啄木鸟

简介⤻

因现在使用的接口服务多种多样，每个服务的请求、响应格式都不相同，需要针对每一类服务都开发对应的自动化测试工具，因此，开发了该测试工具，尽可能的做到通用，防止重复发明轮子。

结构

使用项目、服务、版本来管理不同项目的多个服务，以及每个服务的多个版本。

项目

用于区分不同的业务线，比如：公交、搜索等，针对每个项目，可以配置自己独立的SVN路径，SVN路径需要配置到版本目录的上一级目录（现暂时只支持通过SVN管理CASE）。

版本

每个项目都有自己单独的版本，比如：735，736等

服务

业务线提供的服务，比如：搜索除了搜索以外，还有GEO、RGEO等子服务

目录结构

通过SVN 组织、管理CASE的时候，需要按照 版本->服务 的目录结构组织。例如，snowman项目的svn目录结构如下：

```
--svn_case
        --版本1
                --服务1
                        --测试用例1（文件夹）
                                --测试用例1-1（文件）
                --服务2
                        --测试用例2（文件）
        --版本2
                --服务1
                        --测试用例1（文件）
                --服务2
                        --测试用例2（文件）
```

针对以上目录，名称为snowman的项目svn路径配置，需要将svn路径配置到case这一级：http://domain/svn_case

针对以上目录，名称为snowman的项目svn路径配置，需要将svn路径配置到case这一级：http://domain/svn_case

版本、服务必须在系统中存在（包括大小写都必须一致），否则运行的时候，会无法匹配上CASE目录

配置

针对不同的请求类型，配置格式有所不同。有一些是通用的配置，所有的配置文件都是以xml进行配置。
注：配置可通过服务管理页面直接添加

支持的请求类型

现支持以下几种格式的请求类型：

- protocol buffer
- http get
- http post
- http option

支持的数据交换协议

现在支持解析、比对的响应格式有以下几种：

- protocol buffer
- JSON
- XML

通用配置

一个最简版本的配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<root type="HTTP_POST" requestType="JSON" resultType="XML" resultCharset="GBK"
requestClass="com.snowman.test.Request" responseClass="com.snowman.test.Response" >
<description>
<![CDATA[测试服务]]>
</description>
[<property></property>]
</root>
```

其中

type代表请求类型，现支持的值如下：

- PB
- HTTP_GET
- HTTP_POST
- HTTP_OPTION
- CUSTOM —— 自定义请求方法，比如：连接redis、mq这种非http请求，使用CUSTOM时，需要指定requestClass，使用reuqestClass执行

resultType代表服务返回的格式，现支持：

- PB (protocol buffer)
- JSON
- XML

- BINARY —— 会将二进制返回结果以base64进行转码，由于储存、展示
- IMAGE —— 会将图片转码成base64的形式，在结果页展示

requestType代表发往服务器的请求格式（HTTP_POST才使用该字段），现支持：

- PB
- JSON
- XML

requestCharset生成请求是使用的字符编码，若不做配置，默认使用UTF-8。

resultCharset代表服务返回结果的字符编码，若不做配置，默认使用UTF-8。

requestClass 请求类型为PB时，向服务器发送请求时，指定用于执行merge生成PB的类。

当请求类型为CUSTOM时，必须指定自定义发送请求的requestClass，requestClass需要包含指定方法：execute(String runUrl, Map> params)；返回结果可为预定义的格式，或者由responseClass指定。其中，runUrl是用于运行的url，运行进配置，params为case当中指定的参数集。

例如：

```
__name__ = "test redis"
__command__ = "select 0"
__command__ = "dbsize"
__command2__ = "select 1"
```

那么运行时，传入execute方法的params值为：

```
{
command2=[select 1],
command=[select 0, dbsize]
}
```

responseClass 需要包含一个parseFrom(byte[] content)静态方法，返回结果类型可为：

com.google.protobuf.Message —— 会转换成可读格式

String —— String的内容类型需要与resultType指定的一致，

或者是普通的javabean —— 会自动将javabean转换成对应的json。

description对服务进行概要描述，不是必填选项，但是推荐填写，用于生成帮助文档

在根节点下面，可以有多个子节点，会通过配置的子节点与CASE相匹配，生成对应的请求，示例子节点如下：

```
    <property>
        <description>交通工具换乘避免，多个值使用逗号分隔</description>
        <allowable-values><![CDATA[
        AVOID_OTHER,AVOID_METRO -- 避免地铁,AVOID_RAIL,AVOID_LIGHT_RAIL,AVOID_BUS -
- 避免公交,AVOID_FERRY -- 避免轮渡
        ]]></allowable-values>
        <name>orgin.waypointId</name>
        <parent>DirectionsRequest/originWaypointTyped[0]</parent>
        <field>waypoint_id/muid</field>
        <type>long</type>
        <linkproperty>
            <field>waypoint_type</field>
            <type>enum</type>
            <value>WAYPOINT_ID</value>
        </linkproperty>
    </property>
```

对应标签的含义，可参考字段配置查看配置示例：

- name 编写CASE时使用的字段名，单个配置文件要求唯一
- parent 字段路径
- field 实际请求中的字段名，当以@开头，表示这个是xml中的属性
- type字段类型，现在支持的字段类型如下：
  - base64 当字段对应的是protocol buffer的ByteString时，可使用base64对值进行编码，方便case编写
  - string
  - int
  - double
  - enum
  - time 返回TimeInMillis，值格式：yyyy-MM-dd HH:mm:ss，yyyy-MM-dd HH:mm，HH:mm:ss，HH:mm，EEE HH:mm（Thursday 08:00），EEE HH:mm:ss（Thursday 08:00:00 or 星期三 08:00:00）
  - cfabsolutetime 返回cfabsolutetime（单位秒），值格式：yyyy-MM-dd HH:mm:ss，yyyy-MM-dd HH:mm，HH:mm:ss，HH:mm，EEE HH:mm（Thursday 08:00），EEE HH:mm:ss（Thursday 08:00:00 or 星期三 08:00:00）
- snowman特有字段：
  - LatLng 例如：39.990173,116.482296，会生成
    - ```
      <lat>39.990173</lat><lng>116.482296</lng>
      ```
  - mapRegion 例如：39.990173,116.482296,39.990173,116.482296，会生成
      ```

```
<southLat>39.990173</southLat>
<westLng>116.482296</westLng>
<northLat>116.482296</northLat>
<eastLng>116.482296</eastLng>
```

- linkproperty 关联的常量字段，可有多个。例如：当A字段存在，那么B字段的值必定为C，这时候，可以使用linkproperty，在配置A字段时，将B字段的值设置为C，详细可参考字段配置
- subproperty 可用于配置添加一些自定义类型，用于一些带有子字段的可重复字段。详细可参考字段配置
- description 字段描述，非必填项，但是推荐填写，用于生成帮助文档
- allowable-values 取值范围，不同值之间使用逗号分隔，非必填项，但是推荐填写，用于生成帮助文档
- default-value 默认值， 当节点存在默认值，就算不声明也会生成，若声明了，会优先使用声明的值
- repeatable 非必填项，但是推荐填写，用于生成帮助文档表示字段是否可重复

例如：
有以下配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<root type="HTTP_POST" resultType="JSON" resultCharset="GBK" requestType="XML">
</description>
    <property>
        <name>request_src</name>
        <parent>request</parent>
        <field>@reqsrc</field>
        <type>string</type>
        <linkproperty>
            <field>@reqtype</field>
            <type>string</type>
            <value>0</value>
        </linkproperty>
        <default-value>snowman</default-value>
    </property>
    <property>
        <name>request_version</name>
        <parent>request</parent>
        <field>@vers</field>
        <type>string</type>
        <default-value>2.1</default-value>
    </property>
        <property>
        <name>request_version</name>
        <parent>request/route</parent>
        <field>@type</field>
        <type>string</type>
        <default-value>0</default-value>
    </property>
    <property>
        <name>request_version</name>
        <parent>request/route</parent>
        <field>@flag</field>
        <type>string</type>
        <default-value>0x40</default-value>
    </property>
</root>
```

假设编写以下case

```
__name__ = "test default value"
__request_src__ = "snowman2.0"
```

那么， 实际解析出来的请求为：

```
<?xml version="1.0" encoding="UTF-8"?>
<request vers="2.1" reqtype="0" reqsrc="snowman2.0"><route type="0" flag="0x40"/>
</request>
```

vers、type、flag没有在用例中声明，同样会在请求中使用默认值生成

protocol buffer

protocol buffer的一个初始配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<root type="PB"
requestClass="com.apple.geo.protobuf.geo3.Directions.DirectionsRequest"
responseClass="com.apple.geo.protobuf.geo3.Directions.DirectionsResponse">
</root>
```

其中， requestClass指明需要将请求封装成什么样的protocol buffer message，

responseClass 指明需要使用哪个protocol buffer message解析服务器返回的二进制数据流。若指定的class没有在啄木鸟工程内，需要将对应的jar放置到对应的目录下。

除了初始配置以外，还需要配置请求参数。一个示例的请求参数如下：

```
        <property>
                <name>orgin.waypointId</name>
                <parent>DirectionsRequest/originWaypointTyped[0]</parent>
                <field>waypoint_id/muid</field>
                <type>long</type>
                <subproperty>
                        <field>waypoint_type</field>
                        <type>enum</type>
                        <value>WAYPOINT_ID</value>
                </subproperty>
        </property>
```

详细配置方法，请参考字段配置

http get

一般的http get请求，直接使用最简版本的配置就可以了，若为了便于阅读，要为url参数添加别名，可在配置文件中添加属性。
例如：

一个HTTP_GET的URL是：
/busEngine?
A1=110105&A2=230803&X1=116.48043201434153&Y1=39.9895948721543&X2=130.361008&Y2=46.810871

针对参数A1,A2不是很好理解，无法一眼看出代表的什么含义，可以配置文件中添加属性（***注：因为http get都是url格式，不需要配置type、parentpath等相关属性***）：

```
        <property>
                <name>orgin.citycode</name>
                <field>A1</field>
        </property>
        <property>
                <name>destination.citycode</name>
                <field>A1</field>
        </property>
```

在编写CASE的时候，就可直接使用：

```
__orgin.citycode__=110105
__destination.citycode__=230803
```

在解析CASE的时候，会直接把orgin.citycode解析成A1，destination.citycode解析成A2

http post

根据请求的格式不同， 现在支持xml以及json两种， 但是配置方面都保持一致性，配置方式没有差别。

详细配置方法，请参考字段配置

CASE编写

格式

一个典型的CASE如下：

```
# 这是一个注释，
# 注释可以有多行
__name__ = "纯步行路线 - 小于600m zh-Hans_CN"
__orgin.waypointPlace__ = "39.990173,116.482296"
__destination.waypointPlace__ = "39.990173,116.483296"
__transportType__ = "TRANSIT"
{
        match_count=1
        with .etaResult{
                .status = "STATUS_SUCCESS"
                .placeSearchResponse.status = "STATUS_SUCCESS"
                with .sortedETA{
                        .transportType = "TRANSIT"
                        .status = "STATUS_SUCCESS"
                        .distance > "400"
                        .travelTimeBestEstimate > "360"
                }
        }.match_count=1
}
```

注：请求与预期结果之前不能有空白行，不同CASE之前通过空白行做分隔（空白行可以有多个）

请求

每个请求都是由多个请求字段组成的， 一个请求字段的格式要求：

以两个下划线开始，以两个下载线结果，后面接字段对应的value，

```
__fieldName__ = "value" // 双引号
__fieldName__ = 'value' // 单引号
```

注：若value中有双引号，则应该使用单引号，反之使用双引号，例如：t = 'a"b',同时，最后一个单引号或者双引号的内容会当作注释忽略，例如：t = 'a"b' //comment 取到的value为a"b

预定义字段

有几个通用的预置字段如下：

```
__name__ = "用例名称"
__tag__ = "用例标签"  // 可根据标签对case进行过滤，按标签执行测试用例
__start__ = "subclass"  //存在关联关系的子CASE开始
__service__ = "eta"  //子CASE所属服务，若没有该字段，则直接使用父CASE的service
__env__ = "1508" //子CASE运行环境，若没有该字段，则直接使用父CASE的env
__project__ = "snowman" //子CASE所属项目，若没有该字段，则直接使用父CASE的project
```

- __name__ 是必填选项，若没有该case会直接忽略，不做解析、执行，取值要求同一个CASE文件中保持唯一（不唯一不会导致执行失败，但是后续要做的CASE对比默认使用CASE名称做为匹配项，若有重复，可能导致匹配结果不正确的情况）

- __start__ 表示关联case开始，详细可查看case关联
- __tag__ 用例标签，例如：p1；一个用例支持有多个tag，一个tag也可有多个值，多个值使用 逗号 分隔，运行测试用例的时候，可指定tag进行过滤，只运行指定tag的测试用例。tag有一个预定义值： skip ， 若为测试用例添加一个tag值为skip，那么在运行的时候，会直接忽略不运行。
- __run_url__ 指定运行环境，**注：应当只在做服务监控时使用，正常case请勿使用，会导致case变得混乱**
- __end__ 表示关联case结束，详细可查看case关联
- __project__ 关联case所属项目，**注：应当只在case关联使用，防止CASE变混乱**
- __service__ 关联case所属服务，**注：应当只在case关联使用，防止CASE变混乱，若使用，需要保证这几个最先声明**
- __env__ 关联case运行环境（版本），**注：应当只在case关联使用，防止CASE变混乱**
- __content__ 直接使用content的值做为请求内容，不执行解析等操作，当单独的请求字段与content中的内容有冲突时，会覆盖content中的原有值。**注：当请求类型为PB时，该处应该填写编码后的BASE64，当请求类型为HTTP_GET时，不会覆盖，只会在url后面追加**
- __header__ 需要添加的头信息，可以存在多个header，值格式为：key:value ， 例如：Origin:https://foo.itunes.com
- __request_script__ 使用自定义脚本对request进行处理，并以处理后的结果作为request
- __response_script__ 使用自定义脚本对response进行处理，并以处理后的结果作为response
- __response_check_script__ 使用自定义脚本对response进行检查，检查失败（若输出的内容不为空，则认为检查失败，会把输出的内容当作错误信息给出）CASE执行失败，通过继续与预期结果进行对比
- __request_class__ 使用自定义类生成request。 **只支持PB协议，且request_class必须是正确的pb协议入口类**
- __response_class__ 使用自定义类对response进行处理，并以处理后的结果作为response。 **当返回结果是PB时，response_class必须是正确的pb协议入口类**
- __case_type__ case类型，可取值：
  - HTTP_GET
  - HTTP_POST
  - HTTP_OPTION
  - PB
- __case_request_charset__ 请求的编码类型， **HTTP_GET会使用该编码对请求对编码**
- __case_request_type__ 请求类型，可取值:
  - PB
  - JSON
  - XML
  - BINARY
- __case_response_charset__ 返回结果的编码类型，默认为UTF-8
- __case_response_type__ 返回结果类型，可取值:
  - PB

    - JSON
    - XML
    - BINARY
- __save_cookie__ 缓存当前用例的cookie，用于接下来使用，可取值：
    - true
    - false
- __use_cookie__ 使用缓步的cookie，值为对应的关联case名称。例如：

```
#缓存login的cookie，关联case prepare使用login的cookie，并更新cookie，主case 使用prepare更新后的cookie。
__name__ = "testCookies"
__start__ = "login"
__request__= "request"
__save_cookie__="true"
__end__ ="login"
__start__="prepare"
__use_cookie__="login"
__save_cookie__="true"
__end__="prepare"
__mainCase__="request"
__use_cookie__="prepare"
```

注：request_script/response_script/response_check_script 暂只支持python、shell脚本，执行时会将脚本的所有控制台输出当作执行结果

有三个snowman专用的字段（protocol buffer请求）：

- __rpc_locale__ RPC头语言，例如：zh_CN,en_US等等，默认为 zh_CN
- __rpc_os_version__ RPC头中包含的IOS版本信息，例如：9.9.9999，8.9.9999，默认为9.9.9999
- __rpc_appid__ RPC头中包含的应用信息，有时候会根据appid判断请求由哪发出，默认为com.apple.Maps，例如：
    - com.apple.Maps 大陆请求
    - com.apple.geo.Localsearch 港澳台请求（国外请求同样使用该appid）

示例

在请求当中，若配置某个字段可重复，则直接写多个，例如：

```
__component__ = "COMPONENT_TYPE_ENTITY"
__component__ = "COMPONENT_TYPE_PLACE_INFO"
__component__ = "COMPONENT_TYPE_ADDRESS"
```

针对http_post请求，以及pb请求， 当未在配置文件中配置对应的字段时，会使用请求字段来构造请求，以json格式举例：

在配置文件未配置任何字段的时候

```
__name__ = "test"
__sign__ = "42d10f1045eb218b2bd6551edd5abc36"
__ts__ = "1435817832663"
__data[0].linkId__ = "2680"
__data[0].roadClass__ = "41000"
__data[0].speed__ = "80"
__data[1].linkId__ = "2681"
__data[1].roadClass__ = "41001"
__data[1].speed__ = "81"
```

会生成以下请求：

```
{
  "sign": "42d10f1045eb218b2bd6551edd5abc36",
  "ts": "1435817832663",
  "data": [
    {
      "linkId": "2680",
      "roadClass": "41000",
      "speed": "80"
    },
    {
      "linkId": "2681",
      "roadClass": "41001",
      "speed": "81"
    }
  ]
}
```

其中，data[0].linkId 代表是请求字段在son中的路径 。[0] 表示data是数组，并将对应的值写到第一个数组当中。

注意，以这种格式写case，值类型都会默认为字符串

case关联

当请求之前存在关联时，或者想要对比两个请求内容，可以case当中添加关联关系。

例如：想要判断驾车ETA给出的时间与驾车引擎路径规划给出的时间是否一致，

```
__name__ = "ETA与路线时间对比"
__start__ = "eta_request"
__service__ = "eta"
__orgin.waypointPlace__ = "39.990173,116.482296"
__destination.waypointPlace__ = "39.990173,116.182296"
__mainTransportType__ = "AUTOMOBILE"
__end__ = "eta_request"
__start__ = "eta_request2"
__orgin.waypointPlace__ = "39.990173,116.482296"
__service__ = "eta"
__destination.waypointPlace__ = "39.990173,116.182296"
__mainTransportType__ = "AUTOMOBILE"
__end__ = "eta_request2"
__orgin.waypointPlace__ =  "39.990173,116.482296"
__destination.waypointPlace__ = "39.990173,116.182296"
__transportType__ = "AUTOMOBILE"
{
 .route[0].expectedTime = "$(eta_request.etaResult[0].liveTravelTime)"
}
```

其中， *_start__ 代表关联case开始，* _end__代表关联case结束，两者的值需要一致。

*_service__ ,*_project__ ,*_env_* 分别代表关联case所属服务、项目、需要运行的环境，若不配置，则使用主case的对应的值，像上面的示例，就默认使用主case的项目、运行环境。 **若使用，需要保证这几个最先声明**

$(eta_request.etaResult[0].liveTravelTime) 是结果表达式，eta_request 是关联case的名称，.etaResult[0].liveTravelTime是从关联case中取值的结果表达式，**注：关联case的名称请勿使用中文、空格以及特殊字符**

关联case除了用于预期结果，也可用于请求当中，例如，可以这么写：

__transportType__ = "$(eta_request.etaResult[0].transportType)"

针对不同的请求类型，有不同的注意点，

支持的参数列表

针对每个服务支持的请求参数都不一样，详细请参考服务参数列表

http get

http get使用预置的content字段：

```
__content__ = "baseUrl"
```

可将通用的url参数放入url字段当中，例如：

针对URL：

pure_walk=0&group=1&req_num=5&Ver=2&Src=snowman&multiexport=1&format=xml&req_taxi=0&X1=116.427307&Y1=39.903741&X2=116.379026&Y2=39.865026

假设 pure_walk=0&group=1&req_num=5&Ver=2&Src=snowman&multiexport=1&format=xml&req_taxi=0 变化较少，那么可以请求可以这么编写：

```
__content__ =
"pure_walk=0&group=1&req_num=5&Ver=2&Src=snowman&multiexport=1&format=xml&req_taxi=0
"
__X1__ = 116.427307
__Y1__ = 39.903741
__X2__ = 116.379026
__Y2__ = 39.865026
```

若服务的访问URL为：

http://10.17.130.10:12248/busEngine?

那么，解析出来的请求为：

http://10.17.130.10:12248/busEngine?pure_walk=0&group=1&req_num=5&Ver=2&Src=snowman&multiexport=1&format=xml&req_taxi=0&X1=116.427307&Y1=39.903741&X2=116.379026&Y2=39.865026

预期结果

格式

一个简单的预期结果如下所示：

```
{
        http_code = "200"
        headers.Content-Length != ""
        match_count="1"
        with .etaResult{
                .status = "STATUS_SUCCESS"
                .placeSearchResponse.status = "STATUS_SUCCESS"
                with .sortedETA[0]{
                        .transportType = "TRANSIT"
                        .status = "STATUS_SUCCESS"
                        .distance > "400"
                        .travelTimeBestEstimate > "360"
                }
        }.match_count=2
}
```

以 . 开头的字符串，例如：.status 是一种表达式，通过该表达式可以从服务器返回结果中获取所感兴趣的值，具体的语法以及支持的操作类型， 可以参考 结果表达式。同时，可在case编辑页面进行实验，了解每一种不同类型的表达式的实际效果。

支持对消息体的结果匹配，例如： 上例中的 .etaResult{ ... }.match_count=2 要求整个.etaResult要匹配上2次。

预期结果有几种类型：

- **以.或者[开头，代表叶子节点，用于做结果匹配**
- 以with开头，用于对结果做过滤，取到感兴趣的值，以with开头，代表接下来是一个消息体(可以是数组:.sortedETA，也可是数组的单个元素:.sortedETA[0]，或者非数组的单个消息体:.etaResult)，路径表达式后必须以 { 做为开始，并在消息体的最后用 } 括起来
- 以纯字母开头，代表预定义字段或者变量

不以.开头的表达式，都认为是预定义的字段或者是变量声明，现在预定义的字段只有一个：match_count ， 只有匹配的结果数符合match_count设定的值，CASE才算通过。match_count不是必须填写的， 若没有填写，默认只要匹配的结果数>=1就算通过。

预期结果格式为：

.path operation expectValue ， 其中，expectValue的值用双引号括起来，若预期结果有可能的多个值，则使用|分隔，例如：餐厅|酒店 ，那么只要结果是餐厅或者酒店任一一个，都通过。 ***(注：当操作符为正则表达式类型时，不支持通过|分隔多个预期结果，因为单个正则表达式中就有可能会包含|。所以，当预期结果使用正则表达式时，只支持一个预期结果。)***

预定义字段

列表：

- match_count 希望匹配的预期结果数
- http_code 服务器返回的状态码
- latency 测试用例执行时间，只计算从发送请求，到接收到服务器返回的response 所花的时间
- body 经过解析后的消息体
- headers.headname 服务器返回的头信息，其中headers. 代表从头信息中取相关信息，headname替换成对应的头信息名称
- compare_type 只在服务对比的时候有效，且针对整体内容进行对比时有效，取值范围如下：
  - STRICT不支持扩展，数组顺序必须完全匹配
  - LENIENT支持扩展，数组顺序可不一致，只要保证都包含同样的内容
  - NON_EXTENSIBLE 不支持扩展，数组顺序可不一致，只要保证都包含同样的内容
  - STRICT_ORDER 支持扩展，数组顺序必须完全匹配
    扩展的含义：实际结果比预期结果多字段。例如：预期 {"a":1} 实际 {"a":1,"b":2} 若支持扩展，这两者对比通过
    数组顺序不一致： 登台，预期：{"a":[1,2,3]} 实际 {"a":[3,2,1]} 若数组顺序可不一致，这两者对比通过

支持的操作符

以下操作符（正则表达式不支持，因为在正则表达式中，会出现|）都支持多个预期结果，只需要有一个匹配成功，就匹配成功，多个匹配结果之间使用 | 作为分隔，

例如：name = 'test1|test2' ,当name的值为test1或者test2时，都能匹配成功。

operation，即操作符。 现支持的operation有以下几种：

操作符	描述	示例
>	适用于数值类预期结果	
>=	适用于数值类预期结果	
<	适用于数值类预期结果	
<=	适用于数值类预期结果	
=	适用于字符串以及数值(区分大小写)	
!=	适用于字符串以及数值(区分大小写)	
has	适用于数组类结果，若结果不是数组，则等价于 = 操作符，单独支持 & 操作符，要求多个预期结果都匹配才成功	[1,2,3] has "1
hasNot	适用于数组类结果，若结果不是数组，则等价于 != 操作符，单独支持 & 操作符，要求多个预期结果都不匹配才成功	[1,2,3] hasNot "4
between	适用于数值类预期结果，范围使用 ~ 分隔，例如："30 ~ 50"，注：必须使用双引号括起来	
like	模糊匹配，适用于字符串，当实际结果包含预期结果则通过，若实际结果是预期结果的字集，认定为失败(不区分大小写)	124路电车 like "124路", success 24路电车 like "124路电车", fail
unlike	模糊匹配，适用于字符串，当实际结果不包含预期结果则通过(不区分大小写)	24路电车 unlike "124路", success 124路电车 unlike "124路电车", fail
in	模糊匹配，适用于字符串，当预期结果包含实际结果则通过，若预期结果是实际结果的字集，认定为失败(不区分大小写)	24路 in "124路", success 124路电车 in "124路", fail
notIn	模糊匹配，适用于字符串，当预期结果不包含实际结果则通过(不区分大小写)	24路 notIn "124路", fail 124路电车 notIn "124路", success

=~	匹配正则表达式（JAVA语法的正则表达式）(不区分大小写)	_abc_ =~ "_[a-z]+_", success _abc12_ =~ "_[a-z]+_" fail
!~	不匹配正则表达式（JAVA语法的正则表达式）(不区分大小写)	_abc_ !~ "_[a-z]+_", fail _abc12_ !~ "_[a-z]+_" success

变量

结果之间有关联关系的时候，可以使用变量。

变量定义的格式为：valName := 'value'

变量的值可以直接是常量，例如： val:="abc"

或者是结果表达式，例如： val:="$(.store.books)" ,支持嵌套，例如：val:="$(.store.books[$(.store.price)])" 同时，支持默认值，例如：val:=$(.store.books.name:abc) 当.store.books.name 取不到结果时，val:=abc

或者是函数例如：val:="PathUtils.count(.route)"，假设路径中存在3个route，那么变量val的值为3

当变量值是结果表达式时，格式为： $(表达式) ，表达式可为jsonpath、函数、变量，同时，支持表达式嵌套。

例如，以下都是合法的表达式：

- $(.store.books) --- jsonpath
  - $(PathUtils.count(.store))--- 函数
    - $(.store.books[${index}])--- 表达式嵌套变量
  - $(.store.books[$(PathUtils.sub(PathUtils.count(.store.books),1))])--- 表达式嵌套

需要使用定义的变量时，格式为：${变量名}

示例：
假设有以下JSON：

```
{
"store": {
  "books":[{
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price_index": 0
            },
            {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour",
                "price_index": 2
            }]
       },
    "prices":[
        {"price":10},
        {"price":20},
        {"price":30}
      ]
}
```

若想取到title为Sayings of the Century 的书的价格，可以通过变量获取：

```
{
        priceIndex := "$(.store.books[?(@.title=='Sayings of the
Century')].price_index)"
        .prices[${priceIndex}].price = "10"

}
```

或者这么写：

```
{
        with .store{
                with .books[?(@.title=='Sayings of the Century')]{
                        priceIndex := "$(.price_index)"
                }
        }

        with .prices[${priceIndex}]{
                .price = "10"
        }

}
```

变量在结果表达式，以及预期结果中都适用。同时，也支持常量的声明，比如：

```
{
        variable := "abc"
        .keypath = "${variable}"
}
```

暂不支持运算，例如：

```
a:="1"
b:="2"
.c="${a}+${b}"  //等价于 .c="1+2"，不会对1+2进行运算
```

变量支持嵌套，例如：${val${subval}}

变量支持默认值，变量名与默认值使用冒号分隔，例如：

```
{
        priceIndex := "$(.store.books[?(@.title=='Sayings of the
Century')].price_index)"
        .prices[${priceIndex:1}].price = "10"

}
```

当变量priceIndex取不到值，${priceIndex:1}的值就为1

注意，当变量与结果表达式混合时，只支持结果表达式嵌套变量的形式，比如：$(.test${abc})

注意，当使用下面这种方式写时，若 .books[?(@.title=='Sayings of the Century')] 能匹配出多个结果，那么priceIndex 的值为最后一个可取到的 .price_index

注意，变量的值，可以在运行时，通过自定义参数传入。

注意，变量声明应该放到所有结果比较之前

函数

当使用通用的操作符无法达成目的时候，可以使用函数，假设有以下结果：

```
{
                "store": {
                        "books":[{
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price_index": 0
                },
                {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour"
                }]
                }
        }
```

当要求price_index字段必须存在时，使用通用操作符无法判断，可使用函数实现，预期结果可以这么写：

```
with .store{
        with .books{
                PathUtils.isExist(.price_index) = "true"
        }
}
```

函数支持嵌套使用，例如：

OperationUtils.plus(PathUtils.count(.abc),1)

有以下工具类：

PathUtils， 支持的方法：

- **isExist(String path)** 若指定结果表达式存在，则返回true，否则返回false
- **isNotExist(String path)** 若指定结果表达式不存在，则返回true，否则返回false
- **isEmpty(String path)** 若指定结果表达式不存在或者值为空，则返回true，否则返回false
- **isNotEmpty(String path)** 若指定结果表达式存在且值不为空，则返回true，否则返回false
- **count(String path)** 若指定结果表达式不存在，返回0，为单个值，返回1，为数组，返回数组长度
- **length(String path)** 返回指定结果表达式的值长度
- **toLowerCase(String path)** 将指定结果表达式的值转换成小写
- **toUpperCase(String path)** 将指定结果表达式的值转换成大写
- **length(String path)** 返回指定结果表达式的值长度
- **base64Decode(String path)** 对指定结果表达式的值进行base64解码
- **distance(String startLatPath, String startLngPath, String endLat, String endLng)** 计算指定起终点之前的距离
- **manhattonDistance (String startLatPath, String startLngPath, String endLat, String endLng)** 计算指定起终点之前的曼哈顿距离

- **join(String... result)**使用##分隔符将多个result聚合到一起(result可有多个)，例如：PathUtils.join(.store.books[0].category,.store.books[1].category) 结果为：reference##fiction
- **joinBy(String split,String... result)**使用指定分隔符将多个result聚合到一起(result可有多个)，例如：PathUtils.join('|',.store.books[0].category,.store.books[1].category) 结果为：reference|fiction
- **joinAsList(String... result)**将多个result聚合到一起成一个数组(result可有多个)，例如：PathUtils.joinAsList(.store.books[0].category,.store.books[1].category) 结果为：[reference, fiction]
- **isPointInBoundary(String latPath, String lngPath,double southLat,double westLng,double northLat,double eastLng)** 判断指定的坐标latPath,lngPath是否在矩形框中，成功返回true，否则返回false，例如：PathUtils.isPointInBoundary(.lat,.lng,39.89954438,116.40047578,39.89896822,116.40152721)

OperationUtils， 支持的方法：

- **sub(String... result)** 返回指定多个结果表达式内容相减的结果，返回结果为double类型
- **plus(String... result)**返回指定多个结果表达式内容相加的结果，返回结果为double类型
- **mul(String... result)**返回指定多个结果表达式内容相乘的结果，返回结果为double类型
- **div(String... result)**返回指定多个结果表达式内容相除的结果，返回结果为double类型
- **mod(String path,String path)**返回指定两个结果表达式内容取余的结果，返回结果为double类型
- **abs(String path)**计算绝对值，返回结果为double类型
- **mulInt(String... result)**返回指定多个结果表达式内容相乘的结果，返回结果为int类型
- **divInt(String... result)**返回指定多个结果表达式内容相除的结果，返回结果为int类型
- **modInt(String path,String path)**返回指定两个结果表达式内容取余的结果，返回结果为int类型
- **subInt(String... result)**返回指定多个结果表达式内容相减的结果，返回结果为int类型
- **plusInt(String... result)**返回指定多个结果表达式内容相加的结果，返回结果为int类型
- **absInt(String path)**计算绝对值，返回结果为int类型

LangUtils， 支持的方法：

- **toCFAbsoluteTimeSecond(String dateString)** 返回cfabsolutetime， 值格式：yyyy-MM-dd HH:mm:ss，yyyy-MM-dd HH:mm，HH:mm:ss，HH:mm，EEE HH:mm（Thursday 08:00），EEE HH:mm:ss（Thursday 08:00:00 or 星期三 08:00:00）
- **toTimeMillis(String dateString)** 返回dateString指定的毫秒数， 值格式：yyyy-MM-dd HH:mm:ss，yyyy-MM-dd HH:mm，HH:mm:ss，HH:mm，EEE HH:mm（Thursday 08:00），EEE HH:mm:ss（Thursday 08:00:00 or 星期三 08:00:00）
- **toTimeSecond(String dateString)** 返回dateString指定的秒数， 值格式：yyyy-MM-dd HH:mm:ss，yyyy-MM-dd HH:mm，HH:mm:ss，HH:mm，EEE HH:mm（Thursday 08:00），EEE HH:mm:ss（Thursday 08:00:00 or 星期三 08:00:00）
- **toInt(String value)** 将结果转换成int，如果传入的是double，那么会丢失精度
- **currentTimeMillis()** 当前时间（毫秒）
- **currentTimeSeconds()** 当前时间（秒）
- **currentCFAbsoluteTimeSecond()** 当前cfabsolutetime

注：path必须为正确的路径表达式，如：.book.title 或者[book][title]

结果表达式

结果表达式是以 . 或者 [开头的字符串，例如：.status 是一种表达式，通过该表达式可以从服务器返回结果中获取所感兴趣的值，具体的语法以及支持的操作类型，可以参考 表达式

示例

假设服务返回的JSON如下：

```
{

    "expensive": 10,
        "title":"books list",
    "store": {
        "book": [
            {
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price": 8.95
            },
            {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour",
                "price": 12.99
            },
            {
                "category": "fiction",
                "author": "Herman Melville",
                "title": "Moby Dick",
                "isbn": "0-553-21311-3",
                "price": 8.99
            },
            {
                "category": "fiction",
                "author": "J. R. R. Tolkien",
                "title": "The Lord of the Rings",
                "isbn": "0-395-19395-8",
                "price": 22.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    }
}
```

若要判断根节点的内容，或者需要对根节点的匹配数进行明确定义，则需要以大括号开始，例如：

```
{
        match_count >= 1
        .expensive = "10"
        with .store.book{
                match_count = 1
                .title=”Sword of Honour"
        }
}
或都
{
        match_count = 3
        with .store.book{
                match_count = 1
                .category=”fiction"
        }
}
```

若不需要判断根节点且使用默认的匹配数（>=1），则不需要以大括号开始，例如：

```
with .store.book{
        .category = "reference"
    .author = "Nigel Rees",
    .title = "Sayings of the Century",
    .price = 8.95
}
```

定时任务

当需要周期性的执行一些任务，或者定时任务时，比如：线上服务监控、线上效果监控等，可使用定时任务。并可配置邮件通知相关人员。

定时任务配置，及结果有以下几个关注点：

1. 定时任务状态
    - NORMAL 任务运行正常
    - PAUSED 任务被暂停
    - COMPLETE 任务执行完成
    - ERROR 任务执行失败
    - BLOCKED 任务被阻塞
2. 发送邮件的条件
    - ALWAYS 无论执行结果是否通过都发送邮件通知
    - NEVER 永远不发送邮件通知
    - WHEN_HAS_FAILED 当执行结果不是百分之百通过时，发送邮件通知
    - LESS_THSN 当通过率低一定的阀值时，发送邮件通知
3. 定时任务表达式
   请参考 定时任务表达式 查看详细的语法以及配置方式。

服务对比

当针对一些不太好定制测试的用例服务，比如：导航，根据交通状态每次返回的路线都会有所区别。可以采用直接对比返回结果报文的形式。 现在支持两种对比方式：

整体内容对比

直接对比服务返回的内容(转换成json之后)是否一致。对比的时候，不关注顺序，例如：假设有结果为，

```
{"value":[{"name":"abc"},{"name":"bcd"},{"name":"code"}]}

{"value":[{"name":"code"},{"name":"abc"},{"name":"bcd"}]}
```

对比结果会认为两者一致

在对比之前，可指定忽略不参与对比的字段/消息体（例如：时间戳等每次返回都不一致的字段）。

格式：

```
jsonpath flag 'ignore'
```

示例：

比对返回结果的所有报文

```
__name__ = "Beijing Temple Restaurant Beijing 北京市东城区东华门大街95号, 100006"
__tag__ = tier2
__geocoding.country__ = "中国"
__geocoding.administrativeArea__ = "北京市"
__geocoding.subLocality__ = "东城区"
__geocoding.thoroughfare__ = "东华门大街"
__geocoding.subThoroughfare__ = "95号"
__geocoding.fullThoroughfare__ = "东华门大街 95号"
__display_language__ = "zh-Hans_CN"
__component__ = "COMPONENT_TYPE_PLACE_INFO"
__component__ = "COMPONENT_TYPE_ENTITY"
__component__ = "COMPONENT_TYPE_BOUNDS"
__component__ = "COMPONENT_TYPE_ADDRESS"
__geocoding.viewport_info.map_region__ = "26.4358, 86.1186,40.131,116.778"
__request_type__ = 'REQUEST_TYPE_GEOCODING'
{
}
```

比对的时候，忽略对所有timestamp字段的对比

```
__name__ = "Beijing Temple Restaurant Beijing 北京市东城区东华门大街95号，100006"
__tag__ = tier2
__geocoding.country__ = "中国"
__geocoding.administrativeArea__ = "北京市"
__geocoding.subLocality__ = "东城区"
__geocoding.thoroughfare__ = "东华门大街"
__geocoding.subThoroughfare__ = "95号"
__geocoding.fullThoroughfare__ = "东华门大街 95号"
__display_language__ = "zh-Hans_CN"
__component__ = "COMPONENT_TYPE_PLACE_INFO"
__component__ = "COMPONENT_TYPE_ENTITY"
__component__ = "COMPONENT_TYPE_BOUNDS"
__component__ = "COMPONENT_TYPE_ADDRESS"
__geocoding.viewport_info.map_region__ = "26.4358, 86.1186,40.131,116.778"
__request_type__ = 'REQUEST_TYPE_GEOCODING'
{
    ..timestamp flag ignore
}
```

指定字段对比

对比的时候，只指定对比特定字段/消息体，在这种情况下，预期结果需要以 _compare 做为前缀。在这种对比模式下，支持所有的函数、变量声明。

示例：

```
__name__ = "Beijing Temple Restaurant Beijing 北京市东城区东华门大街95号，100006"
__tag__ = tier2
__geocoding.country__ = "中国"
__geocoding.administrativeArea__ = "北京市"
__geocoding.subLocality__ = "东城区"
__geocoding.thoroughfare__ = "东华门大街"
__geocoding.subThoroughfare__ = "95号"
__geocoding.fullThoroughfare__ = "东华门大街 95号"
__display_language__ = "zh-Hans_CN"
__component__ = "COMPONENT_TYPE_PLACE_INFO"
__component__ = "COMPONENT_TYPE_ENTITY"
__component__ = "COMPONENT_TYPE_BOUNDS"
__component__ = "COMPONENT_TYPE_ADDRESS"
__geocoding.viewport_info.map_region__ = "26.4358, 86.1186,40.131,116.778"
__request_type__ = 'REQUEST_TYPE_GEOCODING'
{
    .place_result[0].component[0] = '$(_compare.place_result[0].component[0])'
}
```

示例：

实际结果与预期结果的导航时间差值不能超过10%：

```
__name__ = "路径规划_长距离-起终点1000KM以内"
__orgin.searchString__ = "方恒国际中心A座"
__destination.searchString__ = "广东广州"
__mainTransportType__ = "AUTOMOBILE"
{
    .status = "STATUS_SUCCESS"
    diff := "$(OperationUtils.sub(.route[0].expectedTime ,
_compare.route[0].expectedTime))"
    OperationUtils.abs(OperationUtils.div(${diff} ,
_compare.route[0].expectedTime)) < 0.1
}
```

参考↪

[定时任务表达式]

[结果表达式]

[字段配置]

[服务参数列表]