ICF 文档

2018-02-07

百度云

cloud.baidu.com

目录

第1章

定时器产品介绍

1.1 定时器产品介绍

1.1.1 概述

定时器提供从云端周期性的发布MQTT消息发送的能力。例如，每天7:30向窗帘控制器，发送开启的命令；每5分钟发送MQTT消息，触发函数计算（CFC）的执行，综合地监控设备的各个指标是否异常等。

1.1.2 产品功能

以固定的间隔，向固定的物接入主题，发送固定的MQTT消息。用户指定开始时间、执行次数、执行间隔，定时器会自动开始执行、直至达到指定的次数。

1.1.3 产品优势

稳定可靠，基于分布式的设计，有效保证了可靠性。

1.1.4 产品定价

暂无收费计划

1.1.5 系统限制

- 定时器发布的消息必须是字符串类型，长度不超过4096个字符；二进制的消息不支持。

- 不支持永远执行的定时器，必须指定一个具体执行次数。
- 开始执行时间，一旦指定，不允许修改。
- 如果发布MQTT消息的物接入实例被删除，则定时器也随之删除。
- MQTT消息以retain=false方式发生，如果消息触发时，订阅端掉线，可能导致消息丢失。
- 定时器的精度为秒。
- 当前仅提供API和SDK访问，暂无Console界面。

第2章

IOT Timer(定时器) API

服务器地址：

区域	服务器地址
华南广州区域	http://re.iot.gz.baidu bce.com
华北北京区域	http://re.iot.bj.baidu bce.com

2.1 1.新建Timer

请求地址 HTTP POST /v1/timer

请求参数

名称	类型	是否必填	默认值	说明
name	String	是		定时器名称
description	String	否	<空>	描述
period	Int	是		定时器触发间隔，单位：秒
beginAt	Int	否	系统当前时间+period	第一次触发时间，UNIX时间戳，单位：秒
times	Int	是		定时器执行次数
endpointName	String	是		定时器触发时，发送MQTT消息的实例名

名称	类型	是否必填	默认值	说明
topic	String	是		定时器触发时，发送MQTT消息的主题
msg	String	是		定时器触发时，发送MQTT消息的内容

请求示例

```
{
    "name": "my_timer_01",
    "description": "trigger every 30 seconds, for 100 times",
    "period": 30,
    "beginAt": 1513667688,
    "times": 100,
    "endpointName": "myendpoint1",
    "topic": "/utils/timer01",
    "msg": "{\\"timer\\":\\"my_timer_1\\"}"
}
```

返回参数

名称	类型	是否必填	默认值	说明
uuid	String	是		定时器id

返回示例

```
{
"uuid":"8bd6102b-24c8-4912-876f-f0b0b08cea18"
}
```

2.2 2.更新Timer

请求地址

HTTP PUT /v1/timer/{uuid}

请求参数

名称	类型	是否必填	默认值	说明
name	String	是		定时器名称
description	String	否	<空>	描述
period	Int	是		定时器触发间隔，单位：秒
times	Int	是		定时器执行次数
endpointName	String	是		定时器触发时，发送MQTT消息的实例名
topic	String	是		定时器触发时，发送MQTT消息的主题
msg	String	是		定时器触发时，发送MQTT消息的内容

请求示例

```
{
    "name": "my_timer_01",
    "description": "trigger every 30 seconds, for 200 times",
    "period": 30,
    "times": 200,
    "endpointName": "myendpoint1",
    "topic": "/utils/timer01",
    "msg": "{\\"timer\\":\\"my_timer_1\\"}"
}
```

返回参数

名称	类型	是否必填	默认值	说明
result	String	是	ok	出错则抛异常

返回示例

```
{
"result":"ok"
```

}

说明：Timer运行后，会记录该Timer已经执行多少次，即executedTimes。如果Timer当前处于运行(RUNNING)状态，更新后的times参数小于等于executedTimes，则Timer进入完成(FINISHED)状态；否则，如果Timer当前处于完成(FINISHED)状态，更新后的times参数大于executedTimes，则Timer进入运行(RUNNING)状态。

2.3 3.读取Timer

请求地址

HTTP GET /v1/timer/{uuid}

请求参数

名称	类型	是否必填	默认值	说明
uuid	String	是		定时器的uuid

请求示例

HTTP GET /v1/timer/8bd6102b-24c8-4912-876f-f0b0b08cea18

返回参数

名称	类型	是否必填	默认值	说明
uuid	String	是		定时器id
name	String	是		定时器名称
accountUuid	String	是		定时器的用户id
description	String	是		描述
period	Int	是		定时器触发间隔，单位：秒
beginAt	Int	是		第一次触发时间，UNIX时间戳，单位：秒
times	Int	是		定时器执行次数
executedTimes	Int	是		定时器已经执行的次数

名称	类型	是否必填	默认值	说明
endpointName	String	是		定时器触发时，发送MQTT消息的实例名
topic	String	是		定时器触发时，发送MQTT消息的主题
msg	String	是		定时器触发时，发送MQTT消息的内容
lastExecuteTime	Int	是		定时器触上次触发时间，如果没有触发过，则为0
status	String	是		定时器状态，可能取值：RUNNING,FINISHED
createTime	String	是		创建时间
updateTime	String	是		更新时间

返回示例


```
{
    "uuid": "8bd6102b-24c8-4912-876f-f0b0b08cea18",
    "accountUuid": "eb105e2e-cbe5-42f2-8464-7cd4cc8316b3",
    "name": "my_timer_01",
    "description": "trigger every 30 seconds, for 100 times",
    "period": 30,
    "beginAt": 1513667688,
    "times": 100,
    "executedTimes": 21,
    "endpointName": "myendpoint1",
    "topic": "/utils/timer01",
    "msg": "{"timer":"my_timer_1"}",
    "lastExecuteTime": 1513667993,
    "status": "RUNNING",
    "createTime": "2017-11-20T11:21:23Z",
    "updateTime": "2017-11-20T11:21:23Z"
}
```

2.4 4.枚举Timer

请求地址

HTTP GET /v1/timer?pageNo={pageNo}&pageSize={pageSize}

请求参数

名称	类型	是否必填	默认值	说明
pageNo	Int	否	1	分页的页标
pageSize	Int	否	50	分页的大小

请求示例

HTTP GET /v1/timer?pageNo=1&pageSize=20

返回参数

名称	类型	是否必填	默认值	说明
totalCount	Int	是		总的定时器个数
pageNo	Int	是		分页的页标
pageSize	Int	是		分页的大小
result	Array	是		定时器详情数组，每个元素个数同接口：读取Timer

返回示例


```
{
    "pageNo": 1,
    "pageSize": 20,
    "totalCount": 1,
    "result": [
        {
            "uuid": "8bd6102b-24c8-4912-876f-f0b0b08cea18",
            "accountUuid": "eb105e2e-cbe5-42f2-8464-7cd4cc8316b3",
            "name": "my_timer_01",
            "description": "trigger every 30 seconds, for 100 times",
            "period": 30,
            "beginAt": 1513667688,
```

```
            "times": 100,
            "executedTimes": 21,
            "endpointName": "myendpoint1",
            "topic": "/utils/timer01",
            "msg": "{\\"timer\\":\\"my_timer_1\\"}",
            "lastExecuteTime": 1513667993,
            "status": "RUNNING",
            "createTime": "2017-11-20T11:21:23Z",
            "updateTime": "2017-11-20T11:21:23Z"
        }
    ]
}
```


2.5 5.删除Timer

请求地址

HTTP DELETE /v1/timer/{uuid}

请求参数

名称	类型	是否必填	默认值	说明
uuid	String	是		定时器的uuid

请求示例

HTTP DELETE /v1/timer/8bd6102b-24c8-4912-876f-f0b0b08cea18

返回参数

名称	类型	是否必填	默认值	说明
result	String	是	ok	出错则抛异常

返回示例


```
{
"result":"ok"
}
```


2.6 6.批量删除Timer

请求地址

HTTP POST /v1/timer/batch/delete

请求参数

名称	类型	是否必填	默认值	说明
ids	Array	是		定时器的uuid数组

请求示例

```
{
    "ids": [
        "8bd6102b-24c8-4912-876f-f0b0b08cea18",
        "30698a2b-24c8-4912-876f-f335508ce9ab"
    ]
}
```

返回参数

名称	类型	是否必填	默认值	说明
result	String	是	ok	出错则抛异常

返回示例

```
{
"result":"ok"
}
```

第3章

Java SDK文档

3.1 概述

本文档主要介绍定时器Java SDK的安装和使用。在使用本文档前，您需要先了解定时器的一些基本知识。

3.2 安装SDK工具包

运行环境

Java SDK工具包可在jdk1.6、jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
    <groupId>com.baidubce</groupId>
    <artifactId>bce-java-sdk</artifactId>
    <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以在SDK下载页面找到。

方式二：直接使用JAR包安装

1. 在官方网站下载Java SDK压缩工具包。
2. 将下载的bce-java-sdk-version.zip解压后，复制到工程文件夹中。

3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。

4. 添加SDK工具包lib/bce-java-sdk-version.jar和第三方依赖工具包third-party/*.jar。

   其中，version为版本号。

SDK目录结构

```
com.baidubce
        ├── auth                                    //BCE签名相关类
        ├── http                                    //BCE的Http通信相关类
        ├── internal                                //SDK内部类
        ├── model                                   //BCE公用model类
        ├── services
        │       └── iottimer                        //定时器的服务相关类
        │           ├── model                      //定时器内部model，如Request
或Response
        │           └── IotTimerClient.class        //定时器客户端入口类
        ├── util                                    //BCE公用工具类
        ├── BceClientConfiguration.class            //对BCE的HttpClient的配置
        ├── BceClientException.class                //BCE客户端的异常类
        ├── BceServiceException.class               //与BCE服务端交互后的异常类
        ├── ErrorCode.class                         //BCE通用的错误码
        └── Region.class                            //BCE提供服务的区域
```

3.3 创建IotTimerClient

用户可以参考如下代码新建一个IotTimerClient：

```
String accessKey = "your-access-key-id";
String secretKey = "your-secret-access-key";
String host = "re.iot.gz.baidubce.com";
// 如果是北京区域，请用 String host = "re.iot.bj.baidubce.com";

 BceClientConfiguration config = new BceClientConfiguration()
              .withCredentials(new DefaultBceCredentials(accessKey, secretKey))
                .withEndpoint(host);
 IotTimerClient client = new IotTimerClient(config);
```

3.4 创建/删除/修改定时器

3.4.1 创建定时器

请参考以下代码创建规则：

```
 // prepare parameters
 String name = "myTimerName";
 String description = "this is a demo timer description";
 String mqttEndpoint = "mymqttendpoint"; // 物接入实例名
 String destTopic = "timer_send_msg_to_this_topic";
 String msg = "{\\"info\\": \\"this message is sent by timer myTimerName, every 30 seconds
\\"}";
 long totalExecuteTimes = 100;
 long period = 30;   // send message every 30 seconds
 // start first time execute period seconds later
 long beginAt = System.currentTimeMillis() / 1000 + period;

 // construct the CreateIotTimerRequest
 CreateIotTimerRequest req = new CreateIotTimerRequest();
 req.setName(name);
 req.setDescription(description);
 req.setEndpointName(mqttEndpoint);
 req.setTopic(destTopic);
 req.setMsg(msg);
 req.setTimes(totalExecuteTimes);
 req.setPeriod(period);
 req.setBeginAt(beginAt);

 // invoke the creation
 String uuid = client.create(req).getUuid();
 // now, uuid the unique id of the timer
```

3.4.2 删除定时器

请参考以下代码删除定时器：

```
// uuid为创建定时器返回的唯一id，请参考创建定时器代码
client.delete(uuid);
```

3.4.3 修改定时器

修改定时器信息 请参考以下代码修改定时器：

```
 String name = "myTimerName_new";
 String description = "this is a demo timer description_new";
 String mqttEndpoint = "mymqttendpoint_new"; // 物接入实例名
 String destTopic = "timer_send_msg_to_this_topic_new";
 String msg = "{\\"info\\": \\"this message is sent by timer myTimerName, every 30 seconds_new
\\"}";
 long totalExecuteTimes = 100 + 200;
 long period = 30 + 30;   // send message every 60 seconds

 UpdateIotTimerRequest req = new UpdateIotTimerRequest();
 req.setName(name);
 req.setDescription(description);
 req.setEndpointName(mqttEndpoint);
 req.setTopic(destTopic);
 req.setMsg(msg);
 req.setTimes(totalExecuteTimes);
 req.setPeriod(period);

 // uuid为创建定时器返回的唯一id，请参考创建定时器代码
 client.update(req, uuid);
```

3.5 查看已经创建的定时器

3.5.1 获取所有定时器名称和ID

请参考以下代码获取所有规则：

```
int pageNo = 0;
ListIotTimerResponse response = null;
do {
    ListIotTimerRequest request = new ListIotTimerRequest();
    request.setPageNo(++pageNo;
    esponse = client.list(request);

    for (IotTimer t : response.getResult()) {
        System.out.printf("Name=%suu, id=%s"tle.getName(), t.getUuid());
    }
```

```
} while (response != null && response.getTotalCount() > response.getPageNo() * response.getPageSize()
```

3.5.2 获取一个定时器的详情

请参考以下代码获取一个定时器的详情：

```
// uuid为创建定时器返回的唯一id，请参考创建定时器代码
IotTimer timer = client.get(uuid);
```

3.6 版本说明

3.6.1 v0.10.25

首次发布。