ALIBABA CLOUD

阿里云

分布式任务调度 SchedulerX

任务类型

文档版本：20210510

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.Java任务

Java调度任务可以在您的应用进程中执行，也可以通过上传JAR包来动态加载。

执行模式

Java任务类型支持单机、广播、并行计算、内存网格、网格计算和分片运行6种执行模式：

- *单机*：在同一个 `groupId` 下的机器随机挑一台执行。
- *广播*：同一个 `groupId` 下的所有机器同时执行。
- *并行计算*：支持子任务300以下，有子任务列表。
- *内存网格*：基于内存计算，子任务50,000以下，速度快。
- *网格计算*：基于文件计算，子任务1,000,000以下。
- *分片运行*：包括静态分片和动态分批，用于处理大数据业务需求。

*单机*和*广播*需要实现JavaProcessor；*并行计算*、*内存网格*、*网格计算*和*分片运行*需要实现MapJobProcessor。

Processor类路径，即实现类的全路径名，例如 `com.apache.armon.test.schedulerx.processor.MySimpleJob` ：

- 如果不上传JAR包，SchedulerX会去您的应用进程中的classpath下查找processor实现类，所以每次修改需要重新编译和发布。
- 如果上传了JAR包，每次会热加载JAR包和processor，不需要重新发布应用。

编程模型

Java任务支持两种编程模型：JavaProcessor和MapJobProcessor。

- JavaProcessor
  - （可选） `public void preProcess(JobContext context) throws Exception`
  - `public ProcessResult process(JobContext context) throws Exception`
  - （可选） `public void postProcess(JobContext context)`
  - （可选） `public void kill(JobContext context)`
- MapJobProcessor
  - `public ProcessResult process(JobContext context) throws Exception`
  - （可选） `public void postProcess(JobContext context)`
  - `public void kill(JobContext context)`
  - （可选） `public void kill(JobContext context)`
  - `public ProcessResult map(List<? extends Object> taskList, String taskName)`

ProcessResult

每个process需要返回ProcessResult，用来表示任务执行的状态、结果和错误信息。

- 任务运行成功： `return new ProcessResult(true)` 。
- 任务运行失败： `return new ProcessResult(false, ErrorMsg)` 或者直接抛异常。
- 任务运行成功并且返回结果： `return new ProcessResult(true, result)` 。 `result` 是一个字符串，不能大于1000字节。

HelloSchedulerx2.0任务示例

```
@Component
public class MyProcessor1 extends JavaProcessor {
  @Override
  public ProcessResult process(JobContext context) throws Exception {
    //TODO
    System.out.println("Hello, schedulerx2.0!");
    return new ProcessResult(true);
  }
}
```

支持Kill功能的任务示例

```
@Component
public class MyProcessor2 extends JavaProcessor {
  private volatile boolean stop = false;
  @Override
  public ProcessResult process(JobContext context) throws Exception {
    int N = 10000;
    while (!stop && N >= 0) {
      //TODO
      N--;
    }
    return new ProcessResult(true);
  }
  @Override
  public void kill(JobContext context) {
    stop = true;
  }
  @Override
  public void preProcess(JobContext context) {
    stop = false; //如果是通过Spring启动，Bean是单例，需要通过preProcess把标记为复位
  }
}
```

通过Map模型批量处理任务示例

```
/**
 * 对一张单表进行分布式批量处理
 * 1. 根任务先查询一张表，获取minId，maxId
 * 2. 构造PageTask，通过map进行分发
 * 3. 下一级获取到如果是PageTask，则进行数据处理
 *
 */
@Component
public class ScanSingleTableJobProcessor extends MapJobProcessor {
  private static final int pageSize = 100;
  static class PageTask {
    private int startId;
    private int endId;
    public PageTask(int startId, int endId) {
```

```
        this.startId = startId;
        this.endId = endId;
    }
    public int getStartId() {
        return startId;
    }
    public int getEndId() {
        return endId;
    }
  }
  @Override
  public ProcessResult process(JobContext context) {
    String taskName = context.getTaskName();
    Object task = context.getTask();
    if (isRootTask(context)) {
      System.out.println("start root task");
      Pair<Integer, Integer> idPair = queryMinAndMaxId();
      int minId = idPair.getFirst();
      int maxId = idPair.getSecond();
      List<PageTask> taskList = Lists.newArrayList();
      int step = (int) ((maxId - minId) / pageSize); //计算分页数量for (int i = minId; i < maxId; i+=step) {
        taskList.add(new PageTask(i, (i+step > maxId ? maxId : i+step)));
      }
      return map(taskList, "Level1Dispatch");
    } else if (taskName.equals("Level1Dispatch")) {
      PageTask record = (PageTask)task;
      long startId = record.getStartId();
      long endId = record.getEndId();
      //TODO
      return new ProcessResult(true);
    }
    return new ProcessResult(true);
  }
  @Override
  public void postProcess(JobContext context) {
    //TODO
    System.out.println("all tasks is finished.");
  }
  private Pair<Integer, Integer> queryMinAndMaxId() {
    //TODO select min(id),max(id) from xxx
    return null;
  }
}
```

2.脚本任务

您可以在创建任务时直接编写Shell、Python和Go脚本以便创建脚本任务。

接入方式

脚本任务支持两种接入方式：

- 嵌在应用进程中
- 安装schedulerx2-agent

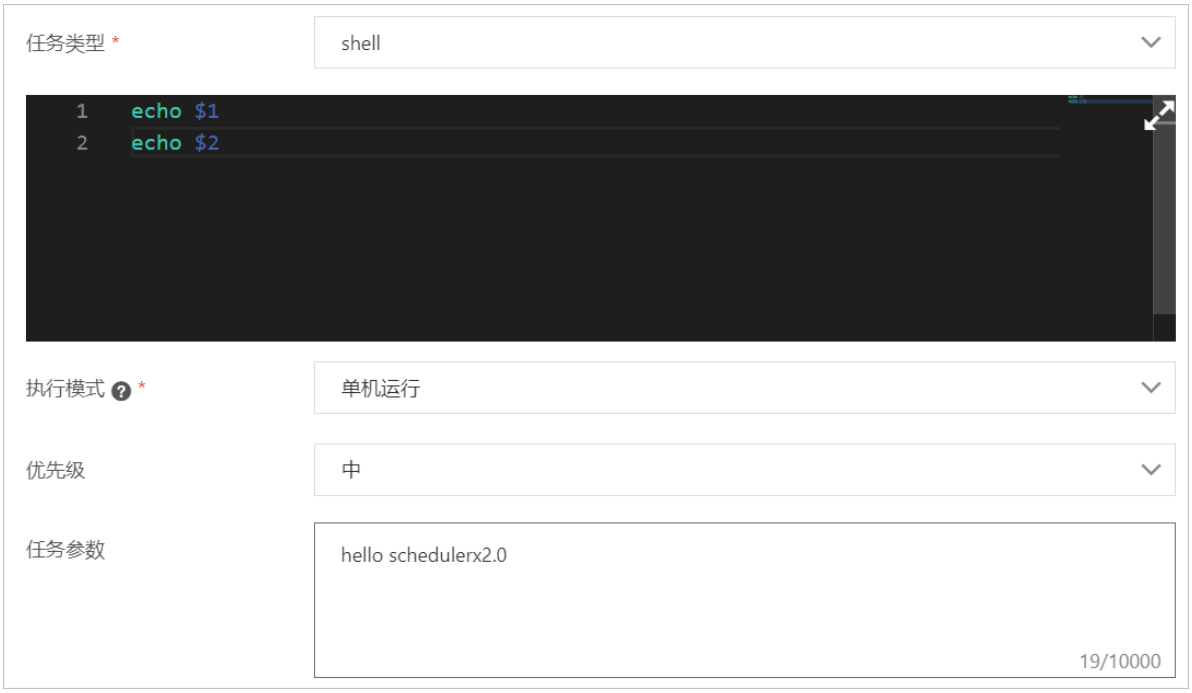
创建方式

脚本任务在创建调度任务时编写脚本即可，详情请参见创建调度任务。

编写脚本的方式包括创建无参数的Shell任务、带参数的Shell任务、Python任务和Go任务：

- 无参数的Shell任务示例。

- 带参数的Shell任务示例。



- Python任务示例。


任务类型 *
python
1 import sys
2 print('Hello Schedulerx2.0')
3 a = int(sys.argv[1])
4 b = int(sys.argv[2])
5 print('a=' + str(a))
6 print('b=' + str(b))
7 c = a+b
8 print('c=' + str(c))
执行模式 *
单机运行

- Go任务示例。


任务类型 *
go
1 package main
2 import (
3 "fmt"
4 "os"
5)
6 func main(){
7 s:= word
8 if len(os.Args) > 1{
执行模式 *
单机运行

3.HTTP任务（Serverless）

SchedulerX支持Serverless的HTTP任务，包含GET和POST两种方法，无需依赖Client，在控制台配置完即可生效使用。

使用限制

- 目前只支持GET、POST，后续根据用户需求陆续开通其他方法。
- HTTP请求返回结果必须是JSON格式，服务需要解析指定key比较本次请求是否成功。
- 不支持秒级任务，支持到分钟级别。
- 请求URL需要有公网权限，如果是 `IP:port` ，需要机器开通公网权限。

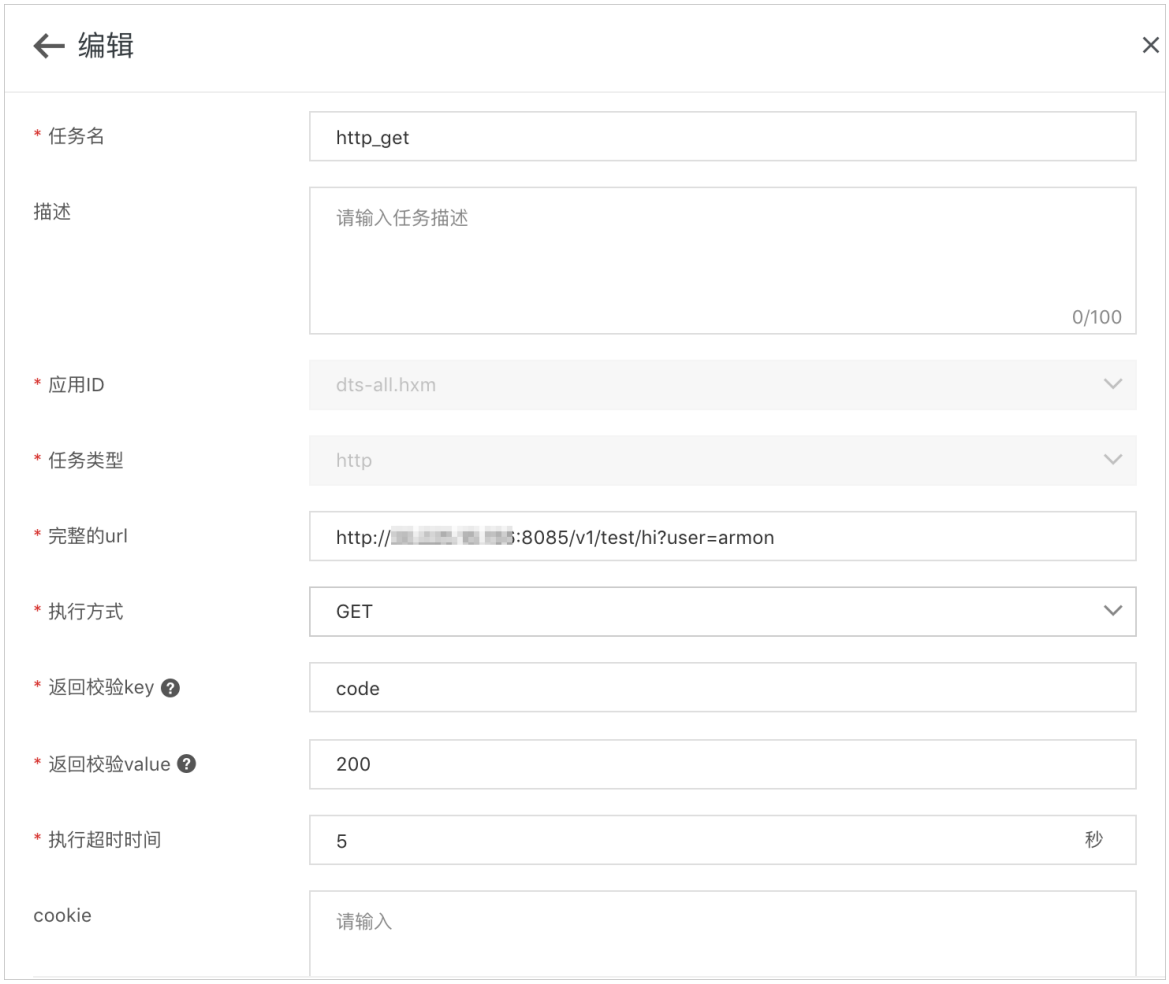
GET

使用HTTP的GET方法，需要在客户端中添加配置，然后在控制台中创建任务。

1. 在客户端中添加配置GET方法配置。客户端接入SchedulerX的详细步骤，请参见Spring Boot应用接入SchedulerX。此处仅介绍GET方法的配置。

```
@GET
@Path("hi")
@Produces(MediaType.APPLICATION_JSON)
public RestResult hi(@QueryParam("user") String user) {
  TestVo vo = new TestVo();
  vo.setName(user);
  RestResult result = new RestResult();
  result.setCode(200);
  result.setData(vo);
  return result;
}
```

2. 在控制台创建HTTP任务。创建调度任务，请参见创建调度任务。此处仅介绍HTTP任务GET方法相关配置。



Serverless HTTP任务参数说明：

参数	说明
完整的URL	需要填写完整URL，包括 `http://` 。
返回校验key和返回校验value	服务端默认HTTP请求结果为JSON格式，根据填写的key和value校验结果是否成功。 `{ code: 200, data: "true", message: "", requestId: "4466550687919236141033812329 71", success: true }` 上面的示例代码可以校验key为success，校验 `value: true` 或者校验code是否为200。
执行超时时间（秒）	最大30秒，超过会报错。

参数	说明
cookie	格式例如 key1=val1;key2=val2 ，多个值用半角分号（;）隔开，最大长度为300字节。

3. 任务创建成功后，在**任务管理**页面的**操作**列单击**运行一次**。
出现以下结果，说明任务执行成功。

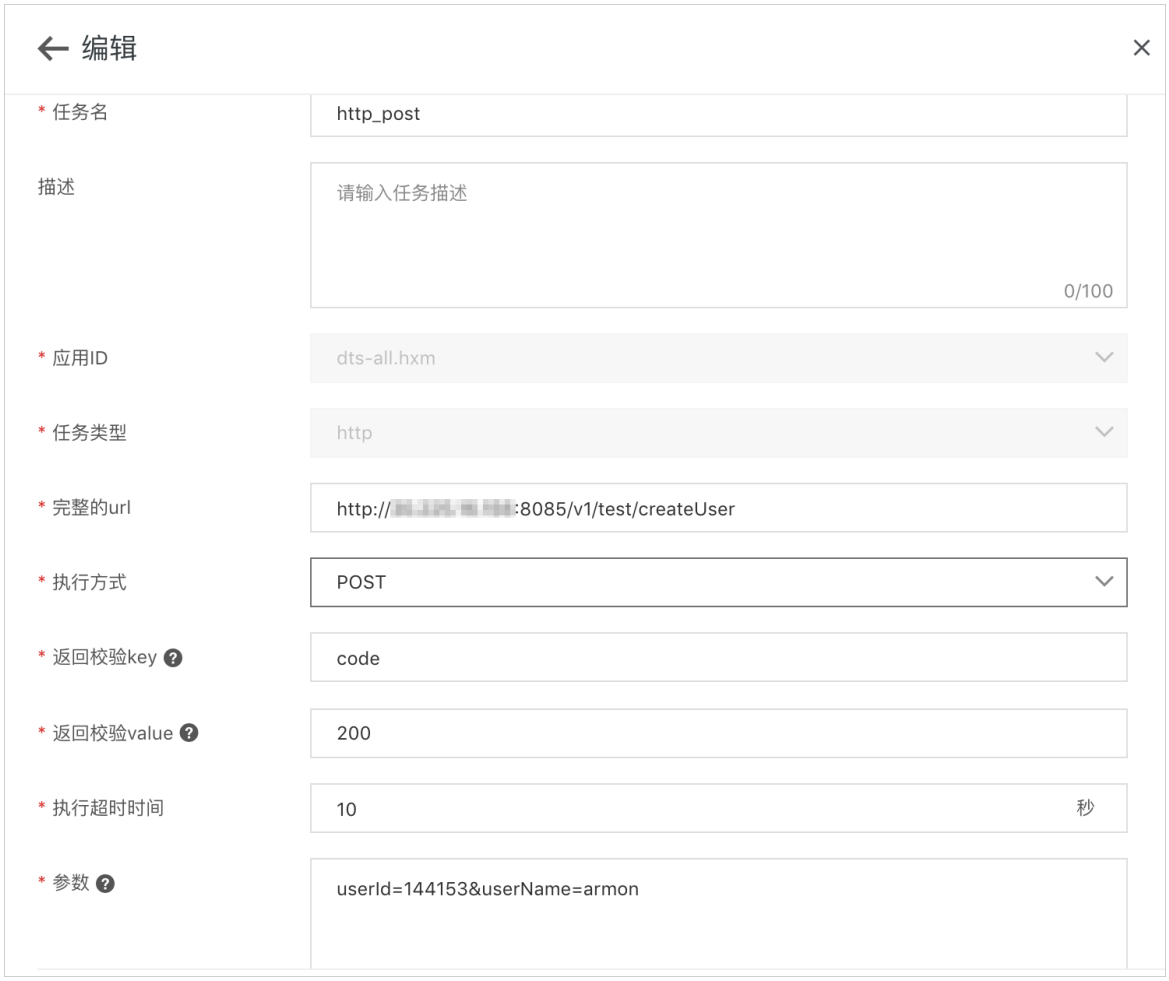
POST

使用HTTP的POST方法，需要在客户端中添加配置，然后在控制台中创建任务。

1. 在客户端中添加配置POST方法配置。客户端接入SchedulerX的详细步骤，请参见Spring Boot应用接入SchedulerX。此处仅介绍POST方法的配置。

```
import com.alibaba.schedulerx.common.constants.CommonConstants;
@POST
@Path("createUser")
@Produces(MediaType.APPLICATION_JSON)
public RestResult createUser(@FormParam("userId") String userId,
    @FormParam("userName") String userName) {
  TestVo vo = new TestVo();
  System.out.println("userId=" + userId + ", userName=" + userName);
  vo.setName(userName);
  RestResult result = new RestResult();
  result.setCode(200);
  result.setData(vo);
  return result;
}
```

2. 在控制台创建HTTP任务。创建调度任务，请参见创建调度任务。此处仅介绍HTTP任务POST方法相关配置。



Serverless HTTP任务参数说明：

参数	说明
完整的URL	需要填写完整URL，包括 `http://` 。

参数	说明
返回校验key和返回校验value	服务端默认HTTP请求结果为JSON格式，根据填写的key和value校验结果是否成功。 `{ code: 200, data: "true", message: "", requestId: "4466550687919236141033812329 71", success: true }` 上面的示例代码可以校验key为success，校验 `value: true` 或者校验code是否为200。
执行超时时间（秒）	最大30秒，超过会报错。
参数	POST表单参数，格式例如 `key1=val1;key2=val2` 。

如何获取任务基本信息

HTTP任务，会将任务基本信息打入header中，如果想获取任务的基本信息，可以在客户端的*pom.xml*中增加以下依赖。

```
<dependency>
  <groupId>com.aliyun.schedulerx</groupId>
  <artifactId>schedulerx2-common</artifactId>
  <version>1.1.5-SNAPSHOT</version>
</dependency>
```

以GET方法为例，通过以下方式获取任务基本信息。

```
import com.alibaba.schedulerx.common.constants.CommonConstants;
@GET
@Path("hi")
@Produces(MediaType.APPLICATION_JSON)
public RestResult hi(@QueryParam("user") String user,
    @HeaderParam(CommonConstants.JOB_ID_HEADER) String jobId,
    @HeaderParam(CommonConstants.JOB_NAME_HEADER) String jobName) {
  TestVo vo = new TestVo();
  vo.setName("armon");
  //JobName可能是中文，需要URLDecode。
  String decodedJobName = URLDecoder.decode(jobName, "utf-8");
  System.out.println("user=" + user + ", jobId=" + jobId + ", jobName=" + decodedJobName);
  RestResult result = new RestResult();
  result.setCode(200);
  result.setData(vo);
  return result;
}
```

可以获取的任务基本信息如下：

CommonConstants常量	key	value描述
JOB_ID_HEADER	schedulerx-jobId	任务ID
JOB_NAME_HEADER	schedulerx-jobName	任务名，需要引文，因为header不支持中文。
SCHEDULE_TIMESTAMP_HEADER	schedulerx-scheduleTimestamp	调度时间的时间戳
DATA_TIMESTAMP_HEADER	schedulerx-dataTimestamp	数据时间的时间戳
GROUP_ID_HEADER	schedulerx-groupId	应用ID
USER_HEADER	schedulerx-user	用户名
MAX_ATTEMPT_HEADER	schedulerx-maxAttempt	实例最大重试次数
ATTEMPT_HEADER	schedulerx-attempt	实例当前重试次数
JOB_PARAMETERS_HEADER	schedulerx-jobParameters	任务参数
INSTANCE_PARAMETERS_HEADER	schedulerx-instanceParameters	任务实例参数，需要API触发

结果验证

HTTP任务执行结果在执行列表页可以进行查询，成功结果可参见GET中的结果。

如果失败可以单击**详情**查看具体失败原因，下面列举典型的失败结果。

- 返回值和期望不相同

- 执行超时

报警

HTTP任务支持错误报警，针对上述超时以及返回值不符合预期用户可以在创建任务时设置报警信息，接收相应的报警信息。