uniapp 安卓腾讯Bugly专业版原生插件

目　录

使用说明 1
- 使用方法 1.1
- 更新日志 1.2

插件方法 2
- 初始化 2.1
- 更新设备ID 2.2
- 更新用户ID 2.3
- 更新设备型号 2.4
- 上报自定义异常 2.5
- 添加自定义数据 2.6
- 移除自定义数据 2.7
- 进入自定义场景 2.8
- 退出自定义场景 2.9
- 设置crash监控动态开关 2.10
- 获取所有监控项 2.11
- 设置性能监控动态开关 2.12
- 设置单个性能监控动态开关 2.13
- 停止所有的性能监控项 2.14
- 开启启动监控 2.15
- 启动监控打点 2.16
- 启动监控结束打点 2.17
- 结束整个启动监控 2.18
- 崩溃模拟 2.19

使用方法

介绍

Bugly专业版是TDS腾讯端服务（Tencent Device-oriented Service）旗下的端质量监控平台，通过采集、监控、定位、告警等核心能力，提供专业的质量监控服务，帮助开发者及时发现并解决质量问题，打造高质量应用。

Bugly专业版本可以帮助你发现多类异常问题，如崩溃，ANR，OOM，内存使用异常，应用操作卡顿，启动耗时过长等质量问题。Bugly专业版通过灵活的告警配置，支持分钟粒度的异常探测，快速识别异常并及时告警。与此同时，平台提供了丰富的归因分析能力，包含不限于异常现场分析，条件对比分析，自定义下钻分析，维度分析等等。通过分钟级的上报趋势，以及多维度统计，用户可以快速了解到各类指标的变化趋势。

官方文档
SDK接入指引

联系作者

关注微信公众号可联系作者

插件地址

https://ext.dcloud.net.cn/plugin?id=15158

用法

在需要使用插件的页面加载以下代码

使用方法

```
const module = uni.requireNativePlugin("leven-qqBuglyPro-BuglyModule");
```

页面内容

```
<template>
  <view>
    <uni-card title="腾讯bugly专业版APP性能监控原生插件">
      <view style="margin-bottom: 20px;">
        <button type="primary" @click="init">初始化</button>
        <button type="primary" @click="updateUniqueId">更新设备ID</button>
        <button type="primary" @click="updateUserId">更新用户ID</button>
        <button type="primary" @click="updateDeviceModel">更新设备型号</button>
        <button type="primary" @click="postException">上报自定义异常</button>
        <button type="primary" @click="putUserData">添加自定义数据</button>
        <button type="primary" @click="removeUserData">移除自定义数据</button>
        <button type="primary" @click="enterScene">进入自定义场景</button>
        <button type="primary" @click="exitScene">退出自定义场景</button>
        <button type="primary" @click="setCrashMonitorAble">设置crash监控动态开关</b
        <button type="primary" @click="getBuglyMonitorNames">获取所有监控项</button>
        <button type="primary" @click="setPerformanceMonitorsAble">设置性能监控动态开
        <button type="primary" @click="setPerformanceMonitorAble">设置单个性能监控动态
        <button type="primary" @click="abolishPerformanceMonitors">停止所有的性能监控
        <button type="primary" @click="appLaunchStart">开启启动监控</button>
        <button type="primary" @click="appLaunchSpanStart">启动监控打点</button>
        <button type="primary" @click="appLaunchSpanEnd">启动监控结束打点</button>
        <button type="primary" @click="appLaunchEnd">结束整个启动监控</button>
        <button type="primary" @click="crashTest('Java')">崩溃模拟JAVA异常</button>
        <button type="primary" @click="crashTest('Native')">崩溃模拟Native异常</butt
        <button type="primary" @click="crashTest('ANR')">模拟ANR异常</button>
        <button type="primary" @click="crashTest('OOM')">模拟OOM异常</button>
      </view>
    </uni-card>
  </view>
  </view>
</template>

<script>
  const module = uni.requireNativePlugin("leven-qq-buglyPro-BuglyModule");
  export default {
    data() {
      return {}
    },
    methods: {
      // 初始化
      init() {
        module.init({
```

```
        // 【必需设置】在Bugly 专业版 注册产品的appID
        appID: "c6a0a9f61f",
        // 【必需设置】在Bugly 专业版 注册产品的appKey
        appKey: "70472715-1aaa-49c4-aef3-1c05275d24e3",
        // 【推荐设置】设置设备唯一ID，必须保证唯一性，不设置则由Bugly生成唯一ID，影响设备异
        uniqueId: "asdasd",
        // 【推荐设置】设置用户ID，影响用户异常率的统计,建议sp保存复用，同一进程生命周期里面
        userId: "leven",
        // 【推荐设置】设置设备类型，设置机型后，Bugly SDK不再读取系统的机型
        deviceModel: "phone",
        // 【推荐设置】设置App版本号，不设置则从packageManager中读取。建议按应用的规范，主
        appVersion: "1.0.0.20231027",
        // 【推荐设置】设置App版本的构建号，用于Java堆栈翻译关联版本，跟上传符号表的构建号参
        buildNumber: "100",
        // 【推荐设置】设置版本类型
        appVersionType: "debug",
        // 设置App的渠道
        appChannel: "android1",
        //设置Crash时是否抓取全部线程堆栈，默认开启
        enableAllThreadStackCrash: true,
        //设置Anr时是否抓取全部线程堆栈，默认开启
        enableAllThreadStackAnr: true,
        //设置性能监控时开启Crash保护模式，默认开启
        enableCrashProtect: true
      }, res => {
        console.log(res)
      })
    },
    // 更新设备id
    updateUniqueId() {
      module.updateUniqueId({
        deviceId: "abcdefg"
      }, res => {
        console.log(res)
      })
    },
    // 更新用户id
    updateUserId() {
      module.updateUserId({
        userId: "leven1"
      }, res => {
        console.log(res)
      })
    },
    // 更新设备型号
    updateDeviceModel() {
      module.updateDeviceModel({
        model: "ANN00"
      }, res => {
        console.log(res)
```

```
        })
      },
      // 上报自定义异常
      postException() {
        module.postException({
          // 异常类型 u3d c# : 4 | js : 8 | cocos2d lua : 6
          category: 8,
          // 错误类型
          errorType: "JSON",
          // 错误信息
          errorMsg: "这是一条自定义异常消息",
          // 出错堆栈
          stack: "Exception",
          // 额外信息
          extraInfo: [{
            key: "extra1",
            value: "extra1"
          }, {
            key: "extra2",
            value: "extra2"
          }]
        }, res => {
          console.log(res)
        })
      },
      // 添加自定义数据
      putUserData() {
        module.putUserData({
          key: "key1",
          value: "value1"
        }, res => {
          console.log(res)
        })
      },
      // 移除自定义数据
      removeUserData() {
        module.removeUserData({
          key: "key1"
        }, res => {
          console.log(res)
        })
      },
      // 进入自定义场景
      enterScene() {
        module.enterScene({
          sceneName: "sceneName1"
        }, res => {
          console.log(res)
        })
      },
```

使用方法

```
    // 退出自定义场景
    exitScene() {
      module.exitScene({
        sceneName: "sceneName1"
      }, res => {
        console.log(res)
      })
    },
    // 设置crash监控动态开关
    setCrashMonitorAble() {
      module.setCrashMonitorAble({
        // crash类型，可填入JAVA_CRASH、NATIVE_CRASH、ANR_CRASH
        crashType: "JAVA_CRASH",
        // true 打开，false 关闭
        isAble: true
      }, res => {
        console.log(res)
      })
    },
    // 获取所有监控项
    getBuglyMonitorNames() {
      module.getBuglyMonitorNames(res => {
        console.log(res)
      })
    },
    // 设置性能监控动态开关
    setPerformanceMonitorsAble() {
      module.setPerformanceMonitorsAble({
        // 监控项
        monitorList: ["looper_metric", "looper_stack"],
        // true 打开，false 关闭
        isAble: true
      }, res => {
        console.log(res)
      })
    },
    // 设置单个性能监控动态开关
    setPerformanceMonitorAble() {
      module.setPerformanceMonitorAble({
        // 监控项
        monitorName: "looper_metric",
        // true 打开，false 关闭
        isAble: false
      }, res => {
        console.log(res)
      })
    },
    // 停止所有的性能监控项
    abolishPerformanceMonitors() {
      module.abolishPerformanceMonitors(res => {
```

```
            console.log(res)
          })
        },
        // 开启启动监控
        appLaunchStart() {
          module.appLaunchStart({
            // 启动标签，可以为空
            tag: "login"
          }, res => {
            console.log(res)
          })
        },
        // 启动监控打点，必须先开启启动监控
        appLaunchSpanStart() {
          module.appLaunchSpanStart({
            // 开始打点的span名称
            spanName: "verification_code",
            // 当前要打点的span的父span的名称
            parentSpanName: "login"
          }, res => {
            console.log(res)
          })
        },
        // 启动监控结束打点，必须先开启启动监控
        appLaunchSpanEnd() {
          module.appLaunchSpanEnd({
            // 开始打点的span名称
            spanName: "verification_code"
          }, res => {
            console.log(res)
          })
        },
        // 结束启动监控
        appLaunchEnd() {
          module.appLaunchEnd(res => {
            console.log(res)
          })
        },
        // 崩溃测试
        crashTest(type) {
          module.crashTest({
            //Java：Java异常，Native：Native异常
            type: type
          }, res => {
            console.log(res)
          })
        }
      }
    }
  </script>
```

使用方法

```
<style>

</style>
```

更新日志

首次发布

初始化

方法名

`init`

> - 设备ID非常重要，Bugly使用设备ID来计算设备异常率，强烈建议应用设置正确的设备ID，以确保设备的唯一性。
> - 需要在调用init接口，完成初始化后，再调用其他接口，进行定制化设置，否则设置不生效
> - 详细说明请参考【Android SDK接入指引】

用法

- 用法如下：

```
module.init({
  // 【必需设置】在Bugly 专业版 注册产品的appID
  appID: "c6a0a9f61f",
  // 【必需设置】在Bugly 专业版 注册产品的appKey
  appKey: "70472715-1aaa-49c4-aef3-1c05275d24e3",
  // 【推荐设置】设置设备唯一ID，必须保证唯一性，不设置则由Bugly生成唯一ID，影响设备异常率的
  uniqueId: "asdasd",
  // 【推荐设置】设置用户ID，影响用户异常率的统计,建议sp保存复用，同一进程生命周期里面，暂不
  userId: "leven",
  // 【推荐设置】设置设备类型，设置机型后，Bugly SDK不再读取系统的机型
  deviceModel: "phone",
  // 【推荐设置】设置App版本号，不设置则从packageManager中读取。建议按应用的规范，主动设置
  appVersion: "1.0.0.20231027",
  // 【推荐设置】设置App版本的构建号，用于Java堆栈翻译关联版本，跟上传符号表的构建号参数保持
  buildNumber: "100",
  // 【推荐设置】设置版本类型
  appVersionType: "debug",
  // 设置App的渠道
  appChannel: "android1",
  //设置Crash时是否抓取全部线程堆栈，默认开启
  enableAllThreadStackCrash: true,
  //设置Anr时是否抓取全部线程堆栈，默认开启
  enableAllThreadStackAnr: true,
  //设置性能监控时开启Crash保护模式，默认开启
  enableCrashProtect: true
}, res => {
```

```
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
appID	String	是	无	注册产品的appID
appKey	String	是	无	注册产品的appKey
uniqueId	String	否	无	设备唯一ID
userId	String	否	无	用户ID
deviceModel	String	否	无	设备类型
appVersion	String	否	无	App版本号
buildNumber	String	否	无	App版本的构建号
appVersionType	String	否	无	版本类型
appChannel	String	否	无	App的渠道
enableAllThreadStackCrash	boolean	否	true	Crash时是否抓取全部线程堆栈
enableAllThreadStackAnr	boolean	否	true	Anr时是否抓取全部线程堆栈
enableCrashProtect	boolean	否	true	性能监控时开启Crash保护模式

回调

- 示例

```
{
    "data": {},
    "message": "初始化成功",
    "code": 0
}
```

- 回调说明：

初始化

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

更新设备ID

方法名

`updateUniqueId`

用法

- 用法如下：

```
module.updateUniqueId({
  deviceId: "abcdefg"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
deviceId	String	是	无	设备id

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

更新用户ID

方法名

`updateUserId`

用法

- 用法如下：

```
module.updateUserId({
  userId: "leven1"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
userId	String	是	无	用户id

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

更新设备型号

方法名

`updateDeviceModel`

用法

- 用法如下：

```
module.updateDeviceModel({
  model: "ANN00"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
model	String	是	无	设备型号

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

上报自定义异常

方法名

`postException`

用法

- 用法如下：

```
module.postException({
  // 异常类型 u3d c# : 4 | js : 8 | cocos2d lua : 6
  category: 8,
  // 错误类型
  errorType: "JSON",
  // 错误信息
  errorMsg: "这是一条自定义异常消息",
  // 出错堆栈
  stack: "Exception",
  // 额外信息
  extraInfo: [{
    key: "extra1",
    value: "extra1"
  }, {
    key: "extra2",
    value: "extra2"
  }]
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
category	String	是	无	异常类型，u3d c# : 4 , js : 8 , cocos2d lua : 6
errorType	String	是	无	错误类型
errorMsg	String	是	无	错误信息
stack	String	是	无	出错堆栈

参数名	参数类型	是否必填	默认值	参数描述
extraInfo	Array	否	无	额外信息

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

添加自定义数据

> Bugly支持业务添加自定义数据，可以保存在Crash发生时、或发生前的一些自定义环境信息。自定义数据会随异常一起上报，并在异常个例详情界面的「自定义字段」中展示。自定义数据最多支持50对key-value

- 用法如下：

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
key	String	是	无	自定义key
value	String	是	无	自定义value

- 示例

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

移除自定义数据

方法名

`removeUserData`

用法

- 用法如下：

```
module.removeUserData({
  key: "key1"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
key	String	是	无	自定义key

回调

- 示例

```
{
    "data": {
        "value":"value1"
    },
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象

移除自定义数据

参数名	参数类型	参数描述
data.value	String	移除的key对应的值
code	Integer	返回类型，0.成功，其他：失败

进入自定义场景

方法名

`enterScene`

用法

- 用法如下：

```
module.enterScene({
  sceneName: "sceneName1"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
sceneName	String	是	无	场景名称

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

退出自定义场景

方法名

用法

- 用法如下：

```

```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
sceneName	String	是	无	场景名称

回调

- 示例

```

```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置crash监控动态开关

方法名

`setCrashMonitorAble`

用法

- 用法如下：

```
module.setCrashMonitorAble({
  // crash类型，可填入JAVA_CRASH、NATIVE_CRASH、ANR_CRASH
  crashType: "JAVA_CRASH",
  // true 打开，false 关闭
  isAble: true
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
crashType	String	是	无	crash类型，可填入JAVA_CRASH、NATIVE_CRASH、ANR_CRASH
isAble	Boolean	是	无	true 打开，false 关闭

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

获取所有监控项

方法名

`getBuglyMonitorNames`

用法

- 用法如下：

```
module.getBuglyMonitorNames(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
    "data": {
        "looperStack": "looper_stack",
        "memoryQuantile": "memory_quantile",
        "launchMetric": "launch_metric",
        "activityLeak": "activity_leak",
        "looperMetric": "looper_metric",
        "bigBitmap": "big_bitmap"
    },
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象

获取所有监控项

参数名	参数类型	参数描述
data.looperStack	String	卡顿监控
data.memoryQuantile	String	内存峰值
data.launchMetric	String	启动指标
data.activityLeak	String	JAVA内存泄漏
data.looperMetric	String	卡顿指标
data.bigBitmap	String	大图分析
code	Integer	返回类型，0.成功，其他：失败

设置性能监控动态开关

方法名

`setPerformanceMonitorsAble`

用法

- 用法如下：

```
module.setPerformanceMonitorsAble({
  // 监控项
  monitorList: ["looper_metric", "looper_stack"],
  // true 打开，false 关闭
  isAble: true
}, res => {
  console.log(res)
})
```

> 监控项可在获取所有监控项接口中查看

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
monitorList	Array	是	无	监控项列表，可在获取所有监控项接口中查看
isAble	Boolean	是	无	true 打开，false 关闭

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
```

```
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置单个性能监控动态开关

方法名

`setPerformanceMonitorAble`

用法

- 用法如下：

```
module.setPerformanceMonitorAble({
  // 监控项
  monitorName: "looper_metric",
  // true 打开，false 关闭
  isAble: false
}, res => {
  console.log(res)
})
```

> 可在获取所有监控项接口中查看

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
monitorName	String	是	无	监控项，可在获取所有监控项接口中查看
isAble	Boolean	是	无	true 打开，false 关闭

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
```

```
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

停止所有的性能监控项

方法名

`abolishPerformanceMonitors`

用法

- 用法如下：

```
module.abolishPerformanceMonitors(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

开启启动监控

方法名

`appLaunchStart`

用法

- 用法如下：

```
module.appLaunchStart({
  // 启动标签，可以为空
  tag: "login"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
tag	String	否	无	启动标签

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

启动监控打点

方法名

`appLaunchSpanStart`

> 必须先开启启动监控

用法

- 用法如下：

```
module.appLaunchSpanStart({
  // 开始打点的span名称
  spanName: "verification_code",
  // 当前要打点的span的父span的名称
  parentSpanName: "login"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
spanName	String	是	无	开始打点的span名称
parentSpanName	String	否	无	当前要打点的span的父span的名称

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
```

```
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

启动监控结束打点

方法名

`appLaunchSpanEnd`

> 必须先开启启动监控

用法

- 用法如下：

```
module.appLaunchSpanEnd({
  // 结束打点的span名称
  spanName: "verification_code"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
spanName	String	是	无	结束打点的span名称

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示

启动监控结束打点

参数名	参数类型	参数描述
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

结束整个启动监控

方法名

`appLaunchEnd`

> 必须先开启启动监控

用法

- 用法如下：

```
module.appLaunchEnd(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
    "data": {},
    "message": "设置成功",
    "code": 0
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

崩溃模拟

方法名

`crashTest`

用法

- 用法如下：

```
module.crashTest({
  type: "Native"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
type	String	是	无	异常类型，Java：Java异常，Native：Native异常，ANR：ANR异常，OOM：OOM异常

回调

无