**NetworkBench**

新透镜移动应用监测系统

数据交换协议SPEC

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **备注** | **人员** |
| 1.0 | 2013-09-03 |  |  |
| 2.0.0 | 2014-03-13 |  |  |
| 2.1.0 | 2014-12-25 | 新版Crash格式 |  |
| 2.1.1 | 2015-03-09 | 增加Crash交互轨迹 |  |
| 2.1.2 | 2015-04-21 | actions及错误追踪中的URL增加请求方法字段(GET/POST/PUT/DELETE等) |  |
| 2.1.3 | 2015-06-30 | 1. networkPerfMetrics增加LIB\_NAME 2. 支持混合式App（Webview） |  |

2013年9月

Changelog:

v2.1.0

2015/1/13 initMobileAp，删除app节点中的SDK\_VERSION项。（廖雄杰）

2015/1/13 initMobileApp， app节点中增加TINGYUN\_BUILD\_ID（听云Build ID）。（廖雄杰）

2015/1/13 增加4.6 上传崩溃日志（新版Crash消息格式）。（廖雄杰）

# 引言

## 编写目的

为研发的系统设计过程提供标准的数据交换格式。

## 参考标准

JSON（JavaScript Object Notation）：<http://www.json.org/>

HTTP协议：<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

# 交换数据格式

## 目标

以精简、高效的格式完成Agent与服务器的数据交换，交换格式遵循以下原则：

* 数据格式尽可能精简，以尽可能小的网络带宽占用完成数据交换
* 较好的编码/解码效率
* 适当的可读性

## 交换格式约定及设计原则

### 统一采用JSON格式交换

数据交换过程中统一采用JSON（<http://www.json.org/>）格式进行交换，如：

{

“k\_num”:1.23456,

“k\_str”:”str\_value”,

“k\_arr”:[10,20,30,40],

“k\_obj”: {

“k1” : “v1”,

“k2” : “v2”

}

}

注意：JSON的Spec规定，KEY部分统一采用string的表达方式，必须使用双引号（”）包围，如下面的例子是不标准的：

{

k1:”v1”, *// 应写成 “k1”*

k2:”v2” *// 应写成 “k2”*

}

### 数据兼容性，避免二义性

VALUE的部分务必严格按照数值的类型采用正确的写法，确保数据的兼容性，避免二义性。

对于数值型，应是用数值型，而非string型的表示方式，如：perf=10.382，应写成：

{

“perf” : 10.382

}

而不是：

{

“perf” : ”10.382”

}

如果是整数型，则尽量采用整数，而非浮点数表示，如：count=100，应写成

{

“count” : 100

}

而不是：

{

“count” : 100.000

}

### 压缩数据格式以达到精简的要求

对于数量比较大的结构化数据，应尽可能考虑压缩数据格式，以达到精简的要求，如：

{

“data”: [

{“k1”:”v11”,“k2”:”v21”,“k3”:”v31”},

{“k1”:”v12”,“k2”:”v22”,“k3”:”v32”},

{“k1”:”v13”,“k2”:”v23”,“k3”:”v33”},

{“k1”:”v14”,“k2”:”v24”,“k3”:”v34”}

]

}

可以精简为：

{

“data”: [

[”v11”,”v21”,”v31”],

[”v12”,”v22”,”v32”],

[”v13”,”v23”,”v33”],

[”v14”,”v24”,”v34”]

]

}

注意：以这种精简格式输出时，务必交换时约定并确保精简后[”v11”,”v21”,”v31”]内元素的顺序一致，避免数据的二义性。

### 去除不必要的空白字符

输出的数据要求去除不必要的换行、空白字符，如：

{

“data”: [

[”v11”, ”v21”, ”v31”],

[”v12”, ”v22”, ”v32”],

[”v13”, ”v23”, ”v33”],

[”v14”, ”v24”, ”v34”]

]

}

可以精简为：

{“data”:[[”v11”,”v21”,”v31”],[”v12”,”v22”,”v32”],[”v13”,”v23”,”v33”],[”v14”,”v24”,”v34”]]}

这样虽然损失了一定的可读性，但通常这些数据不需要人去阅读，必要的时候通过专门的格式化工具重新格式化即可。

### 数值精度

对于时间性能类型的指标，除特殊说明外，一律以毫秒为单位，并取整输出（精确到毫秒）。

### 兼容性

对于K-V格式的数据，多组K-V无顺序要求，如下面2组数据应视为相等：

{“K1” : “V1”, “K2” : “V2”}

{“K2” : “V2”, “K1” : “V1”}

### 本文档书写格式约定

* 接口定义中，*斜体全大写单词*代表变量，字符串变量将用双引号（”）包围，否则，代表数值或对象类型，对于对象类型，后面会紧跟着有格式描述，使用 := 代表格式描述符。如：

{

“hostName” : “*HOST\_NAME*”,

“errors” : *ERRORS\_ARRAY*

}

*ERRORS\_ARRAY* :=

[

*ERROR\_ITEM* \*

]

* 红色字体星号（\*）代表0或多个，通常用于表示数组的元素，如上例中的*ERROR\_ITEM \**
* 枚举值使用竖线( | )分隔，如 “status” : “success | error”，”enabled” : true | false；..表示数值范围枚举值，如1 .. 10

# 数据交互

## 传输方式

交换数据采用标准HTTP(S) 方式传输（缺省https），请求端（Agent端）发送POST请求，接收端（服务器端）返回响应结果，数据经编码后以二进制置于Request Body中发送，请求及响应格式如下：

*Request：*

POST /*DATA\_COLLECTOR\_URI* HTTP/1.1

Host: *DATA\_COLLECTOR\_HOST*

Content-Encoding: *CONTENT\_ENCODING*

Content-Type: *CONTENT\_TYPE*

Accept-Encoding: *ACCEPT\_ENCODINGS*

[MORE REQUEST HEADERS …]

entity-body = \**OCTET*

*Response：*

HTTP/1.1 200 OK

Content-Encoding: *CONTENT\_ENCODING*

[MORE RESPONSE HEADERS …]

entity-body = \**OCTET*

其中：

entity-body := Content-Encoding(Content-Type(data ))

## 数据编码与压缩

除非特别说明，请求端数据必须采用deflate（亦即gzip或zlib）算法对数据进行压缩编码后传输，同时请求头中指定

Content-Encoding: *CONTENT\_ENCODING*

及

Content-Type: application/octet-stream

若请求端支持接收压缩编码的响应内ed容，应明确指定Accept-Encoding，多个encoding以逗号(,)分隔，服务器端应至少支持以下2种标准encoding: gzip, deflate，客户端根据Content-Encoding响应头判断响应的编码方式，如：

Request Headers:

Accept-Encoding: gzip, deflate

Content-Encoding: deflate

Response Headers:

Content-Encoding: gzip

若请求端未指定Accept-Encoding头或指定未约定的编码，则服务器端必须输出未经压缩的响应内容，同时将响应头Content-Encoding置为plain

## Keep-Alive

~~Agent端建议设置Connection头为Keep-Alive，并重用HTTP连接：~~

~~Connection: Keep-Alive~~

注：从服务器的角度考虑，大量并发长连接可能耗用大量服务器资源，暂时不考虑Keep-Alive

## HTTP头

* **User-Agent**

与Newlens服务器交互时使用以下格式User-Agent（其它请求仍然使用系统缺省User Agent）。

User-Agent格式为NBS Newlens Agent/*AGENT\_VERSION* (*OS\_NAME OS\_VERSION*)

如：

User-Agent: NBS Newlens Agent/1.0.0 (iOS 6.1.2)

* **X-License-Key**

与Newlens服务器交互过程中，请求头必须包含X-Lisence-Key头，服务器接收数据前先校验LicenseKey

* **X-App-Sign**

App签名串。此参数用于标识App。若服务器端开启签名校验，则将使用此签名串校验数据的有效性，防止用户数据受到恶意攻击。

initMobileApp请求中必须上传**X-App-Sign**头以便服务器端对App做签名校验。

## 响应及错误处理

正常响应数据格式为（响应码为200）：

{

“status” : “success”,

“result” : SUCCESS\_RESULT

}

错误时响应数据格式为：

{

“status”: “error”,

“result”:*RESULT\_ERROR*

}

*RESULT\_ERROR* :=

{

“errorCode”: ERROR\_CODE,

“errorMessage”: “ERROR\_MESSAGE”

}

**错误码(errorCode)及处理方式**

|  |  |
| --- | --- |
| 错误码 | 描述 |
| 460 | 无效的License或App token或App签名。Agent端检测到该错误应立即停止所有嵌码采集及上传动作。涉及接口：全部。  特别注意：探针检测到该错误，仅停止本次会话的嵌码采集动作，App再次启动，应启动嵌码采集并重新检测。防止服务器端异常响应的情况下，可以加强容错。 |
| 461 | 无效的数据验证码（Data token）。Agent端检测到该错误应重新发起initMobileApp动作，重新请求一个Data token。涉及接口：uploadMobileData、mobileDiagnosticsData |
| 462 | 无效的数据。可能数据格式错误，例如，无效的Json，Agent端丢弃处理即可。涉及接口：uploadMobileData、mobileDiagnosticsData |
| 463 | 无效的设备ID。可能由于设备ID所关联的设备信息被服务器端标识为已失效，Agent端检测到该错误重新发送initMobileApp请求，服务器会可能生成新的设备ID。initMobileApp请求中应携带原did参数，若服务器返回新设备ID，Agent端应更新本地设备ID。涉及接口：initMobileApp |
| 470 | 配置已过期。服务器在收到Agent的上传请求时，可能检测配置是否有更新，并返回该错误码，Agent端应重新发起initMobileApp动作，更新配置及token，然后重新尝试上传。涉及接口：uploadMobileData |
| -1 | 其它错误。可能由于服务器的问题导致，Agent端无需处理该错误，丢弃即可。 |

错误处理方式说明：

服务器端:

* 1. 下发460错误时应谨慎，所有不确定的情况（如校验过程中出现异常），均不应下发460错误

探针端：

探针对错误及异常进行处理时（包括上述定义的错误代码以及网络异常），应增加适当的容错逻辑

* 1. 任何情况下都不能出现永久禁用探针的动作（包括460-无效的License）
  2. 错误或异常情况下如果需要重试，尽可能间隔一定的周期（如1分钟）后再重试；如果有必要立即重试，需要增加最多重试次数（如3次）的限制。避免异常情况下，因重试对服务器造成持续压力。

# Agent-服务器数据交互接口

## 交互流程

这里所说“客户端”，是指移动应用探针（其他称呼：透镜移动应用SDK、SDK、NBSAgent，都是指同一个东西）。

客户端要向服务器报告数据，以反应app的性能。

有两个问题需要解决：1、能反映网络性能的主体数据怎样定义；2、如何区分数据来自哪个设备、哪个app、哪次运行。

## 获取后端服务器地址（getMobileRedirectHost）

**URL:**

http(s)://*REDIRECT\_HOST*/getMobileRedirectHost

**参数说明：**

**请求数据：**

{

}

**响应：**

{

“status” : “success”,

“result” : “*DC\_HOST*”

}

## 初始化应用实例（initMobileApp）

**URL:**

http(s)://*DC\_HOST*/initMobileApp?version=*DATA\_VERSION*

**HTTP头：**

**X-App-Sign**

App签名串。此参数用于标识App。若服务器端开启签名校验，则将使用此签名串校验数据的有效性，防止用户数据受到恶意攻击。

**请求数据：**

{

“did” : “*DEVICE\_ID*”,

“dev” : *DEVICE\_INFO* ?,

“app” : *APP\_INFO* ?

}

其中，

*DEVICE\_INFO* :=

[

*DEVICE\_TYPE* | *“DEVICE\_TEXT”*,

“*MANUFACURER*”,

“*MANUFACURER\_MODEL*”,

“*OS\_NAME*”,

“*OS\_VERSION*”,

*“AGENT\_NAME”*,

*“AGENT\_VERSION”*,

*MISC*

]

*DEVICE\_ID*: 设备ID。第一次置空，由服务器生成，并保存至本地。服务器端应将设备ID作为设备/应用的唯一关联，若设备或应用信息发生改变，客户端应将did置空，请求服务器端重新生成设备ID。

*DEVICE\_TYPE*: 设备类型。（0 – 未知 1 – 手机 2 - Touch 3 – Pad 4 – TV 5 – 机顶盒 9 – 其它）

*DEVICE\_TEXT*: 设备信息。（Andoid设备较杂，直接上传DEVICE信息，由服务器端识别设备类型）

*MANUFACURER*: 设备生产厂家（如Apple, Sumsang等）。

*MANUFACURER\_MODEL*: 设备型号。

*OS\_NAME*: 操作系统名称。

*OS\_VERSION*: 操作系统版本。

*AGENT\_NAME*: 听云探针名称（agent-ios | agent-android）

*AGENT\_VERSION*:听云探针版本号。

*MISC* := {

“size” : *DEVICE\_SIZE,*

“im” :“*IMEI*”*,*

“mac” :“*MAC\_ADDRESS*”,

“aid” :“*ANDROID\_ID*”,

*……*

}

*DEVICE\_SIZE*: 设备尺寸（0 – Unknown 1 – Small 2 – Normal 3 – Large 4 – XLarge）。

*IMEI*: 设备的IMEI号。用于生成did。对于iOS，可能无法获取该参数，无需上传。

*MAC\_ADDRESS*: 设备MAC地址。用于生成did。

*ANDROID\_ID*: Android ID。用于生成did。

*APP\_INFO* :=

[

“*BUNDLE\_ID*”,

“*APP\_NAME*”,

“*APP\_VERSION*”,

*“TINGYUN\_BUILD\_ID”*

]

*BUNDLE\_ID*: App bundle ID（iOS）或App包名（Android）

*APP\_NAME*: App名称（发布名称或显示名称）。

*APP\_VERSION*: 应用版本

*TINGYUN\_BUILD\_ID*: 听云Build ID。由听云探针在构建阶段生成，用于识别唯一一次构建。*TINGYUN\_BUILD\_ID*有别于APP\_VERSION，同一个App版本可能对应多个*TINGYUN\_BUILD\_ID*（例如，Android下可能同一个版本的App通过多个渠道发布，需要通过*TINGYUN\_BUILD\_ID*识别）。iOS使用Image文件对应UUID。

dev和app均为可选项，仅当第一次启动或设备/应用信息发生变化时上传，否则，仅上传did（设备ID），服务器端可自动根据设备ID自动关联设备/应用详细信息。

此操作为可选操作，若本地已保存上次的did及token，则

**响应数据：**

{

“status” : “success”,

“result” : *RESULT\_SUCCESS*

}

或

{

“status”: “error”,

“result”:*RESULT\_ERROR*

}

其中，

*RESULT\_SUCCESS* :=

{

“token”: *DATA\_TOKEN*,

“did”: *DEVICE\_ID*,

“enabled”: *APP\_ENABLED*,

“disabledTimeout”: *APP\_DISABLED\_TIMEOUT*,

“cfg” : {

“interval”: *DATA\_SENT\_INTERVAL\_IN\_SEC*,

“intervalOnIdle”: *DATA\_SENT\_INTERVAL\_ON\_IDLE\_IN\_SEC*,

“actions” : *MAX\_ACTIONS\_COUNT*,

“actionAge” : *MAX\_ACTION\_AGE\_IN\_SEC*,

“features”: *ENABLED\_FEATURES*,

“enableErrTrace” : *ENABLE\_ERROR\_TRACES*,

“errs” : *MAX\_ERRORS\_COUNT*,

“errRspSize” : *MAX\_ERROR\_RESPONSE\_BODY\_SIZE*,

“stackDepth” : *MAX\_STACKTRACE\_DEPTH*,

“dnsconn” : *COLLECT\_DNS\_AND\_CONNECT\_TIME*,

“urlFilterMode”: *URL\_FILTER\_MODE,*

“urlRules”*: URL\_FILTER\_RULES*,

*“*ignoredErrRules*”: IGNORED\_HTTP\_ERROR\_RULES*,

“urlParams”*:* *URL\_PARAMS*,

*“*uiTraceThreshold*”:*

*USER\_INTERACTION\_TRACE\_THRESHOLD\_MILLISEC*,

*“*uiTraceSize*”:*

*USER\_INTERACTION\_TRACE\_MAX\_SIZE*,

*“*uiTraceRetries*”:*

*USER\_INTERACTION\_TRACE\_UPLOAD\_MAX\_RETRIES*,

*“*uiTraces*”:*

*USER\_INTERACTION\_TRACE\_UPLOAD\_MAX\_COUNT*,

*“*crashTrails*”:*

*CRASH\_TRAIL\_NODES\_MAX\_COUNT*

}

}

*DATA\_TOKEN*: 数据token。后续与服务器的交互使用该token进行识别。Agent端初始化成功后应将该token与did一起保存。

*APP\_ENABLED:* App是否启用。0 – 禁用 1- 启用。若禁用，则Agent端不采集及上传任何数据。

*APP\_DISABLED\_TIMEOUT*: App禁用超时时长，单位：分钟。仅当enabled属性为0（禁用）时有效。若服务器返回禁用标识且设置了App禁用超时（如服务器按月活采样时可能返回该值），则探针不但本次会话不启用探针，下次App再启动时，无需发送initMobileApp请求，可立即自行禁用探针，直到距离上次initMobileApp操作超过*APP\_DISABLED\_TIMEOUT*分钟。缺省值：0，表示不使用超时模式（跟返回结果Map中无disabledTimeout参数是等价的，即每次启动App均向dc服务器发送initMobileApp请求请求配置）。

*注：设置这个参数主要是为了服务器按月活采样的场景下，可能某个App实际月活较大，根据合同月活限制的要求，部分探针因采样被临时禁用，在采样的场景下，通常同一设一部分设备如果被采样规则设置为临时禁用，基本上一段时间内都是禁用状态，没必要每次启动都发起initMobileApp请求，给服务器增加额外的压力。*

*DATA\_SENT\_INTERVAL\_IN\_SEC*: 上传数据间隔（单位：秒），默认值：60

*DATA\_SENT\_INTERVAL\_ON\_IDLE\_IN\_SEC*: 空闲（无HTTP访问）时上传数据间隔（单位：秒），默认值：20

*MAX\_ACTIONS\_COUNT*: 当前周期中最多上传的action数量，默认值：1000

*MAX\_ACTION\_AGE\_IN\_SEC*: action 最大保留时间（单位：秒），默认值：600

*ENABLED\_FEATURES*: 已启用的采集模块位值(0~255)。各模块对应的位值如下：

1 (二进制：1) – HTTP网络采集模块

2 (二进制：10) – 用户交互采集模块

4 (二进制：100) - 崩溃采集模块

8 (二进制：1000) - Hydrid/WebView模块

如，若开启HTTP网络采集模块及崩溃模块，则*ENABLED\_FEATURES*=1 | 4 = 5

*ENABLE\_ERROR\_TRACES*: 是否收集错误trace，默认值：true。此选项依赖于*ENABLE\_NETWORK\_FEATURE*选项，若HTTP网络采集模块为禁用状态，则错误收集自动被禁用。

*MAX\_ERRORS\_COUNT*: 内存中最多保留错误trace的个数，默认值：100

*MAX\_ERROR\_RESPONSE\_BODY\_SIZE*: 错误trace的响应body大小（单位：B），默认值：2048

*MAX\_STACKTRACE\_DEPTH*: 错误stacktrace的最大深度，默认值：10

*COLLECT\_DNS\_AND\_CONNECT\_TIME*: 是否手机DNS和建连时间，默认值：false

*URL\_FILTER\_MODE*: URL过滤模式。

0 – 无过滤。

1 – 白名单。仅采集白名单规则匹配到的URL。

2 – 黑名单。忽略黑名单规则匹配到的URL。

*URL\_FILTER\_RULES*: URL过滤规则

*URL\_FILTER\_RULES* :=

[

*URL\_FILTER\_RULE\_ITEM \**

]

*URL\_FILTER\_RULE\_ITEM*  :=

{

“matchMode”: *URL\_RULE\_MATCH\_MODE*,

“rule”:*URL\_RULE*

}

*URL\_RULE\_MATCH\_MODE*: URL规则匹配模式。0 – 普通文本匹配 1 – 正则表达式匹配。两种匹配模式均为包含模式匹配（只要URL包含即可，不需要全文匹配）。

*URL\_RULE*: 需要匹配的URL规则（URL文本或正则表达式）。匹配模式为包含，即只要URL部分匹配即可。如<http://www.networkbench.com/1234.html，若配置正则表达式>: *networkbench\.com\/[0-9]+*，匹配结果为true（和 .\**networkbench\.com\/[0-9]+.\* 等效*）。

*IGNORED\_HTTP\_ERROR\_RULES*: HTTP错误过滤规则列表（符合规则的URL发生相应的HTTP错误时，视为正常数据。）

*IGNORED\_HTTP\_ERROR\_RULES* :=

[

*IGNORED\_HTTP\_ERROR\_RULE\_ITEM \**

]

*IGNORED\_HTTP\_ERROR\_RULE\_ITEM*  :=

{

“matchMode”: *URL\_RULE\_MATCH\_MODE*,

“rule”:*URL\_RULE*,

“errs”: “*IGNORED\_HTTP\_STATUS\_CODES*”

}

*URL\_PARAMS*: URL需要捕捉的参数列表（每一个URL对应一组参数规则），格式定义如下：

*URL\_PARAMS* :=

[

*URL\_PARAM\_ITEM \**

]

*URL\_PARAM\_ITEM* :=

{

“url”: “*URL\_TO\_CAPTURE*”,

“get”:“*URL\_GET\_PARAM\_NAMES*”,

“post”:“*URL\_POST\_PARAM\_NAMES*”,

“headers”:“*URL\_HEADER\_NAMES*”

}

*URL\_RULE\_MATCH\_MODE*: URL规则匹配模式。0 – 普通文本匹配 1 – 正则表达式匹配。两种匹配模式均为包含模式匹配（只要URL包含即可，不需要全文匹配）。

*URL\_RULE*: 需要匹配的URL规则（URL文本或正则表达式）。匹配模式为包含，即只要URL部分匹配即可。如<http://www.networkbench.com/1234.html，若配置正则表达式>: *networkbench\.com\/[0-9]+*，匹配结果为true（和 .\**networkbench\.com\/[0-9]+.\* 等效*）。

*IGNORED\_HTTP\_STATUS\_CODES*: 忽略HTTP状态码列表，多个状态码逗号分隔。

*URL\_GET\_PARAM\_NAMES*: 需要捕捉的URL中问号之后的参数列表，多个参数名以逗号分隔。Agent端应根据App配置捕捉相应的URL参数，并随uploadMobileData上传（参见第4.4节 上传探测数据一节）。若无配置，则返回空串(“”)。

*URL\_POST\_PARAM\_NAMES*: 需要捕捉的POST参数列表，多个参数名以逗号分隔，只对POST请求生效，对GET请求不需要检查该规则。Agent端应根据App配置捕捉相应的POST参数，并随uploadMobileData上传（参见第4.4节 上传探测数据一节）。若无配置，则返回空串(“”)。

*URL\_HEADER\_NAMES*: 需要捕捉的HTTP头列表，多个HTTP头名称以逗号分隔。Agent端应根据App配置捕捉响应的HTTP头，并随uploadMobileData上传（参见第4.4节 上传探测数据一节）。若无配置，则返回空串(“”)。

*USER\_INTERACTION\_TRACE\_THRESHOLD\_MILLISEC*:

用户交互追踪执行时间阈值，当一个交互的时间（Android下的Activity执行时间或iOS下的ViewController执行时间）大于该阈值时记录并提交详细的交互追踪数据。单位：毫秒。缺省值：300

*USER\_INTERACTION\_TRACE\_MAX\_SIZE*:

用户交互追踪数据最大长度，上传的交互追踪结果数据的最大长度，超过该长度时，客户端负责截断数据。缺省值：64KB。

*USER\_INTERACTION\_TRACE\_UPLOAD\_MAX\_RETRIES*:

用户交互追踪数据最多尝试提交次数，尝试上传交互追踪结果数据的次数，包含第首次次上传。当上传失败次数大于等于该值时丢弃交互跟踪数据不再上传。缺省值：1

*USER\_INTERACTION\_TRACE\_UPLOAD\_MAX\_COUNT*:

用户交互追踪数据采集数量，每个数据采集周期记录的交互追踪数据最多条数，超过该数量则丢弃后面的数据。缺省值：1

*CRASH\_TRAIL\_NODES\_MAX\_COUNT*:

Crash交互轨迹节点最大数量限制，超过限制则丢弃交互轨迹中发生时间较早的节点。缺省值：20。特殊值，0表示无限制；-1表示禁用交互轨迹采集。

*RESULT\_ERROR* :=

{

“errorCode”: ERROR\_CODE,

“errorMessage”: “ERROR\_MESSAGE”

}

ERROR\_CODE: 错误代码。

463 – 无效的设备ID （客户端清空did应重新发送initMobileApp请求，服务器会生成新的设备ID）

## 上传探测数据（uploadMobileData）

**URL:**

http(s)://*DC\_HOST*/uploadMobileData?version=*DATA\_VERSION*&token=*DATA\_TOKEN*

**HEAD:**

licenseKey – 许可证Key

**参数说明：**

version – 数据协议版本

token – 当前会话数据token

**请求数据：**

[

*UPLOAD\_DATA\_ITEM \**

]

其中，

*UPLOAD\_DATA\_ITEM* :=

{

“type” : “*DATA\_TYPE*”,

... (其它上传内容参见 4.5.2 ~ 4.5.3各数据类型上传格式)

}

注：服务器端实现支持Agent端将需要上传的不同类型(由type标识)的数据打包到一起或任意组合后打包上传，对于移动Agent，考虑到交互追踪数据可能较大，建议“用户交互追踪数据”单独上传，而将网络请求性能指标数据和用户交互性能指标数据一起上传。

若客户端处在空闲状态（既无HTTP请求，也无用户交互发生），则发送空Json请求：

{} // 兼容版本1.0的格式

或者：

[{}] // 版本2.0的格式

注意：

发送空数组

[]

是非法的。

**响应：**

{

“status”: “success | error”,

“result” : *RESP\_RESULT\_SUCCESS* | *RESP\_RESULT\_ERROR*

}

status: 状态。

success – 服务器正常处理。

error – 返回此状态表示服务器未成功接收，Agent端需根据result中的errorCode做相应的处理，并重新上传；若重新上传后仍然返回error，则放弃本次上传操作。

其中，

*RESP\_RESULT\_SUCCESS*: 参见4.5.2 ~ 4.5.3各数据类型的返回值。

*RESP\_RESULT\_ERROR* :=

{

“errorCode” : *ERROR\_CODE*,

“errorMessage” : “*ERROR\_MESSAGE*”

}

错误代码(errorCode)：

101 – 本地配置已过期，需从服务器更新配置

1. 探测数据类型

探测数据类型及接口中type值定义如下：

* 网络请求性能指标数据 type: networkPerfMetrics
* 用户交互性能指标数据 type: uiPerfMetrics
* 用户交互追踪数据 type: uiTraceData
* Webview性能指标数据 type: webviewPerfMetrics

1. 网络请求性能指标数据 - networkPerfMetrics

**URL:**

http(s)://*REDIRECT\_HOST*/uploadMobileData?version=*DATA\_VERSION*&token=*DATA\_TOKEN*

**HEAD:**

licenseKey – 许可证Key

**请求数据：**

{

“type” : “networkPerfMetrics”,

“interval” : *DATA\_SENT\_INTERVAL\_IN\_SECONDS*,

“dev” : *DEVICE\_DATA*,

“actions” : *ACTIONS\_DATA\_ARRAY*,

“metrics” : *METRIC\_DATA*,

“errs” : *ERROR\_DATA\_ARRAY*

}

其中，

*DATA\_TOKEN* : 上传数据token

*DEVICE\_DATA* :=

[

“*CARRIER*”,

*CONNECT\_TYPE*,

“*NETWORK\_*TYPE”,

*GPS\_LATITUDE*,

*GPS\_LONGITUDE*

]

*CARRIER* : 运营商代码，3位MCC + 2位MNC编码。如：

46000 – 中国移动

46001 – 中国联通

46003 – 中国联通CDMA

46005 – 中国电信

WIFI下该项需置空。

*CONNECT\_TYPE*: 网络连接类型（0 – 未知或无网络连接 1 – WIFI 2 – 2G 3 – 3G 4 – 4G）

*NETWORK\_TYPE*: 网络类型，Android参见TelephonyManager（GPRS/EDGE/UMTS/CDMA/EVDO\_0/EVDO\_A/1xRTT/HSDPA/HSUPA/HSPA/IDEN/EVDO\_B等），iOS无法获取，置空即可

*GPS\_LATITUDE*: 纬度

*GPS\_LATITUDE*: 经度

*DATA\_SENT\_INTERVAL\_IN\_SECONDS*: 数据上传间隔（单位：秒）

*ACTIONS\_DATA\_ARRAY* :=

[

*ACTION\_DATA\_ITEM* \*

]

*ACTION\_DATA\_ITEM* :=

[

“*URL*”,

“*REQUEST\_URL\_PARAMS*”*,*

*REQUEST\_METHOD,*

*HTTP\_LIB\_NAME*,

“*REMOTE\_IP*”,

*DURATION*,

*TIME\_TO\_DNS*,

*TIME\_TO\_CONNECT*,

*TIME\_TO\_FIRST\_PACKET*,

*HTTP\_STATUS\_CODE*,

*NETWORK\_ERROR\_CODE*,

*BYTES\_SENT*,

*BYTES\_RECEIVED*,

“*APP\_DATA*”

]

*URL*: 请求URL

*REQUEST\_URL\_PARAMS*: 根据App配置捕捉的GET/POST/HEADER参数（参见第4.3节 初始化应用实例）。*REQUEST\_PARAMS*格式为：p1=v1&p2=v2&…。若捕捉的参数同时包含GET、POST及HEADER，则将所有参数键值对拼成上述的格式。

*REQUEST\_METHOD*: 请求的方法:

0 – GET

1 – POST

2 – PUT

3 – DELETE

4 - HEAD

5 - TRACE

6 - OPTIONS

*HTTP\_LIB\_NAME*: 使用HTTP库或框架:

0 – URLConnection（Android或iOS原生HTTP库）

1 – URLSession (iOS)

2 – HttpClient

3 – OkHttp

4 – Webview

5 – Webview AJAX

6 – ASIHTTP

7 – AFNetworking

8 – MKNetworkKit

注：未定义的HTTP库，对于Native http调用归为0 – URLConnection，对于Webview方式的请求，归为 4 – Webview

*REMOTE\_IP*: 目标IP

*DURATION*: 响应时间（单位：毫秒）

*TIME\_TO\_DNS*: DNS时间（单位：毫秒）。若无，缺省值置为-1。

*TIME\_TO\_CONNECT*: 建立连接时间（单位：毫秒）。若无，缺省值置为-1。

*TIME\_TO\_FIRST\_PACKET*: 首包时间（单位：毫秒）

*HTTP\_STATUS\_CODE*: 响应代码。

*NETWORK\_ERROR\_CODE*: 异常代码。

0 - 正常或HTTP错误。（正常或错误状态由*HTTP\_STATUS\_CODE定义*）

900 – MalformedURLException

901 – UnknownHostException

902 – ConnectException

903 – TimeoutException

904 – ClientProtocolException (httpclient)

905 – ClientAbortException

906 – 非法响应内容

907 – 需要认证

908 – SSL证书错误

-1 – 其它未定义的异常

4xx-5xx – 标准响应异常（同response.statusCode）

*BYTES\_SENT*: 发送字节数

*BYTES\_RECEIVED*: 接收字节数

*APP\_DATA*: 应用响应头（ X-Newlens-Application-Id）。若服务器端安装newlens agent，则服务器端嵌入该响应头，自动关联应用及移动端请求。

*METRIC\_DATA* :=

{

“mem” : *MEMORY\_USED\_IN\_MB*,

“cpu\_sys” : *CPU\_UTIL\_SYSTEM*,

“cpu\_user” : *CPU\_UTIL\_USER*,

“cpu” : *CPU\_UTIL\_TOTAL*,

“sd” : *SESSION\_DURATION\_IN\_SEC*,

“req” : *SESSION\_REQUEST\_COUNT*

}

*MEMORY\_USED\_IN\_MB*: 内存使用量（单位：MB）

*CPU\_UTIL\_SYSTEM*: CPU使用率（系统）。百分制取整（0-100）。

*CPU\_UTIL\_USER*: CPU使用率（用户）。百分制取整（0-100）。

*CPU\_UTIL\_TOTAL*: CPU使用率（总计）。百分制取整（0-100）。

*SESSION\_DURATION\_IN\_SEC*: 会话时长（仅对最后一次操作有效）。单位：秒

*SESSION\_REQUEST\_COUNT*: 本次会话请求数（仅对最后一次操作有效）

*ERROR\_DATA\_ARRAY* :=

[

*ERROR\_DATA\_ITEM* \*

]

*ERROR\_DATA\_ITEM* :=

[

“*REQUEST\_URL*”,

“*REQUEST\_URL\_PARAMS*”,

*REQUEST\_METHOD,*

*HTTP\_STATUS\_CODE\_OR\_NETWORK\_ERROR\_CODE*,

*COUNT\_OF\_ERRORS*,

“*ERROR\_PARAMS”*

]

*ERROR\_PARAMS* :=

{

“params” : *CUSTOM\_PARAMS*,

“requestParams” : *ALL\_GET\_REQUEST\_PARAMS*,

“message”: “*ERROR\_MESSAGE\_IF\_ANY”*,

“response” : “*RESPONSE\_BODY”*,

“stacktrace” : “*ERROR\_STACK\_TRACE*”

}

*CUSTOM\_PARAMS* :=

{

“responseHeader” : *RESPONSE\_HEADERS*

}

*RESPONSE\_HEADERS* :=

{

*“HEADER\_NAME\_1” : “HEADER\_VALUE\_1”,*

*“HEADER\_NAME\_2” : “HEADER\_VALUE\_2”*

……

}

*URL*: 请求URL

*REQUEST\_URL\_PARAMS*: 根据App配置捕捉的GET/POST/HEADER参数（参见第4.3节 初始化应用实例）。*REQUEST\_PARAMS*格式为：p1=v1&p2=v2&…。若捕捉的参数同时包含GET、POST及HEADER，则将所有参数键值对拼成上述的格式。

*REQUEST\_METHOD*: 请求的方法:

0 – GET

1 – POST

2 – PUT

3 – DELETE

4 - HEAD

5 - TRACE

6 - OPTIONS

*HTTP\_STATUS\_CODE\_OR\_NETWORK\_ERROR\_CODE*: 错误代码。对于HTTP错误（HTTP status code为4xx或5xx），该字段置为HTTP Status Code；对于网络错误，置为响应的网络错误代码（异常代码，参见actions中异常代码的定义）。

*COUNT\_OF\_ERRORS*: 错误计数。以*REQUEST\_URL* + *REQUEST\_URL\_PARAMS + HTTP\_STATUS\_CODE\_OR\_NETWORK\_ERROR\_CODE*为Key汇总。

注：错误trace数据仅采集网络层Exception或5xx HTTP错误；请求头和响应头最大限制为1KB

关于错误信息的其他说明：对于同一个（URI＋statuscode）只保留第一个错误的具体信息，但要计数。

当是异常发生时，没有statusCode，此时statusCode为0.

**响应数据：**

{

“status” : “success”

}

1. 用户交互性能指标数据 - uiPerfMetrics

性能指标类主要分为以下几类：

* Action（对应IOS的Controller或Android的Activity）的交互性能数据(actions)
* Action break down性能数据，即分解到函数级别的性能数据(components)
* 其它维度性能数据(general)

**请求数据：**

{

“type” : “uiPerfMetrics”,

“timeFrom” : *BEGIN\_TIME\_IN\_SECONDS*,

“timeTo” : *END\_TIME\_IN\_SECONDS*,

“interval” : *DATA\_SENT\_INTERVAL\_IN\_SECONDS*,

“dev” : *DEVICE\_DATA*,

“views” : [*VIEW\_METRIC\_DATA\_ARRAY*](#INTERACTION_METRIC_DATA_ARRAY),

“components” : [*COMPONENT\_METRIC\_DATA\_ARRAY*](#COMPONENT_METRIC_DATA_ARRAY),

“classifiedComponents” : [*COMPONENT\_METRIC\_DATA\_ARRAY*](#COMPONENT_METRIC_DATA_ARRAY),

“general” : [*GENERAL\_METRICS\_DATA\_ARRAY*](#GENERAL_METRICS_DATA_ARRAY)

}

**参数说明：**

type: 探测数据类型（参见 4.5.1 探测数据类型）

timeFrom: 当前上传数据时间段的起始EPOCH时间戳（秒）

timeTo: 当前上传数据时间段的起始EPOCH时间戳（秒）

interval: 数据上传间隔(秒)。若Agent端上传故障，可能做合并处理，服务器端入库时应将该时间段内的数据平均分散存储（SUM及Count）。

views: 视图（对应IOS的Controller或Android的Activity）的汇总性能数据。参见“[NetworkBench New Lens-Spec-Metrics.docx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics.docx)”第6.1节“按视图汇总”。具体指标名定义参见文档：“[NetworkBench New Lens-Spec-Metrics-Scope & categories.xlsx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics-Scope%20&%20categories.xlsx)”之“移动探针指标项”

components: 视图分解到函数级别的汇总性能数据。参见“[NetworkBench New Lens-Spec-Metrics.docx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics.docx)”第6.1节“按视图及调用方法汇总”。具体指标名定义参见文档：“[NetworkBench New Lens-Spec-Metrics-Scope & categories.xlsx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics-Scope%20&%20categories.xlsx)”之“移动探针指标项”

classifiedComponents: 视图分解到7大性能分类的汇总性能数据。参见“[NetworkBench New Lens-Spec-Metrics.docx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics.docx)”第6.1节“按视图及性能分类汇总”。具体指标名定义参见文档：“[NetworkBench New Lens-Spec-Metrics-Scope & categories.xlsx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics-Scope%20&%20categories.xlsx)”之“移动探针指标项”

general: 其它性能汇总数据，如CPU，内存等。具体指标名定义参见文档：“[NetworkBench New Lens-Spec-Metrics-Scope & categories.xlsx](https://192.168.1.42/newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics-Scope%20&%20categories.xlsx)”之“移动探针指标项”

其中，

*DEVICE\_DATA* :=

[

“*CARRIER*”,

*CONNECT\_TYPE*,

“*NETWORK\_*TYPE”,

*GPS\_LATITUDE*,

*GPS\_LONGITUDE*

]

*CARRIER* : 运营商代码，3位MCC + 2位MNC编码。如：

46000 – 中国移动

46001 – 中国联通

46003 – 中国联通CDMA

46005 – 中国电信

WIFI下该项需置空。

*CONNECT\_TYPE*: 网络连接类型（0 – 未知或无网络连接 1 – WIFI 2 – 2G 3 – 3G 4 – 4G）

*NETWORK\_TYPE*: 网络类型，Android参见TelephonyManager（GPRS/EDGE/UMTS/CDMA/EVDO\_0/EVDO\_A/1xRTT/HSDPA/HSUPA/HSPA/IDEN/EVDO\_B等），iOS无法获取，置空即可

*GPS\_LATITUDE*: 纬度

*GPS\_LATITUDE*: 经度

*VIEW\_METRIC\_DATA\_ARRAY* :=

[

*METRIC\_DATA\_ITEM* \*

]

*METRIC\_DATA\_ITEM* :=

[

*METRIC\_KEY*,

[

*COUNT*,

*SUM\_PERF*,

*SUM\_PERF\_EXCLUSIVE*,

*MAX\_PERF*,

*MIN\_PERF*,

*SUM\_PERF\_SQUARES*

]

]

*METRIC\_KEY* := {“name” : “*METRIC\_NAME*”}

*COUNT*: 当前采样时间段内该指标的计数。

*SUM\_PERF*: 当前采样时间段内该指标的算术和（单位：毫秒）。

*SUM\_PERF\_EXCLUSIVE*: 独占执行时间和。（单位：毫秒）。独占执行时间的定义参见 [NetworkBench New Lens-Spec-Metrics.docx](../newlens-docs/trunk/Dev/NetworkBench%20New%20Lens-Spec-Metrics.docx) 4.1 响应时间、独占执行时间

*MAX\_PERF*: 当前采样时间段内该指标的最大值（单位：毫秒）

*MIN\_PERF*: 当前采样时间段内该指标的最小值（单位：毫秒）

*SUM\_PERF\_SQUARES* : 当前采样时间段内该指标的平方和（单位：毫秒2）

*COMPONENT\_METRICS\_DATA\_ARRAY* :=

[

*COMPONENT\_METRICS\_DATA\_ITEM* \*

]

*COMPONENT\_METRICS\_DATA\_ITEM* :=

[

*COMPONENT\_METRIC\_KEY*,

[

*COUNT*,

*SUM\_PERF*,

*SUM\_PERF\_EXCLUSIVE*,

*MAX\_PERF*,

*MIN\_PERF*,

*SUM\_PERF\_ EXCLUSIVE\_SQUARES*

]

]

*COMPONENT\_METRIC\_KEY* :=

{

“name” : “*COMPONENT\_METRIC\_NAME*”,

“parent”: “*PARENT\_ACTION\_ROOT\_METRIC\_NAME*”

}

*SUM\_PERF*: 当前采样时间段内该指标的计数

*SUM\_PERF\_EXCLUSIVE*: 独占执行时间和。独占执行时间的定义参见 [NetworkBench New Lens-Spec-Metrics.docx](NetworkBench%20New%20Lens-Spec-Metrics.docx) 4.1 响应时间、独占执行时间

*MAX\_PERF*: 当前采样时间段内该指标的最大值

*MIN\_PERF*: 当前采样时间段内该指标的最小值

*SUM\_PERF\_ EXCLUSIVE\_SQUARES* : 当前采样时间段内该指标的平方和，对于components和classifiedComponents，是指独占执行时间的平方和。

*GENERAL\_METRICS\_DATA\_ARRAY* :=

[

*GENERAL\_METRICS\_DATA\_ITEM* \*

]

*GENERAL\_METRICS\_DATA\_ITEM* :=

[

*METRIC\_KEY*,

[

*COUNT*,

*SUM*,

*CUSTOM\_SUM*,

*MAX\_VALUE*,

*MIN\_VALUE*,

*SUM\_OF\_SQUARES*

]

]

*METRIC\_KEY* := {“name” : “*METRIC\_NAME*”}

*METRIC\_NAME* :=

*METRIC\_SCOPE*/*METRIC\_CATEGORY*/*NAME\_OF\_METRIC*

CUSTOM\_SUM: 自定义统计数据。该项用于扩展，通常置0即可，对于某些特殊的指标，可能需要多一个扩展维度的统计（详细说明请参考指标SPEC文档 [NetworkBench New Lens-Spec-Metrics-Scope & categories.xlsx](NetworkBench%20New%20Lens-Spec-Metrics-Scope%20&%20categories.xlsx)）

**响应数据：**

{

“status” : “success | error”,

“result” : *RESP\_RESULT\_SUCCESS* | *RESP\_RESULT\_ERROR*

}

*RESP\_RESULT\_SUCCESS* :=

{}

*RESP\_RESULT\_ERROR* :=

{

“errorCode” : *ERROR\_CODE*,

“errorMessage” : “*ERROR\_MESSAGE*”

}

1. 用户交互追踪数据 - uiTraceData

**请求数据：**

{

“type” : “uiTraceData”,

“dev” : *DEVICE\_DATA*,

“uiTraces” : *UI\_TRACE\_ARRAY*

}

**参数说明：**

type: 探测数据类型（参见 4.5.1 探测数据类型）

其中，

*DEVICE\_DATA* :=

[

“*CARRIER*”,

*CONNECT\_TYPE*,

“*NETWORK\_*TYPE”,

*GPS\_LATITUDE*,

*GPS\_LONGITUDE*

]

*CARRIER* : 运营商代码，3位MCC + 2位MNC编码。如：

46000 – 中国移动

46001 – 中国联通

46003 – 中国联通CDMA

46005 – 中国电信

WIFI下该项需置空。

*CONNECT\_TYPE*: 网络连接类型（0 – 未知或无网络连接 1 – WIFI 2 – 2G 3 – 3G 4 – 4G）

*NETWORK\_TYPE*: 网络类型，Android参见TelephonyManager（GPRS/EDGE/UMTS/CDMA/EVDO\_0/EVDO\_A/1xRTT/HSDPA/HSUPA/HSPA/IDEN/EVDO\_B等），iOS无法获取，置空即可

*GPS\_LATITUDE*: 纬度

*GPS\_LATITUDE*: 经度

*UI\_TRACE\_ARRAY* :=

[

*UI\_TRACE\_ITEM* \*

]

*UI\_TRACE\_ITEM* :=

[

*UI\_TRACE\_TIME\_IN\_SECONDS*,

*VIEW\_DURATION*,

“*VIEW\_ ROOT\_METRIC\_NAME*”,

“*UI\_TRACE\_DATA*”

]

*UI\_TRACE\_TIME\_IN\_SECONDS* : Trace时间，即关联视图开始时间。EPOCH时间戳（单位：秒）

*VIEW\_DURATION* : 当前视图的持续时间（单位：毫秒）

*VIEW\_ ROOT\_METRIC\_NAME*：关联的视图指标名称，如”MobileView\/Controller\/MyViewController”。

*UI\_TRACE\_DATA* :=

[

*UI\_TRACE\_TIME\_IN\_SECONDS*,

*CPU\_SAMPLE\_DATA*,

*MEMORY\_SAMPLE\_DATA*,

*ROOT\_UI\_TRACE\_SEGMENT*

]

注：*UI\_TRACE\_DATA*：需要序列化成字符串后上传（非结构化数据直接以裸数据保存，以便服务器可选择parse-on-demand）

*UI\_TRACE\_TIME\_IN\_SECONDS* : Trace时间。EPOCH时间戳（单位：秒）。同*UI\_TRACE\_ITEM . UI\_TRACE\_TIME\_IN\_SECONDS*。

*CPU\_SAMPLE\_DATA*: CPU采样数据

*MEMORY\_SAMPLE\_DATA*: 内存采样数据

*CPU\_SAMPLE\_DATA* :=

*MEMORY\_SAMPLE\_DATA* :=

[

*CPU\_OR\_MEMORY\_SAMPLE\_ITEM* \*

]

*CPU\_OR\_MEMORY\_SAMPLE\_ITEM* :=

[

*SAMPLE\_TIME\_FROM\_TRACE\_START*,

*SAMPLE\_VALUE*

]

*SAMPLE\_TIME\_FROM\_TRACE\_START*: 采样时间减Trace开始时间。单位：毫秒。

*SAMPLE\_VALUE*: 采样值：CPU使用率（单位：%，取整）或内存使用量（单位：MB，取整）

*ROOT\_UI\_TRACE\_SEGMENT* :=

*UI\_TRACE\_SEGMENT* :=

[

*ENTRY\_TIME\_FROM\_TRACE\_START*,

*EXIT\_TIME\_FROM\_TRACE\_START*,

*SEGMENT\_ NAME*,

*THREAD\_INFO*,

*SEGMENT\_PARAMS*,

*CHILDREN\_TRACE\_SEGMENTS*

]

*THREAD\_INFO* :=

[

*THREAD\_ID*,

“*THREAD\_NAME*”

]

*CHILDREN\_TRACE\_SEGMENTS* :=

[

*UI\_TRACE\_SEGMENT* \*

]

*ENTRY\_TIME\_FROM\_ACTION\_START* : Tracer方法进入的时间减Trace开始时间。单位：毫秒。

*EXIT\_TIME\_FROM\_ACTION\_START* : Tracer方法执行完毕时间减Trace开始时间。单位：毫秒。

*SEGMENT\_NAME* : Trace segment名称*。*对于函数方法，名称格式为：*SEGMENT\_CLASS\_NAME#SEGMENT\_METHOD\_NAME*。对于网络请求，名称格式为：Network/ *SEGMENT\_CLASS\_NAME#SEGMENT\_METHOD\_NAME*

*SEGMENT\_PARAMS*: 对于普通方法的trace，该参数置空，即记为{}。对于网络请求，需特殊处理（要同时抓取访问的URL、主机及其它重要的网络参数），参数数据结构格式如下：

*SEGMENT\_PARAMS* :=

{

“host” : “*HOST\_NAME\_OR\_IP*”,

“url”: “*REQUEST\_URL*”,

“httpStatus” : *HTTP\_STATUS\_CODE*,

“errorCode”: *ERROR\_CODE*,

“bytesSent” : *BYTES\_SENT*,

“bytesReceived” : *BYTES\_RECEIVED*

}

注：*SEGMENT\_PARAMS*中参数为可选项，仅网络调用类型的需要记录，网络类的参数定义参见4.4.2 网络请求性能指标数据关于*ACTION\_DATA\_ITEM*的描述。

注：*UI\_TRACE\_DATA*为JSON格式序列化用户交互trace详情数据，Agent端应将trace数据序列化为单个字符串，这样的好处是，服务器可在最终展现的时候才解析trace数据，避免每次解析带来额外的性能损耗。

Trace数据为树状的层级关系，如下：

+ 视图 （顶级，即*ROOT\_UI\_TRACE\_SEGMENT*）

+ 方法1

+ 方法1.1

+ 方法1.2

……

+ 方法2

+ 方法2.1

+ 方法2.2

……

+ 方法3

+ 线程3函数调用堆栈

……

顶级为视图，第2级为线程，从第3级开始才是各线程真正的调用堆栈

Trace的组织方式按方法调用关系组织。

示例数据如下：

[ // Stack的顶级为视图

0, // 进入时间永远是0

10236, // 退出时间= *VIEW\_DURATION*

“MyViewController”, // *SEGMENT\_ NAME* = 视图名称

[0, “main”], // ROOT，虚拟的线程信息，线程ID置0即可

{}, // 自定义参数（无）

[

[

[

100, // 方法进入时间

1087, // 方法退出时间

// SEGMENT\_ NAME = CLASS\_NAME#METHOD\_NAME

“MyViewController#viewWillLoad”,

[1, “main”], // 线程信息，主线程ID置为1

{}, // 自定义参数（扩展用，暂置空）

[] // CHILDREN\_TRACE\_SEGMENTS

],

],

[

[

1120, // 方法进入时间

1532, // 方法退出时间

// SEGMENT\_ NAME = CLASS\_NAME#METHOD\_NAME

“MyViewController#viewDidLoad”,

[1, “main”],

{}, // 自定义参数（扩展用，暂置空）

[ // CHILDREN\_TRACE\_SEGMENTS

[

1132,

1402,

“JsonParser#parseJson”,

[1, “main”],

{},

[]

],

[

1133,

1403,

“ClientLogger$1#doInBackground”,

[1733, "AsyncTask #2"],

{},

[]

]

]

]

]

]

]

处于兼容性考虑，服务器端接收并保存该项数据时应同时保存上传数据的版本（DATA\_VERSION）。

**响应数据：**

{

“status” : “success” | “error”

}

服务器端处理成功返回success标识，否则返回error标识，无论成功或失败，Agent端均忽略响应内容。

1. Webview性能指标数据 – webviewPerfMetrics

Webview性能指标数据主要包含Webview方式（如HydridApp）访问的相关数据，包括各阶段网络访问数据（网络层下载、DNS时间、建连时间、首包时间、页面加载时间、页面渲染时间等），也包含页面加载过程中的js错误数据。

以Webview方式发起的请求仍可被探针抓到网络请求性能数据，此部分数据仍然在4.4.2 网络请求性能指标数据 中上传。

对于Webview中的页面级请求，需在webviewPerfMetrics中多提交一份Webview数据。

如果Webview渲染过程中发生了Javascript错误，还需要提交jserr数据。

jserror通过给当前window对象添加error的EventListener实现：

window.addEventListener("error",

function(e) {

alert("onerror(msg=" + e.message +

",filename=" + e.filename +

",lineno=" + e.lineno +

",colno=" + e.colno +

",error=" + e.error);

});

其中e为类型为ErrorEvent的对象，W3C对于ErrorEvent接口描述如下：

[Constructor(DOMString type, optional [ErrorEventInit](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#erroreventinit) eventInitDict), Exposed=(Window,Worker)]

interface **ErrorEvent** : [Event](http://www.w3.org/html/wg/drafts/html/master/infrastructure.html#event) {

readonly attribute DOMString [message](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#dom-errorevent-message);

readonly attribute DOMString [filename](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#dom-errorevent-filename);

readonly attribute unsigned long [lineno](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#dom-errorevent-lineno);

readonly attribute unsigned long [colno](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#dom-errorevent-colno);

readonly attribute any [error](http://www.w3.org/html/wg/drafts/html/master/webappapis.html#dom-errorevent-error);

};

**请求数据：**

{

“type” : “webviewPerfMetrics”,

“interval” : *DATA\_SENT\_INTERVAL\_IN\_SECONDS*,

“dev” : *DEVICE\_DATA*,

“webviews” : *WEBVIEW\_DATA\_ARRAY*,

“jserrs” : *JS\_ERROR\_DATA\_ARRAY*

}

**参数说明：**

type: 探测数据类型（参见 4.4.1 探测数据类型）

dev: 参见[*DEVICE\_DATA*](#DEVICE_DATA)

webviews: webview相关数据。

jserrs: 浏览器端错误

其中，

*WEBVIEW\_DATA\_ARRAY* :=

[

*WEBVIEW\_DATA\_ITEM* \*

]

*WEBVIEW\_DATA\_ITEM* :=

[

“*URL*”,

“*REQUEST\_URL\_PARAMS*”*,*

*REQUEST\_METHOD,*

*RESPONSE\_TIME*,

*PAGE\_LOAD\_TIME*,

*TIME\_TO\_RESOLVE\_DNS*,

*TIME\_TO\_CONNECT*,

*TIME\_TO\_SSL*,

*TIME\_TO\_FETCH\_FROM\_CACHE*,

*TIME\_TO\_FIRST\_PACKET*,

*TIME\_TO\_LOAD\_DOM*,

*BROWSER\_RENDER\_TIME*,

*SERVER\_QUEUEING\_TIME*,

*APPLICATION\_TIME*,

*NETWORK\_TIME*,

*FRONTEND\_TIME*,

*JS\_ERRORS*,

*HTTP\_STATUS\_CODE*,

*ERROR\_CODE*,

“*APP\_DATA*”

]

其中，

*URL*: 请求URL

*REQUEST\_URL\_PARAMS*: 根据App配置捕捉的GET/POST/HEADER参数（参见第4.3节 初始化应用实例）。*REQUEST\_PARAMS*格式为：p1=v1&p2=v2&…。若捕捉的参数同时包含GET、POST及HEADER，则将所有参数键值对拼成上述的格式。

*REQUEST\_METHOD*: 请求的方法:

0 – GET

1 – POST

2 – PUT

3 – DELETE

4 - HEAD

5 - TRACE

6 - OPTIONS

*RESPONSE\_TIME*: HTTP响应时间（单位：毫秒）。同网络性能请求数据中的响应时间。

*PAGE\_LOAD\_TIME*: 页面加载总耗时（毫秒）。从开始页面请求到页面渲染完成的时间，对支持Navigation Timing的浏览器来说，等于loadEventEnd – navigationStart

*TIME\_TO\_DNS*: DNS时间（单位：毫秒）。若无，缺省值置为-1。

*TIME\_TO\_CONNECT*: 建立连接时间（单位：毫秒）。若无，缺省值置为-1。

*TIME\_TO\_SSL*: SSL握手时间（单位：毫秒）。若无，缺省值置为-1。

*TIME\_TO\_FETCH\_FROM\_CACHE*: 当页面使用浏览器缓存时（如304），从缓存中加载的时间(responseEnd - fetchStart)

*TIME\_TO\_FIRST\_PACKET*: 首包时间（单位：毫秒）。（或者称作：后端耗时）：从开始页面请求到页面浏览器开始接收到HTML代码的时间，包括消耗在网络和服务器端的时间，对支持Navigation Timing的浏览器来说，等于responseStart – navigateonStart

*TIME\_TO\_LOAD\_DOM*: 处理DOM耗时（单位：毫秒）。浏览器消耗在将HTML代码解析成Dom树并获取和执行同步脚本的时间，包括从浏览器接收到HTML代码到HTML代码处理结束的时间，对支持Navigation Timing的浏览器来说，等于domContentLoadedEventEnd – responseStart。在处理Dom阶段，浏览器可能已经开始下载页面上的图片等元素了。

*BROWSER\_RENDER\_TIME*: 页面渲染时间（毫秒）。从HTML代码处理结束到页面渲染完成的时间，对支持Navigation Timing的浏览器来说，等于loadEventEnd – domContentLoadedEventEnd。在页面渲染阶段中包括了执行异步脚本和下载静态页面元素的时间。

*SERVER\_QUEUEING\_TIME*: 服务器阻塞时间（毫秒）。指服务器端的请求阻塞时间，即请求从Web前端服务器（例如Apache, nginx或F5负载均衡设备）到达应用服务端的时间，该性能值由应用探针采集并通过RUM嵌码写入页面底部的代码段中。

*APPLICATION\_TIME*: 应用耗时（毫秒）。指应用服务器处理请求的所消耗的时间，即应用响应时间，等于请求到达应用服务器到应用代码执行完成并输出响应信息的时间，该性能值由应用探针采集并通过RUM嵌码写入页面底部的代码段中

*NETWORK\_TIME*: 网络层耗时（单位：毫秒）。指应用请求和响应过程中消耗在网络上的时间，包括发出请求的前网络耗时和接收到响应的后网络耗时。网络耗时等于后端耗时（首包耗时）扣除服务器端阻塞耗时和应用耗时。网络耗时只包括应用基础页面（主HTML响应）的网络耗时，不包括应用页面内其他静态元素的的网络层耗时，静态元素的网络耗时已经被包含在处理Dom和渲染页面的过程中了。

*FRONT\_END\_TIME*: 前端耗时（单位：毫秒）。消耗在浏览器端的应用页面处理时间，等于处理Dom耗时 + 页面渲染耗时。

*HTTP\_STATUS\_CODE*: 参见 4.4.2 网络请求性能指标数据之[*HTTP\_STATUS\_CODE*](#HTTP_STATUS_CODE)

*NETWORK\_ERROR\_CODE*: 参见 4.4.2 网络请求性能指标数据之[*NETWORK\_ERROR\_CODE*](#NETWORK_ERROR_CODE)

*JS\_ERRORS*: 页面JS错误发生的次数。

*APP\_DATA*: 应用响应头（ X-Newlens-Application-Id）。若服务器端安装newlens agent，则服务器端嵌入该响应头，自动关联应用及移动端请求。

*JS\_ERROR\_DATA\_ARRAY* :=

[

*JS\_ERROR\_DATA\_ITEM* \*

]

*JS\_ERROR\_DATA\_ITEM* :=

[

“*REQUEST\_URL*”,

“*REQUEST\_URL\_PARAMS*”,

*REQUEST\_METHOD,*

“*ERROR\_MESSAGE*”,

*LINE\_OF\_ERROR*,

*COLUMN\_OF\_ERROR*,

“*ERROR\_DESCRIPTION*”,

“*SOURCE\_URL\_OF\_ERROR*”

]

其中，

*REQUEST\_URL*: 发生错误页面的URL

*REQUEST\_URL\_PARAMS*: 根据App配置捕捉的GET/POST/HEADER参数（参见第4.3节 初始化应用实例）。*REQUEST\_PARAMS*格式为：p1=v1&p2=v2&…。若捕捉的参数同时包含GET、POST及HEADER，则将所有参数键值对拼成上述的格式。

*REQUEST\_METHOD*: 请求的方法:

0 – GET

1 – POST

2 – PUT

3 – DELETE

4 - HEAD

5 - TRACE

6 - OPTIONS

*ERROR\_MESSAGE*: 错误消息。对应ErrorEvent的message属性。

*LINE\_OF\_ERROR*: 错误所在行号。对应ErrorEvent的lineno属性。

*COLUMN\_OF\_ERROR*: 错误所在列。对应ErrorEvent的colno属性。

*ERROR\_DESCRIPTION*: 错误详细描述。对应ErrorEvent的error属性。

*SOURCE\_URL\_OF\_ERROR*: 错误源代码的URL。对应ErrorEvent的filename属性。若与*REQUEST\_URL*相同，则置为#

## ~~诊断消息日志 – mobileDiagnoticsData~~

该消息格式为2.0.0版以前的设计，新的Crash消息参见4.6 上传崩溃日志

主要用于上传Crash日志。

**URL:**

http(s)://*DC\_HOST*/mobileDiagnosticsData?token=*DATA\_TOKEN*

**请求数据：**

[

*DIAGNOSTIC\_MESSAGE\_ITEM* \*

]

其中：

*DIAGNOSTIC\_MESSAGE\_ITEM* :=

{

“timestamp” : *DIAGNOSTIC\_TIME\_IN\_SECONDS*,

“message” : “*DIAGNOSTIC\_MESSAGE*”,

“log” : “*DIAGNOSTIC\_LOG*”,

“level” : *DIAGNOSTIC\_LEVEL*,

“appstart” : *APP\_START\_TIME\_IN\_SECONDS*

}

*DIAGNOSTIC\_MESSAGE*: 诊断消息简短描述

*DIAGNOSTIC\_LOG* : 诊断消息日志信息

*DIAGNOSTIC\_LEVEL* : 诊断消息级别。0 – 常规诊断消息 1 – 轻微异常（不影响Agent正常运行） 2 – 严重异常（可能影响Agent正常运行或导致数据异常，但不影响Agent以外的应用程序运行） 3 – 致命异常（Crash，或可能导致Agent崩溃，或影响应用程序运行） 9 – 由于探针本身缺陷引发的异常或崩溃

*APP\_START\_TIME\_IN\_SECONDS*: 应用启动时间，即发生initMobileApp操作的时间戳（秒）

**响应数据：**

{

“status” : “success” | “error”

}

服务器端处理成功返回success标识，否则返回error标识，无论成功或失败，Agent端均忽略响应内容。

## 上传崩溃日志 – reportCrash

1. 上传崩溃日志

上传崩溃日志，一旦有崩溃，应立即上传（可能发生在initMobileApp之前）。

**URL:**

http(s)://*DC\_HOST*/reportCrash?version=*DATA\_VERSION*&token=*DATA\_TOKEN*

注：token参数为非必需项。若崩溃在initMobileApp操作之前发生，则不设置token参数。

**请求数据：**

{

“did” : “*DEVICE\_ID*”?,

“dev” : *DEVICE\_INFO* ?,

“app” : *APP\_INFO* ? ,

“data” : *CRASH\_DATA*

}

其中：

*DEVICE\_ID* : 设备ID。非必需项。若崩溃在initMobileApp操作之后发生，即连同token参数一起上传时，则无需设置did参数；若崩溃在initMobileApp操作之前发生，则必须上传did参数（嵌码后第一次运行即发生崩溃时例外，此时，尚未从服务器生成设备ID）。

*DEVICE\_INFO*: 初始化设备信息。数据结构同[“4.3 初始化应用实例” 一节*DEVICE\_INFO*](#DEVICE_INFO)。若崩溃在initMobileApp操作之后发生，即连同token参数一起上传时，则无需设置dev参数；若崩溃在initMobileApp操作之前发生，则必须上传dev参数

*APP\_INFO*: 初始化App信息。数据结构同[“4.3 初始化应用实例” 一节*APP\_INFO*](#APP_INFO)。若崩溃在initMobileApp操作之后发生，即连同token参数一起上传时，则无需设置dev参数；若崩溃在initMobileApp操作之前发生，则必须上传dev参数

*CRASH\_DATA* :=

[

*CRASH\_ITEM* \*

]

*CRASH\_ITEM* :=

[

*CRASH\_TIME\_IN\_SECONDS*,

*APP\_START\_TIME\_IN\_SECONDS*,

*APP\_INIT\_FLAG*,

"*CRASH\_UUID*",

"*CRASH\_MESSAGE*"*,*

*CRASH\_STACK\_TRACES,*

*DEVICE\_DATA,*

*APP\_INFO\_OVERRIDE,*

"*ADDITIONAL\_INFO*"*,*

"*OS\_BUILD\_VERSION*",

*IMAGE\_UUID\_LIST*

]

*CRASH\_TIME\_IN\_SECONDS*: 崩溃时间戳（秒）

*APP\_START\_TIME\_IN\_SECONDS*: 应用启动时间，通常是发生initMobileApp操作的时间戳（秒）。若崩溃在initMobileApp操作之前发生且无法捕捉App真实启动时间，则该时间同崩溃时间戳。

*APP\_INIT\_FLAG*: 探针是否成功执行initMobileApp：0 – 未执行initMobileApp或执行失败 1 –initMobileApp操作执行成功。若崩溃在initMobileApp操作之前发生，该标识应置为0。服务器端需要为该崩溃触发一次类似于initMobileApp的动作（但无需返回任何配置项的响应），否则可能导致启动次数及崩溃率计算错误。

*CRASH\_UUID*:标识Crash的唯一ID，用于服务器端识别是否重复Crash上传。iOS的Crash内置此UUID。

*CRASH\_MESSAGE*: 崩溃消息摘要描述

*CRASH\_STACK\_TRACES*: 多个线程调用堆栈的数组。包含1个或2个线程的堆栈，至少应该包含崩溃线程的堆栈，如果有可能，也应该包含主线程堆栈。崩溃线程的堆栈应始终置于堆栈列表第1个。其内部Json格式表示如下：

*CRASH\_STACK\_TRACES* :=

[

*CRASH\_STACK\_TRACE\_ITEM* \*

]

*CRASH\_STACK\_TRACE\_ITEM* :=

[

*THREAD\_ID*,

"*THREAD\_NAME*",

"*THREAD\_STACK\_TRACE*"

]

*THREAD\_ID*: 线程ID。非必需项，若无，则置0。

*THREAD\_NAME*: 线程名称。

*THREAD\_STACK\_TRACE*: 线程调用堆栈信息，格式为文本，每行（换行符为\n）为调用堆栈的一项。每线程采集的调用堆栈有最大深度限制，最大深度配置使用initMobileApp返回配置项中的["stackDepth"](#MAX_STACK_DEPTH)。

调用堆栈信息格式化规范为：

Android:

*EXCEPTION\_CLASS\_NAME*: *EXCEPTION\_MESSAGE*

at *PACKAGE\_NAME*.*CLASS\_NAME*.*METHOD\_NAME*(源文件名.java:行号)

at *PACKAGE\_NAME*.*CLASS\_NAME*.*METHOD\_NAME*(源文件名.java:行号)

Caused by *CAUSED\_EX\_CLASS\_NAME*: *CAUSED\_EX\_MESSAGE*

at *PACKAGE\_NAME*.*CLASS\_NAME*.*METHOD\_NAME*(源文件名.java:行号)

at *PACKAGE\_NAME*.*CLASS\_NAME*.*METHOD\_NAME*(源文件名.java:行号)

… *REMAINS\_STACK\_ITEMS* more

按总stacktrace项行数限制stackDepth，如果有尚有剩余的stack未打印，则显示一行 “\t… xxx more”

下面是一个Android stacktrace内容的范例：

java.lang.RuntimeException: An RuntimeException is rethrown

at com.networkbench.newlens.datacollector.util.ReTraceTest.rethrowEx(ReTraceTest.java:35)

at com.networkbench.newlens.datacollector.util.ReTraceTest.testStackTrace(ReTraceTest.java:27)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

Caused by: java.io.IOException: An IOException is thrown.

at com.networkbench.newlens.datacollector.util.ReTraceTest.throwEx(ReTraceTest.java:40)

at com.networkbench.newlens.datacollector.util.ReTraceTest.rethrowEx(ReTraceTest.java:32)

... 25 more

iOS（每行）：

IMAGE\_UUID索引编号#0x开头的十六进制内存地址#*SYMBOL*

注：*SYMBOL*可能是空字符串或者<redacted>。

iOS符号化的过程如下：

./atosl -arch armv7 -o 453\_C950-6E33-31E1-98E5-B0D7DEEF57B5.dsymb 0x0a0b3

main (in 453\_C950-6E33-31E1-98E5-B0D7DEEF57B5.dsymb) (main.m:14)

服务器端需要对符号化结果做处理，处理后的格式规范为：

* + - 1. 每行对应调用堆栈的一项，使用换行符(\n)分隔
      2. 符号化后的调用堆栈项规范格式为：

IMAGE\_NAME\t0x十六进制内存地址\t符号化结果

符号化结果需去掉中间(in xxxx.dysmb)的部分

例如，上例中的符号化调用堆栈为：

SpeedTest 0x0a0b3 main (main.m:14)

注:

崩溃时调用堆栈列表包含1个或2个线程的堆栈，至少应该包含崩溃线程的堆栈，如果有可能，也应该包含主线程堆栈。崩溃线程的堆栈应始终置于堆栈列表第1个。

*DEVICE\_DATA :=*

[

“*CARRIER*”,

*CONNECT\_TYPE*,

“*NETWORK\_*TYPE”

]

*CARRIER*: 运营商代码。[同initMobileApp的CARRIER](#CARRIER)

*CONNECT\_TYPE*: 网络接入方式。[同initMobileApp的CARRIER](#CARRIER)

*NETWORK\_*TYPE: 网络类型。[同initMobileApp的CARRIER](#CARRIER)

*APP\_INFO\_OVERRIDE* : App相关信息。此项为非必需项，仅当App版本或*TINGYUN\_BUILD\_ID*或探针版本信息发生变化，若未发生变化，则该项置为null或空数组（[]）。其中，*TINGYUN\_BUILD\_ID*通常是随App版本变化的。

*APP\_INFO\_OVERRIDE :=*

[

*“APP\_VERSION”,*

*“AGENT\_VERSION”,*

*“TINGYUN\_BUILD\_ID”*

]

*APP\_VERSION*: App版本。[同initMobileApp的APP\_VERSION](#APP_INFO)。

*AGENT\_VERSION*: [同initMobileApp的AGENT\_VERSION](#AGENT_VERSION)。

*TINGYUN\_BUILD\_ID*: [同initMobileApp的TINGYUN\_BUILD\_ID](#TINGYUN_BUILD_ID)。

*ADDITIONAL\_INFO*: 以文本存储的附加信息，文本内容为KEY/VALUE的Json map结构（需采集的附加项见附表）。注意：该字段是文本类型，需将Json map序列化为字符串。

*ADDITIONAL\_INFO :=*

{

"*ITEM\_KEY*":*ITEM\_VALUE*

}

*OS\_BUILD\_VERSION*: 操作系统Build版本。iOS符号化时需要该参数，用于定位系统镜像dSYM文件。Android可忽略此项，置为null或空字符串即可。

*IMAGE\_UUID\_LIST*: iOS崩溃信息服务器端符号化时所需的Image UUID列表。Android忽略此项，置为null或空数组（[]）即可。

*IMAGE\_UUID\_LIST :=*

[

"*IMAGE\_ITEM*" \*

]

*IMAGE\_ITEM := IMAGE\_NAME*,*UUID*,*CPU\_ARCH*

表：附加信息采集项一览表

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Value类型 | 名称 | 备注 |
| pt | string | CPU型号 | 如高通MSM8274AC |
| arch | string | CPU指令集架构 | 如ARM A8 |
| pu | int | CPU使用率(%) | 百分数取整。如80 |
| mem | int | 剩余内存（MB） |  |
| sp | int | 剩余存储空间（MB） | 内置ROM存储的空间 |
| sd | int | SD卡剩余存储空间（MB） | iOS忽略此项 |
| pwr | int | 剩余电量（%） | 百分数取整，如81 |
| jb | int | 是否越狱或root：0 – 未越狱1 – 已越狱 |  |
| gps | int | 是否打开GPS：0 – 关闭 1 – 开启 |  |
| bt | int | 是否打开蓝牙：0 – 关闭 1 – 开启 |  |
| or | int | 设备朝向：0 - 竖屏 1 – 横屏 |  |
| orui | int | 应用UI朝向：0 – 竖屏 1 – 横屏 |  |
| ne | int | 是否贴耳：0 – 关闭 1 – 开启 |  |
| img | string | （iOS）App Image名称 | 适用于iOS，用于识别堆栈中用户级代码。Android忽略此项。 |
| cust | map | 用户自定义附加信息。 | 用户可通过调用Agent开放接口自定义附加信息，Agent开放接口及调用由移动探针定义。此项非必需项。 |
| tr | list | *TRAIL\_INTERACTIONS\_AND\_ACTIONS*用户交互操作列表 | *TRAIL\_INTERACTIONS\_AND\_ACTIONS*的详细数据结构参见[注2：*TRAIL \_INTERACTIONS\_AND\_ACTIONS*结构](#TRAIL_INTERACTIONS_AND_ACTIONS) |

注：

* 1. 崩溃对应的App版本/听云探针版本在个别情况下是可能与initMobileApp请求时的App版本/听云探针版本不一致。个别情况下（这种现象主要发生在iOS设备），崩溃日志可能无法及时上报，通常探针会在下次App重启后上报，但如果在App重启前，恰好用户升级了App，则可能出现initMobileApp请求时的App版本/听云探针版本不一致的现象。
  2. *TRAIL\_INTERACTIONS\_AND\_ACTIONS* 数据结构如下：

*TRAIL\_INTERACTIONS\_AND\_ACTIONS* :=

[

*TRAIL\_INTERACTION\_AND\_ACTION\_ITEM* \*

]

*TRAIL\_INTERACTIONS\_AND\_ACTIONS*: Crash交互轨迹节点列表。交互轨迹节点为2层结构：视图及动作。若交互轨迹节点数量超过节点最大数量限制（[参见“初始化应用实例”一节*CRASH\_TRAIL\_NODES\_MAX\_COUNT*](#CRASH_TRAIL_NODES_MAX_COUNT)），则丢弃交互轨迹中较早发生的节点。

*TRAIL\_INTERACTION\_AND\_ACTION\_ITEM* :=

[

“*UI\_VIEW\_ACTIVITY\_NAME*”,

*EVENT\_TIME\_SINCE\_APPSTART\_IN\_SECONDS*,

“*EVENT\_NAME*”,

“*EVENT\_TARGET\_ID*”,

“*EVENT\_TAG*”]

*UI\_VIEW\_TIME\_SINCE\_APPSTART\_IN\_SECONDS*: ViewController/Activity开始时间（相对于App启动的时间，单位：秒）。若事件无法关联到View，则置0。

*UI\_VIEW\_ACTIVITY\_NAME:* ViewController/Activity名称（类名，不含包名）。若事件无法关联到View，则置空(“”)。

*EVENT\_TIME\_SINCE\_APPSTART\_IN\_SECONDS*: 用户交互动作或事件发生时间（相对于App启动的时间，单位：秒）

*EVENT\_NAME*:动作名称（如onClick）。

*EVENT\_TARGET\_ID*: 触发该事件的控件标识。Android下为控件ID，iOS下使用控件frame坐标作为标识。

*EVENT\_TAG*: 用户自定义参数（控件关联的Tag值）。

**响应数据：**

{

“status” : “success” | “error”

}

服务器端处理成功返回success标识，否则返回error标识，无论成功或失败，Agent端均忽略响应内容。

**崩溃上传及重传机制：**

1. 崩溃发生后应第一时间上传
2. 若上传失败，需将崩溃相关信息暂存至本地；下次App重启initMobileApp操作后应立即将暂存崩溃信息上传，成功后将本地暂存的崩溃删除。
3. 重传操作之前检查本地暂存的崩溃消息，最多5个，最长时间为3天（若过期或超过最大数量限制，删除较早的崩溃消息）。
4. Mapping-file或Symbol-file的处理

注：此节描述探针SDK通过脚本或SDK自动上传mapping-file或symbol-file的流程，可能不遵循其它节约定的交互协议。

上传接口：

POST

[https://report.tingyun.com/symbol/authkey/*AUTH\_KEY*/appkey/*APP\_KEY*](https://report.tingyun.com/symbol/authkey/AUTH_KEY/appkey/APP_KEY)

*SYMBOL\_FILE\_OR\_MAPPING\_FILE*

或：

[https://report.tingyun.com/symbol/authkey/*AUTH\_KEY*/appkey/*APP\_KEY*](https://report.tingyun.com/symbol/authkey/AUTH_KEY/appkey/APP_KEY)*/appversion/APP\_VERSION*

*SYMBOL\_FILE\_OR\_MAPPING\_FILE*

支持上传编码：multipart/form-data或application/octet-stream

注：multipart/form-data编码是浏览器文件表单的上传方式

参数说明：

*AUTH\_KEY*: 听云API账号授权Key。由报表系统生成。

*APP\_KEY*: 听云App授权Key。即交互协议中HTTP头中对应的X-License-Key。

*APP\_VERSION*: App版本。此参数为可选参数，iOS通过IDE自动上传时推荐上传此参数。

HTTP头：

X-Upload-Type: 上传方式 1 - 通过命令行上传（缺省） 2 - SDK打包时自动上传

SDK打包自动上传时需携带HTTP头X-Upload-Type: 2

以上参数均为用户配置，如通过SDK在打包阶段自动上传mapping/symbol file，需在开发环境下配置如下：

tingyun.properties:

# 是否自动上传mapping file

mapping\_file\_auto\_upload=true|false

# 听云API账号授权Key

auth\_key=xxxxx

# 听云App授权Key

app\_key=xxxx

Mapping-file文件格式：

Android探针上传的mapping-file中须包含TINGYUN\_BUILD\_ID（用于标识mapping-file和App版本之间的对应关系），mapping-file格式为：

proguard原生mapping-file内容 + 换行符（\n）+ # TY\_BUILD\_ID=*TINGYUN\_BUILD\_ID*

iOS探针上传的符号文件中包含1个或多个Image UUID，服务器端应使用符号文件中的Image UUID作为TINGYUN\_BUILD\_ID。