can be completed.

That is:

□ 1.GET (SELECT) : Extract a resource (one or more items) from the server. □

2.POST (CREATE) : Create a new resource on the server.

□ 3.PUT (UPDATE) : Update the resource on the server (the client provides complete resource data).

□ 4.PATCH (UPDATE) : Update the resource on the server (the client provides the resource data that needs to be modified).

□ 5.DELETE (DELETE) : Delete the resource from the server.

1.2 Principles of best practices

1.2.1 URL naming specification

API naming should be done in a conventional way to keep it simple and clear. In the restful architecture, each URL represents a resource, so there can be no verbs in the URL, only nouns, and the plural should also be used in the nouns. The implementer should use the corresponding HTTP verbs get, post, put, patch, delete and head to operate these resources[1].

1.2.2 Return data in the same format

For legitimate requests, uniform return of JSON data

1.Code —— contains an integer type HTTP response status code.

2.Status——Contains text: "success", "fail" or "error.". HTTP status response code is "fail" between 500-599, "error" between 400-499, "success" for others (for example, response status codes are 1XX, 2XX and 3xx). This is a choice based on the actual situation.

3.Message——Valid when the status values are "fail" and "error" to display error messages. Referring to the International (i18n) standard, it can contain information numbers or codes, only one of them, or both and separated by separators.

4.Data——The body containing the response. When the status value is "fail" or "error", data only contains the error reason or exception name, or null.

1.2.3 The HTTP status code

The HTTP status code itself has enough meaning:

1.200 means the request was successful

2.500 indicates that the request failed

3.401 means user authentication failed

According to the HTTP status code, you can know whether the deletion, addition, modification, and so on were successful.

1.3 KANG's store RESTful API practices

The following examples show how best practices can be implemented in this project.

1.3.1 User

1. Query the User list

Get/mall/users

Implementation Notes: Query the User list

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
-----------	-------	-------------	----------------	-----------

Parameter	Value	Description	Parameter Type	Data Type
pageIndex	0	pageIndex	query	integer
pageSize	15	pageSize	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

Response Example Value

```
{
  "data": [{id: 1, username: "xx", password: "xx", name: "xx", email:
"xx@xx.com",...},...],
  "message": "Success",
  "state": 200
}
```

2. Add user

```
POST /mall/user
```

Implementation Notes: Add user

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
username		username	query	string
password		password	query	string
name		name	query	string
phone		phone	query	string
email		email	query	string
addr		addr	query	string

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

3. Edit user

PUT /mall/user

Implementation Notes: Edit User information

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer
username		username	query	string
password		password	query	string
name		name	query	string
phone		phone	query	string
email		email	query	string
addr		addr	query	string

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

4. Delete user

DELETE /mall/user

Implementation Notes: Delete user by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

1.3.2 Classification

1. Query classification

GET /mall/classifications

Implementation Notes : Query all classification

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
type		type	query	integer
pageIndex		pageIndex	query	integer
pageSize		pageSize	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

Response Example Value

```
{
  "data": [{id: 1, parentId: 0, cname: "clothing", type: 1}, {id: 2, parentId: 0,
cname: "computer", type: 1},...],
  "message": "Success",
  "state": 200
}
```

2. Add classification

POST /mall/classification

Implementation Notes: New level 1 or level 2 categories

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
cname		cname	query	string
parentId		parentId	query	integer

Parameter	Value	Description	Parameter Type	Data Type
type		type	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

3. Modify classification

PUT /mall/classification

Implementation Notes: Modify Classification by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer
cname		cname	query	string
parentId		parentId	query	integer
type		type	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

4. Delete classification

DELETE /mall/classification

Implementation Notes: Delete classification by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

1.3.3 Product

1. Search all items

GET /mall/products

Implementation Notes: Search all items information

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
pageIndex		pageIndex	query	integer
pageSize		pageSize	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

Response Example Value

```
{  
  "data": [{id: 1, title: "xx", shopPrice: 100, image: "xx.jpg", desc: "xxx",...},...],  
  "message": "Success",  
  "state": 200
```

```
}
```

2. Query specified item

```
GET /mall/product
```

Implementation Notes: Query item by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

Response Example Value

```
{  
  "data": {id: 1, title: "xx", shopPrice: 100, image: "xx.jpg", desc: "xxx",...},  
  "message": "Success",  
  "state": 200  
}
```

3. Add products

POST /mall/product

Implementation Notes: Add products

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
image		image	formData	file
title		title	query	string
marketPrice		marketPrice	query	double
shopPrice		shopPrice	query	double
isHot		isHot	query	integer
desc		desc	query	string
csid		csid	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

4. Update specified product

PUT /mall/product

Implementation Notes: Update product by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer
title		title	query	string
marketPrice		marketPrice	query	double
shopPrice		shopPrice	query	double
desc		desc	query	string
csid		csid	query	integer
isHot		isHot	query	integer
image		image	formData	file

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

5. Delete product

DELETE /mall/product

Implementation Notes: Delete product by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
id		id	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

1.3.4 Cart

1. View cart

GET /mall/cart

Implementation Notes: View shopping cart information

As the current user is directly determined by session, there is no request parameter.

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

Response Example Value

```
{
  "data": [
    {
      "count": 0,
      "id": 0,
      "orderId": 0,
      "product": {
        "categorySec": {
          "cname": "string",
          "id": 0,
          "parentId": 0,
          "type": 0
        },
        "csid": 0,
        "desc": "string",
        "id": 0,
        "image": "string",
        "isHot": 0,
        "marketPrice": 0,

```

```

        "pdate": "2020-05-06T16:26:54.089Z",
        "shopPrice": 0,
        "title": "string"
    },
    "productId": 0,
    "subTotal": 0
}
],
"message": "string",
"state": 200}

```

2. Add to Cart

POST /mall/cart/product

Implementation Notes: Add items to the cart based on the id or modify the item information in the cart based on the id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
productId		productId	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }

HTTP Status Code	Reason	Response Model
500	Fail	{ "data": {}, "message": "string", "state": 500 }

3. Delete product

DELETE /mall/cart/product

Implementation Notes: Delete product by Id

Request Parameters

Parameter	Value	Description	Parameter Type	Data Type
productId		productId	query	integer

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	{ "data": {}, "message": "string", "state": 200 }
401	Unauthorized	{ "data": {}, "message": "string", "state": 401 }
500	Fail	{ "data": {}, "message": "string", "state": 500 }

2.Serverless and web hosting

2.1 The definition of a Serverless

Serverless means no server, but what it really means is that developers don't think too much about the server, but it doesn't mean completely eliminating the server and relying on other resources on the server back end[2]. For example, using Amazon Web Services(AWS) Lambda computing Services to execute code, the Serverless architecture is divided into backend as a Service(BaaS) and Functions as a Service(FaaS)[3]. This architecture eliminates most of the need for traditional always online servers. Serverless is the server side logic implemented by the developer that runs in a stateless computing container. It is triggered by an event and is completely managed by a third party.

BaaS stands for back-end as a service, and its application architecture consists of a large number of third-party cloud servers and APIs, so that the logic and state of the server in the application are managed by the service provider. For example, in our typical single-page application SPA and mobile APP rich client application, the front-end and back-end interactions are mainly based on REST API calls. All you need to do is call the service provider's API to perform functions such as common authentication, cloud data/file storage, message push, application data analysis, etc.

FaaS can be called: function as a service. The developer can deploy the service business logic code directly and run it in a stateless computing container provided by a third party. The developer only needs to write the business code without paying attention to the server, and the execution of the code is triggered by events. Among them, AWS Lambda is one of the best FaaS implementations at present.

The application architecture of Serverless is an application that combines BaaS and FaaS. Users only need to pay attention to the application's business logic code, write functions to run it on the FaaS platform for granularity, and integrate it with BaaS third-party services, and finally build a complete system. There is no need to focus on the server during the entire system process. There is no need to focus on the server during the entire system process.

2.2 Relevance

In the old days when we were working with a container or a virtual machine, we would upload code to a container or a virtual machine, and we would start a process, and the code would run, and it would accept external requests, and it would respond in real time.

Serverless is different from the original container or virtual machine. It implements the computing hosting service. The first thing the Serverless user needs to do is to submit the code we call the cloud function to the platform for code hosting and then configure the trigger.

Why do you need to configure triggers? Because cloud functions are triggered, the code will actually run when triggered. So configuring the trigger means that we set it up with a trigger source, which defines the event at which the code actually runs.

After the user code is hosted to the platform, it is just a code file and configuration store before the event, and the code is not running.

Under what conditions? It is when the event trigger actually arrives that the cloud function actually starts an instance, which means a cell. Once the cell is pulled up, the event is passed to the cell for processing.

If this trigger source has many events and high concurrency, the platform will automatically pull up multiple instances of the same code and configuration for concurrent processing based on the accumulation of events, or the speed at which the events arrive. So you can see that Serverless is running as required, meaning that the code will only be pulled up and running if an event comes along.

In addition, PaaS means platform as service, which is the abstraction of the entire application, and FaaS is the abstraction of the unit (function) of the application logic. For example, PaaS does not pull up a service per request, it is a resident service, whereas FaaS pulls up a container per request for execution, which is a more fine-grained split[3].

In other words, PaaS hosts the entire application, while FaaS hosts some fragmented logical code of the application, which is more advantageous in terms of resource utilization and cost.

3.Circuit breaker pattern

It is very common for one software system to call another remote system in a distributed environment, especially in a distributed system with microservice architecture. The caller of this remote call may be another process or another host across the network, and the biggest difference between this remote call and the process's internal call is that the remote call may fail or hang without any response until the timeout. Worse, if there are multiple callers calling the same suspended service, it is likely that a service's timeout wait will quickly spread throughout the distributed system, causing a chain reaction that consumes a large amount of the distributed system's resources. Eventually, the system could fail.

The Circuit Breaker pattern is designed to prevent such a cascade of disasters in a distributed system.

Circuit breaker is similar to the circuit fuse, implementation approach is very simple, and can use remote services need to be protected in dia encapsulated, internally to monitor number of failure, failure to reach a certain threshold, all subsequent calls to the service, after the circuit breaker contact directly return an error to the calling party, and the services will not continue to call has a problem, so as to achieve the purpose of protecting the caller, also won't appear because of the whole system timeout waterfall chain reaction.

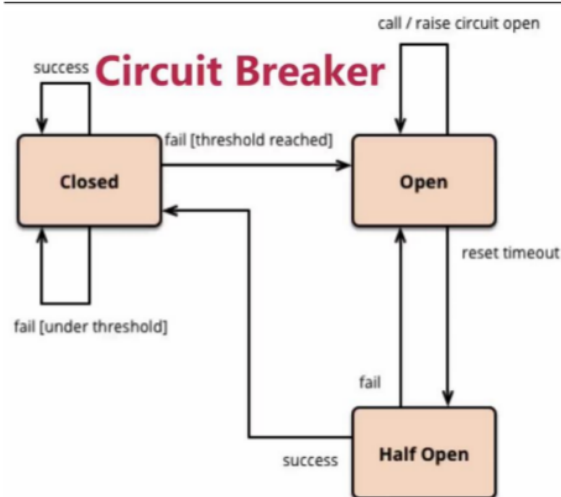
3.1 Breaker condition

The diagram below is the schematic diagram of the circuit breaker. There are three states closed, open and half open

1.Closed is the closed state, that is, the service request is successful. When the request fails for abnormal times, the circuit breaker will be triggered and the open mode will be entered.

2.Open is the open state, which returns an error directly to the service, but involves a clock option, when the default clock reaches this time, it enters the half open half-fuse state.

3. Half open: if the invocations are successful or a percentage of the invocations are successful, the service is considered restored and closed. Otherwise, the service is considered unsuccessful and returns to the open state.



3.2 Advantages

Circuit breakers are valuable for service monitoring. Monitor, record and recover any circuit breaker status changes to ensure service availability. Changes in circuit breaker status can help you add logic to a fault-tolerant system. For example, if the product catalog service is not available (the disconnect is turned on), the UI can retrieve the list of products from the cache. The breaker mode gracefully handles downtime and speed for critical services and helps these services recover by reducing load.

Reference

- [1] Richardson, L., & Ruby, S. (2008). *RESTful web services*. " O'Reilly Media, Inc."
- [2] Anderson, T. E., Dahlin, M. D., Neeffe, J. M., Patterson, D. A., Roselli, D. S., & Wang, R. Y. (1996). Serverless network file systems. *ACM Transactions on Computer Systems (TOCS)*, 14(1), 41-79.
- [3] Fox, G. C., Ishakian, V., Muthusamy, V., & Slominski, A. (2017). Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*.
- [4] Kezunovic, M., Ren, Z., Latisko, G., Sevcik, D. R., Lucey, J. S., Cook, W. E., & Koch, E. A. (2005). Automated monitoring and analysis of circuit breaker operation. *IEEE Transactions on Power Delivery*, 20(3), 1910-1918.

Report

GRADEMARK REPORT

FINAL GRADE

80/100

GENERAL COMMENTS

Instructor

Implementation (50): 44
REST (25): 17

Serverless (25): 19

Feedback

You should have provided some discussion on how REST works, benefits and drawback. You provide some discssion of features and code below it, but you should have used the code to illustrate these features more clearly.

You could have expanded on the benefits and drawbacks of servierless FaaS.

Nice webservice and a good client using a number of ajax operations.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29
