**优必选-RESTful接口定义标准**

**编制部门：软件部**

**编 制 人：贾亮亮**

**审 核：**

**会 签：**

**批 准：**

|  |  |  |  |
| --- | --- | --- | --- |
| **文件修订履行** | | | |
| 版本 | 修订内容简述 | 生效日期 | 修订部门/修订者 |
| 1 | 新制订 | 2017-08-17 | 软件部 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**说明**

1. 本文对整个RESTful接口设计规范进行介绍
2. 对RESTful进行接口定义标准说明

[1. Restful API 的设计规范 5](#_Toc16825)

[2. URI 6](#_Toc13597)

[2.1. URI规范 6](#_Toc3733)

[2.2. 资源集合 vs 单个资源 6](#_Toc23155)

[2.3. 避免层级过深的URI 6](#_Toc21621)

[2.4. 对Composite资源的访问 7](#_Toc23118)

[3. Request 8](#_Toc1817)

[3.1. HTTP方法 8](#_Toc14307)

[3.2. 安全性和幂等性 9](#_Toc26995)

[3.3. 复杂查询 9](#_Toc3244)

[3.4. Bookmarker 10](#_Toc12341)

[3.5. Body Format 10](#_Toc16167)

[3.6. httpClient4 实现文件上传 11](#_Toc541)

[3.7. Content Negotiation 12](#_Toc5259)

[3.8. URL encode编码 12](#_Toc2883)

[4. Response 13](#_Toc7492)

[4.1. 分页response 13](#_Toc17460)

[5. 状态码处理 14](#_Toc14902)

[5.1. 正常状态码处理 14](#_Toc21164)

[5.2. 错误状态码处理 14](#_Toc10508)

[6. 服务型资源 17](#_Toc22806)

[7. 异步任务 18](#_Toc14376)

[8. API的演进 20](#_Toc29520)

[8.1. 版本 20](#_Toc1775)

[8.2. URI失效 20](#_Toc21053)

[9. 安全 21](#_Toc7551)

[9.1. Https协议支持 21](#_Toc13608)

[9.2. Client调用认证 22](#_Toc16148)

[10. 接口定义标准 24](#_Toc14089)

[10.1. 简要说明 24](#_Toc13485)

[10.2. 规范说明 24](#_Toc22313)

[10.3. 文档实例 26](#_Toc31145)

[10.4. 接口版本号 27](#_Toc8127)

[10.5. Token安全访问机制 28](#_Toc17009)

[10.6. Token刷新机制 29](#_Toc8812)

[10.7. 初始化信息记录 31](#_Toc21879)

[10.8. 用户信息保存 33](#_Toc465)

[10.9. HTTP返回码说明 35](#_Toc12550)

[11. 参考文档 42](#_Toc14477)

# Restful API 的设计规范

RESTful架构，就是目前最流行的一种互联网软件架构。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。

接口风格采用RESTful 接口风格，在 REST 样式的 Web 服务中，每个资源都有一个地址。资源本身都是方法调用的目标，方法列表对所有资源都是一样的。这些方法都是标准方法，包括 HTTP、GET、POST、PUT、DELETE。

# URI

URI 表示资源，资源一般对应服务器端领域模型中的实体类。

## URI规范

（1）不用大写；

（2）用中杠 - 不用下杠 \_ ；

（3）参数列表要**encode**；

（4）URI中的名词表示资源集合，使用复数形式。

## 资源集合 vs 单个资源

URI表示资源的两种方式：资源集合、单个资源。

资源集合：

/zoos //所有动物园

/zoos/1/animals //id为1的动物园中的所有动物

单个资源：

/zoos/1 //id为1的动物园

/zoos/1;2;3 //id为1，2，3的动物园

## 避免层级过深的URI

/ 在url中表达层级，用于 按实体关联关系进行对象导航 ，一般根据id导航，过深的导航容易导致url膨胀，不易维护。

如 GET /zoos/1/areas/3/animals/4 ，尽量使用查询参数代替路径中的实体导航，如 GET /animals?zoo=1&area=3 ；

## 对Composite资源的访问

服务器端的组合实体必须在uri中通过父实体的id导航访问， 组合实体不是first-class的实体，它的生命周期完全依赖父实体，无法独立存在，在实现上通常是对数据库表中某些列的抽象，不直接对应表，也无id。

一个常见的例子是 User — Address，Address是对User表中zipCode/country/city三个字段的简单抽象，无法独立于User存在。必须通过User索引到Address： GET /user/1/addresses

# Request

## HTTP方法

通过标准HTTP方法对资源CRUD：

GET：查询

GET /zoos

GET /zoos/1

GET /zoos/1/employees

POST：创建单个资源。 POST一般向“资源集合”型uri发起

POST /animals //新增动物

POST /zoos/1/employees //为id为1的动物园雇佣员工

PUT：更新单个资源（全量），客户端提供完整的更新后的资源。与之对应的是

PATCH，PATCH 负责部分更新，客户端提供要更新的那些字段，PUT/PATCH一般向“单个资源”型uri发起

PUT /animals/1

PUT /zoos/1

DELETE：删除

DELETE /zoos/1/employees/2

DELETE /zoos/1/employees/2;4;5

DELETE /zoos/1/animals //删除id为1的动物园内的所有动物

HEAD / OPTION 用的不多，就不多解释了。

## 安全性和幂等性

**安全性 ：**不会改变资源状态，可以理解为只读的；

**幂等性 ：**执行1次和执行N次，对资源状态改变的效果是等价的。

| . | 安全性 | 幂等性 |
| --- | --- | --- |
| GET | √ | √ |
| POST | × | × |
| PUT | × | √ |
| DELETE | × | √ |

安全性和幂等性均不保证反复请求能拿到相同的response。以 DELETE 为例，第一次DELETE返回200表示删除成功，第二次返回404提示资源不存在，这是允许的。

## 复杂查询

查询可以捎带以下参数：

| . | 示例 | 备注 |
| --- | --- | --- |
| 过滤条件 | ?type=1&age=16 | 允许一定的uri冗余，如 /zoos/1 与 /zoos?id=1 |
| 排序 | ?sort=age,desc |  |
| 投影 | ?whitelist=id,name,email |  |
| 分页 | ?limit=10&offset=3 |  |

## Bookmarker

经常使用的、复杂的查询标签化，降低维护成本。

如：

GET /trades?status=closed&sort=created,desc

快捷方式：

GET /trades#recently-closed

或者

GET /trades/recently-closed

## Body Format

只用以下常见的3种body format：

**Json格式请求**

Content-Type: application/json

POST /v1/animal HTTP/1.1

Host: api.example.org

Accept: application/json

Content-Type: application/json

Content-Length: 24

{

"name": "Gir",

"animalType": "12"

}

**表单提交：**

Content-Type: application/x-www-form-urlencoded

(浏览器POST表单用的格式)

POST /login

HTTP/1.1Host: example.com

Content-Length: 31

Accept: text/html

Content-Type: application/x-www-form-urlencoded

username=root&password=Zion0101

**文件上传：**

Content-Type: multipart/form-data; boundary=—-RANDOM\_jDMUxq4Ot5

(表单有文件上传时的格式)

POST /upload

HTTP/1.1Host: file.example.org

Accept-Language: zh-cn,zh;q=0.5

Accept-Charset: GBK,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Content-Length: 60408

Content-Type: multipart/form-data; boundary=—-RANDOM\_jDMUxq4Ot5

## httpClient4 实现文件上传

httpClient4使用http-mime.jar包的MultipartEntity实现，代码如下（为了简洁，处理了异常处理代码）：

HttpPost httpPost = newHttpPost(url);

Log.debug("post url:"+url);

httpPost.setHeader("User-Agent","SOHUWapRebot");

httpPost.setHeader("Accept-Language","zh-cn,zh;q=0.5");

httpPost.setHeader("Accept-Charset","GBK,utf-8;q=0.7,\*;q=0.7");

httpPost.setHeader("Connection","keep-alive");

MultipartEntity mutiEntity = newMultipartEntity();

File file = new File("d:/photo.jpg");

mutiEntity.addPart("desc",

new StringBody("美丽的西双版纳", Charset.forName("utf-8")));

mutiEntity.addPart("pic", newFileBody(file));

httpPost.setEntity(mutiEntity);

HttpResponse  httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity =  httpResponse.getEntity();

String content = EntityUtils.toString(httpEntity);

## Content Negotiation

资源可以有多种表示方式，如json、xml、pdf、excel等等，客户端可以指定自己期望的格式，通常有两种方式：

Http header Accept ：

Accept:application/xml;q=0.6,application/atom+xml;q=1.0

q为各项格式的偏好程度

url中加文件后缀： /zoo/1.json

## URL encode编码

对URL请求时，参数要进行编码，进行URL encode主要是为了解决一下问题：

1. URL有规范，在参数中出现参数&会截断参数。
2. URL中文问题，编码可以转换成英文，防止出现中文乱码。
3. 也是第一种问题的扩展，防止URL中有个参数值也是URL时，传输时会出现错误。

# Response

包装：response 的 body 直接就是数据，将数据放入到data中，示例如下：

{

"success":true,

"msg":"success",

"status":10200

"data":{"id":1,"name":"xiaotuan"},

}

各HTTP方法成功处理后的数据格式：

| · | response 格式 |
| --- | --- |
| GET | 单个对象、集合 |
| POST | 新增成功的对象 |
| PUT/PATCH | 更新成功的对象 |
| DELETE | 空 |

json格式的约定：

时间用长整形(毫秒数)，客户端自己按需解析（ [moment.js](http://mementjs.com/" \t "http://blog.csdn.net/youhan26/article/details/_blank) ），不传 null 字段。

## 分页response

{

"paging":{"limit":10,"offset":0,"total":729},

"data":[{},{},{}...],

"status":10200,

"msg":"success",

"success":true

}

# 状态码处理

## 正常状态码处理

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

| 状态码 | 使用场景 |
| --- | --- |
| 200 OK | [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。 |
| 201 CREATED | [POST/PUT/PATCH]：用户新建或修改数据成功。 |
| 202 Accepted | [\*]：表示一个请求已经进入后台排队（异步任务） |
| 204 NO CONTENT | [DELETE]：用户删除数据成功。 |

## 错误状态码处理

不要发生了错误但给2xx响应，客户端可能会缓存成功的https请求；

正确设置https状态码，不要自定义；

Response body 提供

1. 错误的代码（日志/问题追查）；
2. 错误的描述文本（展示给用户）。

对第三点的实现稍微多说一点：

Java 服务器端一般用异常表示 RESTful API 的错误。API 可能抛出两类异常：业务异常和非业务异常。

业务异常 由自己的业务代码抛出，表示一个用例的前置条件不满足、业务规则冲突等，比如参数校验不通过、权限校验失败。

非业务类异常 表示不在预期内的问题，通常由类库、框架抛出，或由于自己的代码逻辑错误导致，比如数据库连接失败、空指针异常、除0错误等等。

业务类异常必须提供2种信息：

如果抛出该类异常，HTTPS 响应状态码应该设成什么；

异常的文本描述；

在Controller层使用统一的异常拦截器：

设置 HTTP 响应状态码：对业务类异常，用它指定的 HTTPS code；对非业务类异常，统一500；

Response Body 的错误码：异常类名

Response Body 的错误描述：对业务类异常，用它指定的错误文本；对非业务类异常，线上可以统一文案如“服务器端错误，请稍后再试”，开发或测试环境中用异常的 stacktrace，服务器端提供该行为的开关。

常用的https状态码及使用场景：

| 状态码 | 使用场景 |
| --- | --- |
| 400 INVALID REQUEST | [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。 |
| 401 Unauthorized | [\*]：未经验证的用户，常见于未登录。如果经过验证后依然没权限，应该 403（即 authentication 和 authorization 的区别）。 |
| 403 forbidden | [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。 |
| 404 not found | [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。 |
| 406 Not Acceptable | [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。 |
| 410 Gone | [GET]：用户请求的资源被永久删除，且不会再得到的。 |
| 422 Unprocesable entity | [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。 |
| 500 internal server error | [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。 |
| 503 service unavaliable | [\*]：由容器抛出，自己的代码不要抛这个异常 |

# 服务型资源

除了资源简单的CRUD，服务器端经常还会提供其他服务，这些服务无法直接用上面提到的URI映射。如：

按关键字搜索；

1)计算地球上两点间的距离；

2)批量向用户推送消息

可以把这些服务看成资源，计算的结果是资源的presentation，按服务属性选择合适的HTTP方法。

例：

GET /search?q=filter&category=file 搜索

GET /distance-calc?lats=47.480&lngs=-122.389&late=37.108&lnge=-122.448

POST /batch-publish-msg

[

{"from":0,"to":1,"text":"abc"},

{},

{}...

]

# 异步任务

对耗时的异步任务，服务器端接受客户端传递的参数后，应返回创建成功的任务资源，其中包含了任务的执行状态。客户端可以轮训该任务获得最新的执行进度。

提交任务：

POST /batch-publish-msg

[{"from":0,"to":1,"text":"abc"},{},{}...]

返回：

{ "data":{

"taskId":3,

"createBy":"Anonymous",

"status":"running"

},

"success":true,

"msg":"success",

"status":10200

}

GET /task/3

{ "data":{

"taskId":3,

"createBy":"Anonymous",

"status":"success"

},

"success":true,

"msg":"success",

"status":11200

}

如果任务的执行状态包括较多信息，可以把“执行状态”抽象成 组合资源 ，客户端查询该状态资源了解任务的执行情况。

提交任务：

POST /batch-publish-msg

[{"from":0,"to":1,"text":"abc"},{},{}...]

返回：

{

"data":{

"taskId":3,

"createBy":"Anonymous"

},

"msg": "OK",

"status": 10200,

"success": true

}

GET /task/3/status

{

"data":{

"progress":"50%",

"total":18,

"success":8,

"fail":1

}

"msg":"OK",

"status": 10200,

"success":true

}

# API的演进

## 版本

常见的三种方式：

在uri中放版本信息： GET /users/1/v1

Accept Header： Accept: application/json+v1

自定义 Header： X-Api-Version: 1

用第一种，接口调用的版本信息在最后面，方便编码实现，虽然没有那么优雅，但最明显最方便。

## URI失效

随着系统发展，总有一些API失效或者迁移，对失效的API，返回404 not found 或 410 gone；对迁移的API，返回 301 重定向。

# 安全

## Https协议支持

HTTPS以保密为目标研发，简单讲是HTTP的安全版。其安全基础是SSL协议，因此加密的详细内容请看SSL。全称Hypertext Transfer Protocol over Secure Socket Layer。  
      它是一个URI scheme，句法类同http:体系。它使用了HTTP，但HTTPS存在不同于HTTP的默认端口及一个加密／身份验证层（在HTTP与TCP之间）。这个协议的最初研发由网景公司进行，提供了身份验证与加密通讯方法，现在它被广泛用于互联网上安全敏感的通讯，例如交易支付方面。  
      SSL极难窃听，对中间人攻击提供一定的合理保护。严格学术表述HTTPS是两个协议的结合，即传输层SS＋应用层HTTP。  
 HTTPS默认使用TCP端口443（HTTP默认则是TCP端口80），也可以指定其他TCP端口。  
要使协议正常运作，至少服务器必需有PKI证书，而客户端则不一定。  
它的加密强度依赖软件的正确实现，以及服务器客户端双方加密[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/lxqluo/article/details/_blank)的支持。  
   
即便HTTPS被正确实现，仍有以下人为因素：  
**冒充网站**   
**钓鱼攻击**   
        制造与原网站相似的假冒网址，并诱导客户访问，常见例子是仿制银行网站。   
**中间人攻击**       在通讯线路中途篡改证书，从而充当网站客户双方的中间人，这样可知道全部通讯内容。检查证书才有可能发现中间人的存在。   
**冒充客户**  
       由于证书费用昂贵，通常只有网站服务器拥有证书。往往客户身份得不到验证。   
        在TLS 1.1之前SSL证书仅能对应IP，使得HTTPS无法在虚拟主机（仅有域名）上正常运作。现在的TLS 1.1早已完全支持基于域名的虚拟主机。  
   
**HTTPS和HTTP的区别：   
https协议**需要到ca申请证书，一般免费证书很少，需要交费。   
**http是超文本传输协议**，信息是明文传输，https 则是具有安全性的ssl加密传输协议   
**http和https使用**的是完全不同的连接方式用的端口也不一样,前者是80,后者是443。   
**http的连接很简单**,是无状态的   
**HTTPS协议是由SSL+HTTP**协议构建的可进行加密传输、身份认证的网络协议 要比http协议安全

## Client调用认证

通过用户自定义的协议生成Token，获得Token接口的调用说明说下：

接口说明：根据用户名密码，获得Token信息

1. 接口说明：根据用户名和密码获得Token信息

* 接口地址：

GET https://api.ubtechinc.com/openapi/token/v1

* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 是否必须 | 参数类型 | 备注 |
| name | 是 | String | 用户名 |
| password | 是 | String | 密码 |
| client\_id | 是 | String | 表示客户端的ID，必选项 |
| client\_secret | 是 | String | 表示客户端的Secret，必选项 |
| grant\_type | 是 | String | 表示授权类型，此处的值固定为"password"，必选项。 |

* 返回信息：

{

[ {"access\_token":"78b6654d01034997842f7e37eb93189712"} ]

}

Curl调用方式例子如下：

curl -X GET --header 'Accept: application/json'

'https://api.ubtechinc.com/openapi/token/v1?name=18820602060&password=123456'

客户端通过Client每次访问接口时，需要在header 中添加该参数，对用户的身份进行认证，如下：

**authorization**:”**78b6654d01034997842f7e37eb93189712**”

Curl调用方式例子如下：

curl -X GET --header 'Accept: application/json' --header 'authorization: 78b6654d01034997842f7e37eb93189712' 'https://api.ubtechinc.com/task/daily/v1?language=en'

# 接口定义标准

## 简要说明

接口风格采用RESTful 接口风格。

在 REST 样式的 Web 服务中，每个资源都有一个地址。资源本身都是方法调用的目标，方法列表对所有资源都是一样的。这些方法都是标准方法，包括 HTTP GET、POST、PUT、DELETE

## 规范说明

一．基础url

单资源：

/{resource}

~GET—返回一个新的资源（用默认值生成）

~POST—创建一个新资源

单资源带id

/{resource}/{:id}

~GET—返回对应id的资源

~PUT—修改对应id的资源

~DELETE—删除对应id的资源

复数资源

/{resources}

~GET—返回所有资源

复数资源带查找

/{resources}

~GET—返回满足key值为value条件的资源,支持key\_like和key\_in，分别表示sql的 key like ‘%{value}%’和 key in ({value})

/{resources}?pageIndex=1&pageSize=10

~GET—返回分页数据

资源下的子集

/{resource}/{:id}/{item}

~GET—返回一个新的资源（用默认值生成）

~POST—创建一个新资源

其它说明：

{resource}:资源名称

{resources}:资源名称复数形式

{:id}:资源唯一id

{item}:子集资源名称

二．返回值

单资源

data: ｛json格式数据｝

复数资源

{

［｛json格式数据｝,｛json格式数据｝...］

}

复数资源带分页

｛

{

paging:{

totalElements:‘总数’,totalPages:‘总页数’

pageSize:‘每页个数’

pageIndex:‘页下标（0开始）’

sort:‘排序’

last:‘是否最后一页’

first:‘是否第一页’

}

data:[

｛json格式数据｝,｛json格式数据｝,｛json格式数据｝...

]

}

｝

三．高级url

/{domain}/{rest-convention}/{version}

其它说明：

domain：业务域，用于隔离不同权限等作用

rest-convention：实际接口

Version: 版本号

## 文档实例

地址：<https://api.github.com/api/v1/customer/:id>

方法：get

调用方法：

curl -H " authorization:XXXXXXXXXXX" https://api.github.com/api/v1/customer/1

**Header**

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| authorization | String | token |

**参数**

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | Number | ID |

**返回 Success 200**

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| message | String | 消息体 |
| code | String | 状态码 200 |
| succes | Boolean | true |
| data | Object | 返回结果 |
| deleted | Object |  |
| id | Object |  |
| nickname | Object | 昵称 |
| name | Object | 用户名 |
| password | Object |  |
| cate | Object | 用户分类（0：游客，1注册用户） |
| headimgId | Object | 用户头像 |
| mobile | Object | 用户电话 |
| gender | Object | 性别（1男，2女） |
| age | Object | 年龄 |
| birth | Object | 出生年月日 |
| city | Object | 用户所在城市 |
| province | Object | 用户所在国家 |
| country | Object | 用户所在省份 |
| headimgurl | Object | 用户头像 |
| enabled | Object | 是否开启 |
| optId | Object |  |
| optDate | Object |  |
| createId | Object |  |
| createDate | Object |  |

## 接口版本号

使用前缀的方式，用来作为接口的版本信息，例如：

http://localhost:8080/userservice-rest/v2/user/{userId}?param1=1

（以前版本的接口要兼容，新增的接口用v2版本开发，以前的版本默认为v1）

## Token安全访问机制

Token加入签名规则方式，具体签名规则如下：

       在用户可以使用REST接口之前，首先需要通过向REST接口开放方申请，当获准后会收到两个key：accessKey和secretKey。其中 accessKey相当于用户标识，应用会通过它区分不同用户，而secreKey相当于提供给用户的密码，在接口使用过程中都不会在网络中传输，只有用户和应用系统知道。

       用户对接口的每一次请求都会默认通过某种机制生成一个签名，然后发送给REST服务端。服务端收到后，会根据该机制验证该请求的合法性，从而避免非法用户的随意使用。

       一般，签名以如下形式保存在HTTP Header中发送给REST服务器(其中oss为固定字，Signature是摘要内容)：

token = “oss” + “ ”+ accessKey + “:” + Signature

计算的伪代码如下：

StringToSing =HTTP-Verb +”\n”+ //http请求的动作

Content-MD5+”\n”+ //http请求的MD5值

Content-Type+”\n”+ //http请求的类型

Date+”\n”+ //http请求时间

CanonicalizdResource； //http请求资源

其中HTTP-Verb是http请求动作，对应于GET,PUT,DELETE,POST等（不能为空），Content-MD5表示请求内容数据的 MD5值（可以为空，以空字符代替，下同），Content-Type为请求的类型（可以为空），Date为本次操作的时间（不能为 空），CanonicalizdResource表示请求的资源（不能为空），当然我们还可以根据系统本身情况指定某些可选或必选参数。

Signature =BASE64(HMAC-SHA1(UTF-8-Encoding-Of(secretKey, StringToSign)))

采用utf-8提前对参数进行编码，可以减少客户端编码差异带来的影响，因为服务端到时是统一采用utf-8来做的，HMAC-SHA1生成签名摘 要指纹信息是不可逆的，安全性得到提高，最后还进行了BASE64编码的原因是将摘要内容全部转换为可显字符，应对某些不可显字符在网络传输中的丢失。

　　服务端验证方式则是根据传输过来的token解析出accessKey和Singature，根据accessKey得到本地保存的对应 secretKey(注意其并未在网络中传输)，然后再重新根据客户端Signature生成方式重新生成与解析的进行比较，相同则认证成功结束，否则， 直接返回错误信息给客户端。

## Token刷新机制

1. 用户通过用户名和密码进行登录后，服务端返回获得access\_token信息，同时返回access\_token有效期，到期的时间戳。

用户登录的HTTP POST 方法

https://account.ubtrobot.com/authorization-rest/openapi/token/v2?grant\_type=password&client\_id=${client\_id}&client\_secret=${client\_secret}&access\_token=${access\_token}

参数说明：

|  |  |
| --- | --- |
| 参数名称 | 参数说明： |
| username | 表示用户名，必选项。 |
| password | 表示用户名，必选项。 |
| client\_id | 表示客户端的ID，必选项 |
| client\_secret | 表示客户端的Secret，必选项 |
| grant\_type | 表示授权类型，此处的值固定为"password"，必选项。 |

获得access\_token的结果信息如下：

{

{ //返回的Token 信息

"access\_token":"60044cb520f814321c21afafbace0a26",

"expires\_in":1504578365148,

}

}

**2）**用户通过client\_id和client\_secret进行access\_token的刷新操作，通过如下接口重新获取用户的access\_token信息，HTTP POST 方法

https://account.ubtrobot.com/authorization-rest/openapi/token/v2?grant\_type=refresh\_token&client\_id=${client\_id}&client\_secret=${client\_secret}&grant\_type=refresh\_token&access\_token=${access\_token}

参数说明：

|  |  |
| --- | --- |
| 参数名称 | 参数说明： |
| client\_id | 表示客户端的ID，必选项 |
| client\_secret | 表示客户端的Secret，必选项 |
| grant\_type | 表示授权类型，此处的值固定为"refresh\_token"，必选项。 |
| access\_token | 用户登录时获得的access\_token信息。 |

刷新access\_token后，重新生成的token结果信息如下：

{

{ //返回的Token 信息

"access\_token":"12e775ef512f3fcbd84f01fdb25f29a5",

"expires\_in":1504578865148

}

}

返回结果说明：

|  |  |
| --- | --- |
| 参数名称 | 参数说明： |
| access\_token | 表示访问令牌，必选项。 |
| expires\_in | 表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间。 |

## 初始化信息记录

**通用请求头部字段声明**：

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| devType | String | 设备类型 | 区分：  android,ios,robot |
| sysVersionName | String | 系统版本名 | 如： 客户端android系统版本名如 Jelly Bean机器人端android系统版本 KitKat |
| sysVersionCode | Integer | 系统版本号 | 如： android Jelly Bean系统版本名可能对应 16， 17， 18 |
| appName | String | 应用程序名 | 如：客户端的闲聊应用，它的名字是“闲聊”  主服务的应用名是“alpha”或者“cruzr” |
| appId | Integer | 应用程序包名(ID) | 应用的包名，如：闲聊应用， 包名“com.ubtechinc.xianliao” alpha机器人主服务的包名“com.ubtechinc.alpha” |
| appVersionName | String | 应用版本名 | 如：客户端闲聊应用的版本名可能是v1.2.1  主服务的版本名可能是v2.1.0  这个值用来显示的 |
| appVersionCode | String | 应用版本号 | 如：客户端闲聊应用的v1.2.1对应的版本号是3主服务v2.1.0对应的版本号是5这个值用于应用升级，每发一个版本，版本号递增 |
| robotSn | Integer | 机器人序列号 | 机器人设备信息，机器人端必传字段？**待定** |
| sysLanguage | String | 系统语言 | 多国语言支持：en\_us,zh\_cn 等 |
| protocolVersionCode | Integer | 协议版本 | 待定 |

**手机端设备请求头部字段声明:**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| imei | String | 手机imei |  |
| deviceMode | String | 手机设备型号 |  |
| deviceManufactor | String | 手机生产商 |  |
| deviceScreenDpi | Integer | 手机屏幕密度 |  |
| deviceScreenWidth | Integer | 手机屏幕宽度 |  |
| deviceScreenHeight | Integer | 手机屏幕高度 |  |
| channelId | Integer | 渠道ID | 客户端应用可能来自不同的应用市场，通过这个字段记录用户是从哪里下载手机端应用的 |
| macAddress | String | Mac地址 |  |

**机器人端STM设备信息请求头部字段声明:**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| chestSoftwareVersion | String | 胸部软件版本 |  |
| chestHardwareVersion | String | 胸部硬件版本 |  |
| chestBootVersion | String | 胸部boot版本 |  |
| headerSoftwareVersoin | String | 头部软件版本 |  |
| headerHardwareVersion | String | 头部硬件版本 |  |
| headerBootVersion | String | 头部boot版本 |  |

## 用户信息保存

**文件信息请求头部字段声明:**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| fileName | String | 文件名称 |  |
| fileType | String | 文件类型 |  |
| createTime | Date | 创建时间 |  |
| updateTime | Date | 更新时间 |  |
| filePath | String | 文件路径 |  |

**图片信息请求头部字段声明:**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| fileName | String | 名称 |  |
| content | String | 内容简介 |  |
| createTime | Date | 创建时间 |  |
| updateTime | Date | 更新时间 |  |
| filePath | String | 文件路径 |  |
| size | Integer | 文件大小 | 大小需要换算成M |

**视频信息请求头部字段声明:**

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 类型 | 含义 | 备注 |
| fileName | String | 名称 |  |
| content | String | 内容简介 |  |
| createTime | Date | 创建时间 |  |
| updateTime | Date | 更新时间 |  |
| filePath | String | 文件路径 |  |
| size | Integer | 文件大小 | 大小需要换算成M |
| codeRate | String | 码率 |  |
| videoDuration | String | 视频时长 |  |
| videoType | String | 视频格式 |  |

## HTTP返回码说明

| 状态码 | 使用场景 |
| --- | --- |
| 1XX | 信息提示：这些状态代码表示临时的响应。客户端在收到常规响应之前，应准备接收一个或多个 1xx 响应。 |
| 100 | Continue 初始的请求已经接受，客户应当继续发送请求的其余部分。（HTTP 1.1新） |
| 101 | Switching Protocols 服务器将遵从客户的请求转换到另外一种协议（HTTP 1.1新） |
| 2XX | 成功，这类状态代码表明服务器成功地接受了客户端请求。 |
| 200 | OK 一切正常，对GET和POST请求的应答文档跟在后面。 |
| 201 | Created 服务器已经创建了文档，Location头给出了它的URL。 |
| 202 | Accepted 已经接受请求，但处理尚未完成。 |
| 203 | Non-Authoritative Information 文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝，非权威性信息（HTTP 1.1新）。 |
| 204 | No Content 没有新文档，浏览器应该继续显示原来的文档。如果用户定期地刷新页面，而Servlet可以确定用户文档足够新，这个状态代码是很有用的。 |
| 205 | Reset Content 没有新的内容，但浏览器应该重置它所显示的内容。用来强制浏览器清除表单输入内容（HTTP 1.1新）。 |
| 206 | Partial Content 客户发送了一个带有Range头的GET请求，服务器完成了它（HTTP 1.1新）。 |
| 3XX | 重定向，客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。 |
| 300 | Multiple Choices 客户请求的文档可以在多个位置找到，这些位置已经在返回的文档内列出。如果服务器要提出优先选择，则应该在Location应答头指明。 |
| 301 | Moved Permanently 客户请求的文档在其他地方，新的URL在Location头中给出，浏览器应该自动地访问新的URL。 |
| 302 | Found 类似于301，但新的URL应该被视为临时性的替代，而不是永久性的。注意，在HTTP1.0中对应的状态信息是“Moved Temporatily”。出现该状态代码时，浏览器能够自动访问新的URL，因此它是一个很有用的状态代码。注意这个状态代码有时候可以和301替换使用。例如，如果浏览器错误地请求 http://host/~user （缺少了后面的斜杠），有的服务器返回301，有的则返回302。严格地说，我们只能假定只有当原来的请求是GET时浏览器才会自动重定向。请参见 307 |
| 303 | See Other 类似于301/302，不同之处在于，如果原来的请求是POST，Location头指定的重定向目标文档应该通过GET提取（HTTP 1.1新）。 |
| 304 | Not Modified 客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。 |
| 305 | Use Proxy 客户请求的文档应该通过Location头所指明的代理服务器提取（HTTP 1.1新）。 |
| 307 | Temporary Redirect 和302（Found）相同。许多浏览器会错误地响应302应答进行重定向，即使原来的请求是POST，即使它实际上只能在POST请求的应答是303时才能重定向。由于这个原因，HTTP 1.1新增了307，以便更加清除地区分几个状态代码：当出现303应答时，浏览器可以跟随重定向的GET和POST请求；如果是307应答，则浏览器只能跟随对GET请求的重定向。（HTTP 1.1新） |
| 4XX | 客户端错误 |
| 400 | Bad Request 请求出现语法错误。400的返回body里面经常有Invalid argument provided （提供无效的参数）表明header或parameter里面有无效的参数。 |
| 401 | Unauthorized 访问被拒绝，客户试图未经授权访问受密码保护的页面。应答中会包含一个WWW-Authenticate头，浏览器据此显示用户名字/密码对话框，然后在填写合适的Authorization头后再次发出请求。IIS 定义了许多不同的 401 错误，它们指明更为具体的错误原因。这些具体的错误代码在浏览器中显示，但不在 IIS 日志中显示：。 |
| 40101 | 登录失败。 |
| 40102 | 服务器配置导致登录失败。 |
| 40103 | 由于 ACL 对资源的限制而未获得授权。 |
| 40104 | 筛选器授权失败。 |
| 40105 | ISAPI/CGI 应用程序授权失败。 |
| 40107 | 访问被 Web 服务器上的 URL 授权策略拒绝。这个错误代码为 IIS 6.0 所专用。 |
| 403 | Forbidden 资源不可用。服务器理解客户的请求，但拒绝处理它。通常由于服务器上文件或目录的权限设置导致。禁止访问：IIS 定义了许多不同的 403 错误，它们指明更为具体的错误原因： |
| 40301 | 执行访问被禁止。 |
| 40302 | 读访问被禁止。 |
| 40303 | 写访问被禁止。 |
| 40304 | 要求 SSL。 |
| 40305 | 要求 SSL 128。 |
| 40306 | IP 地址被拒绝。 |
| 40307 | 要求客户端证书。 |
| 40308 | 站点访问被拒绝。 |
| 40309 | 用户数过多。 |
| 40310 | 配置无效。 |
| 40311 | 密码更改。 |
| 40312 | 拒绝访问映射表。 |
| 40313 | 客户端证书被吊销。 |
| 40314 | 拒绝目录列表。 |
| 40315 | 超出客户端访问许可。 |
| 40316 | 客户端证书不受信任或无效。 |
| 40317 | 客户端证书已过期或尚未生效。 |
| 40318 | 在当前的应用程序池中不能执行所请求的 URL。这个错误代码为 IIS 6.0 所专用。 |
| 40319 | 不能为这个应用程序池中的客户端执行 CGI。这个错误代码为 IIS 6.0 所专用。 |
| 40320 | Passport 登录失败。这个错误代码为 IIS 6.0 所专用。 |
| 404 | Not Found 无法找到指定位置的资源。这也是一个常用的应答。  404返回的body里面经常有description  ：The requested resource () is not available.很有可能是url出了问题 |
| 40400 | （无） – 没有找到文件或目录。 |
| 40401 | 无法在所请求的端口上访问 Web 站点。 |
| 40402 | Web 服务扩展锁定策略阻止本请求。 |
| 40403 | MIME 映射策略阻止本请求。 |
| 405 | Method Not Allowed 请求方法（GET、POST、HEAD、Delete、PUT、TRACE等）对指定的资源不适用，用来访问本页面的 HTTP 谓词不被允许（方法不被允许）（HTTP 1.1新） |
| 406 | Not Acceptable 指定的资源已经找到，但它的MIME类型和客户在Accpet头中所指定的不兼容，客户端浏览器不接受所请求页面的 MIME 类型（HTTP 1.1新）。 |
| 407 | Proxy Authentication Required 要求进行代理身份验证，类似于401，表示客户必须先经过代理服务器的授权。（HTTP 1.1新） |
| 408 | Request Timeout 在服务器许可的等待时间内，客户一直没有发出任何请求。客户可以在以后重复同一请求。（HTTP 1.1新） |
| 409 | Conflict 通常和PUT请求有关。由于请求和资源的当前状态相冲突，因此请求不能成功。（HTTP 1.1新） |
| 410 | Gone 所请求的文档已经不再可用，而且服务器不知道应该重定向到哪一个地址。它和404的不同在于，返回407表示文档永久地离开了指定的位置，而404表示由于未知的原因文档不可用。（HTTP 1.1新） |
| 411 | Length Required 服务器不能处理请求，除非客户发送一个Content-Length头。（HTTP 1.1新） |
| 412 | Precondition Failed 请求头中指定的一些前提条件失败（HTTP 1.1新）。 |
| 413 | Request Entity Too Large 目标文档的大小超过服务器当前愿意处理的大小。如果服务器认为自己能够稍后再处理该请求，则应该提供一个Retry-After头（HTTP 1.1新）。 |
| 414 | Request URI Too Long URI太长（HTTP 1.1新）。 |
| 415 | 不支持的媒体类型。 |
| 416 | Requested Range Not Satisfiable 服务器不能满足客户在请求中指定的Range头。（HTTP 1.1新） |
| 417 | 执行失败。 |
| 422 | 当创建一个对象时，发生一个验证错误。 |
| 423 | 锁定的错误。 |
| 5XX | 服务器错误 服务器由于遇到错误而不能完成该请求。 |
| 500 | Internal Server Error 服务器遇到了意料不到的情况，不能完成客户的请求。 |
| 501 | Not Implemented 服务器不支持实现请求所需要的功能，页眉值指定了未实现的配置。例如，客户发出了一个服务器不支持的PUT请求。 |
| 502 | Bad Gateway 服务器作为网关或者代理时，为了完成请求访问下一个服务器，但该服务器返回了非法的应答。 亦说Web 服务器用作网关或代理服务器时收到了无效响应。 |
| 50201 | CGI 应用程序超时。 |
| 50202 | CGI 应用程序出错。 |
| 503 | Service Unavailable 服务不可用，服务器由于维护或者负载过重未能应答。例如，Servlet可能在[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://www.cnblogs.com/111testing/p/_blank)连接池已满的情况下返回503。服务器返回503时可以提供一个 Retry-After头。这个错误代码为 IIS 6.0 所专用。 |
| 504 | Gateway Timeout 网关超时，由作为代理或网关的服务器使用，表示不能及时地从远程服务器获得应答。（HTTP 1.1新） 。 |
| 505 | HTTP Version Not Supported 服务器不支持请求中所指明的HTTP版本。（HTTP 1.1新） |

# 参考文档

[1] 阮一峰.RESTful API 设计指南[N] 网络日志. 2014.05

http://www.ruanyifeng.com/blog/2014/05/restful\_api.html

[2] [克鲁斯卡尔.](http://www.tuicool.com/sites/N32qI3" \t "http://www.tuicool.com/articles/_blank)Restful API 的设计规范[N] 推酷日志.2015.8

http://www.tuicool.com/articles/bUZNZbZ

上传文件格式 （已添加）

Url Encode （已添加）

状态码 5位数 项目编号（已添加）

版本标识 （将版本号放到URL最后，容易实现，调用方便）