腾讯云可观测平台

用户体验监控

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

用户体验监控
接入指南
Android SDK 接入指引
SDK 集成

最近更新时间：2026-09-28 16:41:40

用户体验监控 Android SDK 支持 Gradle 依赖引入 Fat-AAR，并配合编译期插桩插件实现网络 / 启动的自动埋点。

前提条件

- Android 5.0（API 21）及以上。
- Android Studio 2022.3+、AGP 7.3.1+、Kotlin 1.7.20。
- 崩溃监控需要 NDK 21.4.7075529、CMake 3.10.2+。

操作步骤

步骤1：添加仓库

插件与 SDK 均从 CNB 制品库获取，需在 `settings.gradle` 中同时配置依赖仓库与插件仓库。

```
// settings.gradle
pluginManagement {
    repositories {
        maven { url 'https://maven.cnb.cool/tencent-
dem/release/-/packages/' }
        google()
        mavenCentral()
        gradlePluginPortal()
    }
}

dependencyResolutionManagement {
    repositories {
        maven { url 'https://maven.cnb.cool/tencent-
dem/release/-/packages/' }
        google()
```

```
        mavenCentral()
    }
}
```

若项目未使用 `dependencyResolutionManagement`（Gradle 6.x 或沿用 `allprojects`），可在项目级 `build.gradle` 中配置：

```
// 项目级 build.gradle
allprojects {
    repositories {
        maven { url 'https://maven.cnb.cool/tencent-dem/release/-/packages/' }
        google()
        mavenCentral()
    }
}
```

> ⚠ **注意：**
> 制品库完整地址须包含 `/-/packages/` 路径段，仅写 `https://cnb.cool/tencent-dem/release` 无法解析制品。

步骤2：添加依赖

```
dependencies {
    implementation 'com.tencent.tdem:tdem:0.1.24'
}
```

步骤3：应用插桩插件（网络 / 启动 / Compose Navigation 插桩）

引入并启用 tdem-plugin 0.1.12 插件，支持以下两种写法，选择其一即可。

- `plugins {}` DSL（推荐）：

```
// 模块级 build.gradle
plugins {
    id 'tdem-plugin' version '0.1.12'
}
```

- `buildscript` classpath：

```
// 项目级 build.gradle
buildscript {
    repositories {
        maven { url 'https://maven.cnb.cool/tencent-
dem/release/-/packages/' }
    }
    dependencies {
        classpath 'com.tencent.tdemplugin:tdem-plugin:0.1.12'
    }
}

// 模块级 build.gradle
apply plugin: 'tdem-plugin'
```

插桩开关（两种写法通用，下列为当前默认值）：

```
tdem {
    instrumentOkHttp = true              // OkHttp3 自动监控
    instrumentHUC = true                 // HttpURLConnection 自动监控
    instrumentSSE = true                 // SSE 自动监控
    instrumentLaunch = true              // Application/Activity 启动埋点
    instrumentComposeNavigation = false  // Compose Navigation 页面流转，
默认关闭
}
```

说明：
tdem {} 是 tdem-plugin 的插件配置块，在插件引入代码之后写入。该配置块不会立即执行，仅用于定义字节码插桩开关；在项目编译打包阶段，插件读取此处配置，根据各项开关对 Class 字节码执行插桩，自动采集对应的监控埋点数据。

SDK 初始化

最近更新时间：2026-09-28 16:41:57

本文将为您介绍如何初始化用户体验监控 Android SDK。

操作步骤

1. 在自定义 `Application` 的 `onCreate` 中创建配置对象并初始化。除 `env` 、`logLevel` 外的配置项**均为必填**：`id` / `url` 缺失会导致初始化被直接跳过；`userId` / `deviceId` / `version` 缺失会导致控制台的用户、设备、版本维度无法聚合。

Kotlin

```
class MyApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        val config = TDEMConfiguration.Builder(this)
            .id("your-project-key")                    // 必填：项目标识，对应 project_key
            .url("https://dem.rumt-zh.com")            // 必填：上报服务端基地址（国内站）
            .userId("user-123")                        // 必填：业务用户标识（用户维度聚合）
            .deviceId("deviceid-123")                  // 必填：设备标识（设备异常率统计）
            .version("1.0.0")                          // 必填：应用版本号（版本对比与分布）
            .env("production")                         // 可选：运行环境，默认 production
            .logLevel(TDEM.LEVEL_DEBUG)                // 可选：日志级别，默认 WARN
            .build()
        TDEM.init(config)
    }
}
```

Java

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        TDEMConfiguration config = new TDEMConfiguration.Builder(this)
            .id("your-project-key")                          // 必填：项目标
识，对应 project_key
            .url("https://dem.rumt-zh.com")                  // 必填：上报服务
端基地址（国内站）
            .userId("user-123")                              // 必填：业务用户
标识（用户维度聚合）
            .deviceId("deviceid-123")                        // 必填：设备标识
（设备异常率统计）
            .version("1.0.0")                                // 必填：应用版本
号（版本对比与分布）
            .env("production")                               // 可选：运行环
境，默认 production
            .logLevel(TDEM.LEVEL_DEBUG)                      // 可选：日志级
别，默认 WARN
            .build();
        TDEM.init(config);
    }
}
```

2. 初始化后即可自动采集崩溃、ANR、卡顿、启动、设备信息等数据，无需额外打点。用户行为（Behavior）默认关闭，需显式开启。

> ⚠ **注意：**
>
> - `id` 与 `url` 为必填，二者任一为空时 `TDEM.init` 会跳过初始化并返回空壳实例，不会上报任何数据。`userId`、`deviceId`、`version` 同为必填，缺失不会导致初始化失败，但会使控制台的用户、设备、版本维度无法聚合，务必一并传入。
> - SDK 初始化前会先请求 `POST {url}/api/v1/config` 拉取远程配置，仅在远程返回启用时才挂载各监控插件；拉取失败时沿用本地缓存，无缓存则偏保守（不上报）。
> - 建议在用户授权个人信息保护规则后再初始化 SDK。
> - 切勿把真实 project key、采集凭证或生产环境地址提交进代码仓库。

上报域名

请根据项目所在地域选择对应的上报域名。不同站点的数据相互隔离，跨站填写会导致数据无法入库。

站点	上报域名
国内站	`https://dem.rumt-zh.com`
新加坡站	`https://dem.rumt-sg.com`
美国站	`https://dem.rumt-us.com`

接入时填写站点根地址即可，无需拼接路径，SDK 会自行拼接具体协议端点。

完整配置项说明

基础配置

下表中 `id` / `url` / `userId` / `deviceId` / `version` 为必填，其余为可选（均有默认值或可不配置）。`id` 与 `url` 由 SDK 强制校验，缺失会直接跳过初始化；`userId`、`deviceId`、`version` 缺失虽不影响 SDK 运行，但会导致控制台对应的用户 / 设备 / 版本维度无法聚合。

配置项	说明
`id`	**必填**。项目标识，对应请求体中的 `project_key`。
`url`	**必填**。上报服务端基地址，SDK 会自行拼接具体协议端点（如 `/api/v1/collect/events`）。国内站填 `https://dem.rumt-zh.com`，详见「上报域名」。
`userId`	**必填**。业务用户标识，对应上报体的 `context.user_id`；不填时该字段不会写入上报体，控制台无法按用户维度聚合分析。
`deviceId`	**必填**。宿主提供的设备标识（`String?`）。由业务侧设置，**SDK 不会读取系统设备标识**（如 `ANDROID_ID`）；未设置或传空白时，上报字面量 `not_set`，按设备维度的异常率统计会失真。也可在运行期通过 `TDEM.getInstance()` 调用 `setDeviceId` 设置、`clearDeviceId()` 清除。
`version`	**必填**。应用版本号，对应上报体的 `app.version`，用于版本对比与版本分布分析。该值是控制台版本维度的唯一来源，**SDK 不会自动回填**（插件虽会自行读取 `versionName`，但仅用于设备明细，不会写入版本维度所在的主字段）；不填则版本维度聚合失效。
`env`	运行环境标识，允许值 `production` / `development` / `gray` / `pre` / `daily` / `local` / `test` / `others`，默认 `production`；未知值（含 `staging`）归一为 `others`。
`logLevel`	日志级别，可选 `TDEM.LEVEL_DEBUG` / `TDEM.LEVEL_INFO` / `TDEM.LEVEL_WARN` /

	`TDEM.LEVEL_ERROR`，默认 `WARN`。
`batchDelay`	事件批量上报延迟（ms），自首个事件进入 buffer 后等待的最长时间，默认5000。
`batchSize`	事件批量上报大小上限，达到后立即 flush，默认50。
`sampleRate`	全局采样率（0.0 – 1.0），默认1.0。
`beforeReport`	事件上报前过滤回调，返回 `true` 放行、`false` 丢弃。
`retryConfig`	重试配置，默认最大重试3次、初始间隔 1000ms、退避乘数2.0、最终失败丢弃（`DISCARD`）。

崩溃监控（crashPluginConfig，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。开启时同时捕获 Java 与 Native 崩溃。

ANR 监控（anrPluginConfig，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。
`captureAllInitializedProcesses`	覆盖所有初始化了 SDK 的进程，默认 `true`。
`dedupeWindowMs`	重复 ANR 抑制窗口（ms）。
`javaThreadDumpEnabled`	Java 多线程堆栈采集，默认 `true`。
`signalMonitorEnabled`	Native SIGQUIT 监控，默认 `true`。
`messageQueueDumpEnabled`	主线程消息队列现场采集，默认 `true`。
`foregroundStateEnabled`	前后台状态采集，默认 `true`。
`resourceLoadEnabled`	CPU / 内存 / IO 负载采集，默认 `true`。

`maxMessageScan`	消息队列采集扫描条数上限。

卡顿监控（lagPluginConfig，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。
`lagThresholdMs`	卡顿判定阈值（ms），默认1000，最低100。
`contextCollectEnabled`	是否收集 MQ 诊断和系统状态，默认 `true`。
`foregroundOnly`	是否仅在前台检测，默认 `true`。
`dedupWindowMs`	事件去重时间窗口（ms），默认60000。
`multiStackEnabled`	是否启用多堆栈采集，默认 `true`。
`multiStackIntervalMs`	多堆栈采集间隔（ms），默认200。
`multiStackMaxSamples`	多堆栈最大采集次数，默认15，范围1 – 15。
`multiStackMaxDurationMs`	多堆栈总时长上限（ms），默认3000。

启动监控（launchPluginConfig，默认开启）

配置项	说明
`enabled`	是否启用，默认 `true`。启动监控通过 ASM 编译期插桩实现，早于 SDK 初始化执行。
`manualEndEnabled`	是否等待业务调用 `TDEM.endLaunch()` 才封口，默认 `false`（首个 Activity resume 后自动上报）。
`warmStartThresholdMs`	温启动判定阈值（ms），默认180000。
`launchTimeoutMs`	启动超时兜底（ms），默认60000。

网络监控（httpPluginConfig，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。

`okHttpEnabled`	OkHttp3 监控开关，默认 `true` 。
`urlConnectionEnabled`	HttpURLConnection 监控开关，默认 `true` 。
`sseEnabled`	SSE 监控开关，默认 `true` 。
`traceEnabled`	全链路追踪注入（W3C traceparent + SkyWalking sw8），默认 `false` 。
`collectHeadersEnabled`	采集请求 / 响应头（白名单 + 黑名单过滤），默认 `true` 。
`reportAllSuccessfulRequests`	是否上报全部成功请求，默认 `false` （仅慢成功与错误 / 取消上报）。
`slowThresholdMs`	慢请求阈值（ms），默认1000。
`urlFilter`	URL 过滤函数，返回 `false` 的 URL 不上报。
`sseStallThresholdMs`	SSE 停滞检测阈值（ms），默认60000。

用户行为监控（behaviorPluginConfig，默认关闭）

配置项	说明
`enabled`	总开关，默认 `false` （合规整改起默认 opt-in，需显式开启）。开启后采集 `page_view` / `click` / `double_tap` / `long_press` / `scroll` / `pinch_zoom` / `key_press` / `screen_toggle` / `screen_rotation` / `text_input` 等行为事件。

用户挣扎检测（strugglePluginConfig，默认开启）

配置项	说明
`enabled`	总开关，默认 `true` 。
`clickRulesEnabled`	是否检测 rage / dead / error click，默认 `true` 。
`formRulesEnabled`	是否检测 long focus，默认 `true` 。
`navigationRulesEnabled`	是否检测 back-forward，默认 `true` 。
`deadClickIgnoredViewIds`	精确忽略的 resource-id 列表（Android 特有）。

`ignoreDeadClickSelectors`	子串匹配 id / class，批量忽略 dead click（Android 特有）。

规则阈值（rage 1s/5次、dead/error 3s、long focus 20s、back-forward 10s/3次）内部写死，对齐 iOS，接入方不可修改。

Session Replay 会话录屏（replayPluginConfig，默认关闭）

配置项	说明
`enabled`	总开关，默认 `false`。
`sessionSampleRate`	全量会话采样率（0.0–1.0），默认 0.1。
`onErrorSampleRate`	错误触发采样率（0.0–1.0），默认 1.0。
`quality`	录制质量预设（LOW / MEDIUM / HIGH），默认 LOW。
`maskAllText`	是否遮罩所有文本内容，默认 `false`。
`maskAllImages`	是否遮罩所有图片内容，默认 `false`。
`maskViewClasses`	需要遮罩的 View 类全限定名集合（默认遮罩 WebView / VideoView / PlayerView / PreviewView 等媒体控件）。
`unmaskViewClasses`	不遮罩的 View 类全限定名集合。

需要针对个别控件调整遮罩时，可通过 `TDEMPrivacyMetadata.setSensitive` / `setReplayMasking` 主动声明，详见《API 说明》的「隐私标记」章节。

设备信息采集（devicePluginConfig，默认开启）

配置项	说明
`enabled`	是否启用设备信息采集，默认 `true`。

数据上报验证

最近更新时间：2026-09-28 17:02:02

本文介绍接入用户体验监控 Android SDK 后，如何验证各监控能力的数据上报是否成功。

前提条件

- 崩溃（`crashPluginConfig`）、ANR（`anrPluginConfig`）、卡顿（`lagPluginConfig`）、启动（`launchPluginConfig`）、网络（`httpPluginConfig`）、挣扎检测（`strugglePluginConfig`）、设备信息（`devicePluginConfig`）默认开启。
- 用户行为（`behaviorPluginConfig`）、Session Replay（`replayPluginConfig`）默认关闭，需显式开启。
- 验证期间建议设置 `.logLevel(TDEM.LEVEL_DEBUG)`，方便观察上报请求与运行日志。日志输出开关由 SDK 内部固定开启，`logLevel` 只控制输出级别下限，默认 `WARN`。

步骤1：检查上报请求

初始化后，SDK 会发出下列请求（Session Replay 默认关闭，开启后才有回放上报）。HTTP 2xx / 204 均视为成功：

端点	内容
`POST {url}/api/v1/config`	远程配置，仅在返回启用时才挂载各监控插件
`POST {url}/api/v1/collect/events`	标准事件（崩溃 / ANR / 卡顿 / 启动 / 网络 / 行为 / 挣扎 / 自定义事件等）
`POST {url}/api/v1/collect/replay`	Session Replay 录屏数据

`url` 填站点根地址即可，SDK 会自行拼接路径，无需手动补 `/api/v1`。

步骤2：验证各监控能力

崩溃监控

初始化后，可在页面中主动触发 Java 异常来验证：

```
// 触发未捕获的 Java 异常
throw RuntimeException("tdem verify crash")
```

Native 崩溃（SIGSEGV / SIGABRT）通过 Signal Handler 捕获，可在 demo 中触发验证。

ANR 监控

主线程 sleep 或死循环可触发 SIGQUIT ANR 检测：

```
Thread.sleep(10_000)  // 主线程阻塞触发 ANR
```

卡顿监控

主线程执行长时间任务，超过 `lagThresholdMs`（默认1000ms）即可触发卡顿上报。

启动监控

启动监控默认开启，通过 ASM 编译期插桩实现，早于 SDK 初始化执行，不依赖 `TDEM.init` 的调用时机。

- 启动类型共6种：
  - `first_start`：首次安装。
  - `upgrade_start`：版本升级后首次启动。
  - `cold_start`：进程被杀后重新启动。
  - `warm_start`：从后台切回且停留超过阈值。
  - `preheat_start`：从后台切回且停留不超过阈值，即热启动。
  - `prepare_start`：进程由 Service / BroadcastReceiver / ContentProvider 触发。
- 后台停留阈值由 `warmStartThresholdMs` 控制，默认 180000ms。
- 默认在首个 Activity resume 后自动封口并上报事件 `app_launch`。若配置 `.launchPluginConfig(LaunchPluginConfig(manualEndEnabled = true))`，则改为等待业务调用 `TDEM.endLaunch()` 后才封口。
- 首次安装后的第一次启动不采集：启动监控的开关通过 SharedPreferences 持久化，第一次启动时开关尚未写入，SDK 初始化成功写入后，第二次及后续启动才正式开启。验证启动数据需要至少冷启动两次。
- 验证方式：冷启动 App 后观察 Logcat 中的启动日志。

网络监控

网络监控默认开启，SDK 自动监听 OkHttp3 / HttpURLConnection / SSE 请求并上报耗时与错误。可通过配置 `slowThresholdMs` 验证慢请求上报。

行为监控

开启行为监控后，SDK 自动采集以下行为事件：页面（`page_view`）、触摸（`click` / `double_tap` / `long_press` / `scroll` / `pinch_zoom`）、按键与系统（`key_press` / `screen_toggle` / `screen_rotation`）、输入与表单（`text_input` / `form_focus` / `form_blur` / `form_change`）。

Kotlin

```kotlin
val config = TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
    .behaviorPluginConfig(BehaviorPluginConfig(enabled = true))
    .build()
TDEM.init(config)
```

Java

```java
import android.app.Application;
import com.tencent.tdem.TDEM;
import com.tencent.tdem.TDEMConfiguration;
import com.tencent.tdem.behavior.BehaviorPluginConfig;

TDEMConfiguration config = new TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
    .behaviorPluginConfig(new BehaviorPluginConfig(true))
    .build();
TDEM.init(config);
```

手动逻辑页可通过 `startPage` / `leavePage`（LIFO）声明：

Kotlin

```kotlin
TDEM.getInstance()?.startPage("checkout")
// ... 页面逻辑 ...
TDEM.getInstance()?.leavePage()
```

Java

```java
import com.tencent.tdem.TDEM;

TDEM.getInstance().startPage("checkout");
// ... 页面逻辑 ...
TDEM.getInstance().leavePage();
```

挣扎检测

挣扎检测默认开启，产出5类子事件：

- `rage_click`：同一元素1秒内连续点击5次。
- `dead_click`：点击后3秒内既无页面跳转也无接口调用，判定为无响应。
- `error_click`：点击后3秒内出现接口错误或 JS / Promise 错误。
- `long_focus_time`：表单控件持续聚焦超过20秒。
- `back_forward`：10秒内连续返回3次。

规则阈值内部写死，接入方不可修改。可通过 `clickRulesEnabled` / `formRulesEnabled` / `navigationRulesEnabled` 分别关闭点击类、表单类与返回类规则。
需要注意的是，默认配置下通常只有 `back_forward` 能被触发。`rage_click` / `dead_click` / `error_click` 依赖点击事件，`long_focus_time` 依赖表单聚焦事件，而这些事件全部由用户行为监控（`behaviorPluginConfig`）产生，行为监控默认关闭，因此这4类子事件会静默失效。要完整验证挣扎检测，需要同时开启行为监控：

Kotlin

```
val config = TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
    .behaviorPluginConfig(BehaviorPluginConfig(enabled = true))     // 必需：提供点击 / 表单事件源
    .strugglePluginConfig(StrugglePluginConfig(enabled = true))
    .build()
TDEM.init(config)
```

Java

```
import android.app.Application;
import com.tencent.tdem.TDEM;
import com.tencent.tdem.TDEMConfiguration;
import com.tencent.tdem.behavior.BehaviorPluginConfig;
import com.tencent.tdem.struggle.StrugglePluginConfig;

TDEMConfiguration config = new TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
```

```
    .behaviorPluginConfig(new BehaviorPluginConfig(true))     // 必需：提供点击 / 表单事件源
    .strugglePluginConfig(new StrugglePluginConfig())
    .build();
TDEM.init(config);
```

`back_forward` 消费的是页面事件，而页面事件有不受行为监控开关影响的来源（ `startPage` / `setPage` / `leavePage` ），因此它单独可用。
验证方式：连点同一按钮5次以上触发 `rage_click` ，或在10秒内连续返回3次触发 `back_forward` ，随后观察 Logcat 中的挣扎日志。

会话回放

Session Replay 默认关闭，需在初始化时显式开启：

Kotlin

```kotlin
val config = TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
    .replayPluginConfig(ReplayConfig(enabled = true))
    .build()
TDEM.init(config)
```

Java

```java
import android.app.Application;
import com.tencent.tdem.TDEM;
import com.tencent.tdem.TDEMConfiguration;
import com.tencent.tdem.replay.ReplayConfig;
import com.tencent.tdem.replay.ReplayQuality;
import com.tencent.tdem.replay.ScreenshotStrategyType;
import java.util.Collections;

TDEMConfiguration config = new TDEMConfiguration.Builder(this)
    .id("your-project-key")
    .url("https://dem.rumt-zh.com")
    .replayPluginConfig(new ReplayConfig(
        true, 0.1, 1.0, ReplayQuality.LOW, false, false,
```

```
        ScreenshotStrategyType.PIXEL_COPY,
        ReplayConfig.Companion.getDEFAULT_MASK_VIEW_CLASSES(),
        Collections.<String>emptySet()))
    .build();
TDEM.init(config);
```

- 采样是双通道：`sessionSampleRate`（默认0.1）决定常规会话是否被录制；未命中时若 `onErrorSampleRate`（默认1.0）大于0，SDK 会走滚动缓冲策略，在发生错误时冲刷并上报。
- 控制台下发的远程配置 `replay_sample_rate`（0 – 100）会覆盖本地 `sessionSampleRate`。因此“开启了回放却没看到数据”不一定是接入失败，需要依次确认：本地 `enabled` 已开启、控制台远程配置已启用回放、且会话被采样命中。
- 验证方式：开启后操作 App 若干秒，观察 Logcat 中的会话回放日志。

设备信息

- 设备信息采集默认开启，SDK 会在每条事件的上下文中附带下列设备字段：`os` / `device_type` / `device_model` / `app_version` / `sdk_version` / `network_type` / `screen_resolution` / `language` / `carrier` / `storage_free_mb` / `memory_free_mb`
- 验证方式：在控制台打开任一事件的详情，确认事件上下文中的上述字段已填充。

步骤3：主动上报(可选)

除自动采集外，SDK 提供下列主动上报接口，均通过 `TDEM.getInstance()` 获取实例后调用。

Kotlin

```
val tdem = TDEM.getInstance()

// 自定义事件
tdem?.track(
    "order_submit",
    tags = mapOf("channel" to "app"),
    properties = mapOf("amount" to 99),
)

// 自定义测速：可直接传耗时，或 start / end 成对使用
tdem?.measure("api_latency", 320, tags = mapOf("api" to "/user/info"))
tdem?.startMeasure("render")
val duration = tdem?.endMeasure("render")     // 返回耗时（ms）；未调用
startMeasure 时返回 -1
```

```
// 日志与异常：用于上报业务已捕获、未导致崩溃的异常
tdem?.captureMessage("用户完成注册", level = "info")
try {
    // 业务代码
} catch (e: Exception) {
    tdem?.captureException(e, tags = mapOf("module" to "payment"))
}
```

Java

```
import com.tencent.tdem.TDEM;
import java.util.HashMap;
import java.util.Map;

TDEM tdem = TDEM.getInstance();

// 自定义事件
Map<String, String> tags = new HashMap<>();
tags.put("channel", "app");
Map<String, Object> properties = new HashMap<>();
properties.put("amount", 99);
tdem.track("order_submit", tags, properties);

// 自定义测速：可直接传耗时，或 start / end 成对使用
Map<String, String> measureTags = new HashMap<>();
measureTags.put("api", "/user/info");
tdem.measure("api_latency", 320, measureTags, new HashMap<String,
Object>());
tdem.startMeasure("render");
long duration = tdem.endMeasure("render");     // 返回耗时（ms）；未调用
startMeasure 时返回 -1

// 日志与异常：用于上报业务已捕获、未导致崩溃的异常
tdem.captureMessage("用户完成注册", "info", null);
try {
    // 业务代码
} catch (Exception e) {
    Map<String, Object> crashTags = new HashMap<>();
```

```
    crashTags.put("module", "payment");
    tdem.captureException(e, crashTags);
}
```

- 用户标识、设备标识与全局标签：

Kotlin

```
tdem?.setUser("user-456")                                    // 设置用户
标识，用于用户维度聚合
tdem?.clearUser()                                            // 清除用户
标识
tdem?.setDeviceId("device-abc")                              // 设置设备
标识，由宿主提供
tdem?.clearDeviceId()                                        // 清除设备
标识，回落 not_set
tdem?.setTags(mapOf("role" to "admin", "team" to "dev"))     // 设置 /
追加标签
tdem?.removeTags(listOf("team"))                             // 移除指定
标签
tdem?.clearTags()                                            // 清空所有
标签
```

Java

```
import com.tencent.tdem.TDEM;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

TDEM tdem = TDEM.getInstance();

tdem.setUser("user-456");                                    // 设置用户
标识，用于用户维度聚合
tdem.clearUser();                                            // 清除用户
标识
tdem.setDeviceId("device-abc");                              // 设置设备
标识，由宿主提供
```

```
tdem.clearDeviceId();                                    // 清除设备
标识，回落 not_set

Map<String, Object> tags = new HashMap<>();
tags.put("role", "admin");
tags.put("team", "dev");
tdem.setTags(tags);                                      // 设置 /
追加标签
tdem.removeTags(Arrays.asList("team"));                  // 移除指定
标签
tdem.clearTags();                                        // 清空所有
标签
```

验证方式：调用后在控制台对应的自定义事件、日志或用户页确认数据已入库。完整签名与参数说明见 API 说明。

- 隐私标记：自动识别覆盖不到的场景（如订单金额、收货地址这类字面无语义特征的文案），可用 `TDEMPrivacyMetadata` 主动声明为敏感。

Kotlin

```
import com.tencent.tdem.common.privacy.TDEMPrivacyMetadata
import com.tencent.tdem.common.privacy.TDEMReplayMasking

// 标记为敏感：行为监控不采集该控件文本，Session Replay 遮罩该控件及其子树
TDEMPrivacyMetadata.setSensitive(orderAmountText, true)
TDEMPrivacyMetadata.setSensitive(orderAmountText, false)
// 取消标记

// 只调 Session Replay 遮罩，不影响文本采集
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.MASKED)
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.INHERIT)    // 恢复默认判定
```

Java

```
import android.view.View;
import com.tencent.tdem.common.privacy.TDEMPrivacyMetadata;
```

```
import com.tencent.tdem.common.privacy.TDEMReplayMasking;

// 标记为敏感：行为监控不采集该控件文本，Session Replay 遮罩该控件及其子树
TDEMPrivacyMetadata.setSensitive(orderAmountText, true);
TDEMPrivacyMetadata.setSensitive(orderAmountText, false);
// 取消标记

// 只调 Session Replay 遮罩，不影响文本采集
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.MASKED);
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.INHERIT);  // 恢复默认判定
```

- 标记父容器时，Session Replay 会把标记向下传递（等价于遮罩整棵子树），但行为监控只判定被标记的那个控件本身，子控件文本仍会被采集，需逐个标记。
- 验证方式：开启 Session Replay 后操作 App，确认被标记的控件在录像中已被遮罩，且该控件的文本不再出现在行为事件中。`TDEMReplayMasking` 取值与完整遮罩判定次序见 API 说明中的 隐私标记 。

步骤4：检查数据上报

在 Logcat 中观察 SDK 日志。全模块日志统一使用 `TDEM-Android` 作为 tag，模块名只出现在日志正文中：

```
adb logcat -s TDEM-Android:D

# 只看某个模块（如网络）的日志
adb logcat -s TDEM-Android:D | grep HttpPlugin
```

API 说明

最近更新时间：2026-09-28 17:02:02

本文详细介绍用户体验监控 Android SDK 的各功能接口，帮助您更灵活、深度地使用 SDK。

初始化

通过 `TDEM.init(config)` 创建实例并自动开始采集，重复调用会返回同一实例；初始化后通过 `TDEM.getInstance()` 获取单例调用实例 API，未初始化时返回 `null`。

- `TDEM` 上的静态方法共5个：

方法	用途
`TDEM.init(config)`	创建并启动 SDK，返回实例
`TDEM.getInstance()`	获取单例，未初始化返回 `null`
`TDEM.addInitListener(listener)`	注册启动结算监听，可在 `init` 之前调用
`TDEM.removeInitListener(listener)`	注销启动结算监听
`TDEM.endLaunch()`	业务就绪时手动封口启动测量

- 另有日志级别常量 `TDEM.LEVEL_DEBUG` / `TDEM.LEVEL_INFO` / `TDEM.LEVEL_WARN` / `TDEM.LEVEL_ERROR` 与版本常量 `TDEM.VERSION`。

Kotlin

```kotlin
val tdem = TDEM.getInstance()
```

Java

```java
TDEM tdem = TDEM.getInstance();    // 未初始化时返回 null
```

启动结算可异步感知。`TDEM.addInitListener` 既可在 `TDEM.init` 之前注册，也可在之后注册；远程配置闸门落定后回调一次，若注册时决策已落定则立即回放结果。

Kotlin

```kotlin
import com.tencent.tdem.core.TDEMInitListener
```

```kotlin
val listener = object : TDEMInitListener {
    override fun onInitSettled(started: Boolean, remoteConfig:
RemoteConfigResult?) {
        // started：是否已进入采集态
        // remoteConfig：本次决策使用的远程配置；拉取耗尽且无缓存时为 null
    }
}

TDEM.addInitListener(listener)      // 可在 TDEM.init 之前调用
TDEM.removeInitListener(listener)   // 注销
```

Java

```java
import com.tencent.tdem.core.TDEMInitListener;

TDEMInitListener listener = (started, remoteConfig) -> {
    // started：是否已进入采集态
    // remoteConfig：本次决策使用的远程配置；拉取耗尽且无缓存时为 null
};

TDEM.addInitListener(listener);
TDEM.removeInitListener(listener);
```

回调覆盖三种结果：成功启动、远端关闭或未抽中、拉取耗尽；若 `id` / `url` 为空导致初始化被跳过，同样会以 `started=false` 回调。实例层另有 `registerInitListener` / `unregisterInitListener`，语义相同；接入方应统一使用上述入口层静态方法，避免 JVM 签名冲突。

自定义事件

Kotlin

```kotlin
tdem?.track(
    "button_click",
    tags = mapOf("button_id" to "submit"),
    properties = mapOf("page" to "/checkout"),
)
```

Java

```
import java.util.HashMap;
import java.util.Map;

Map<String, String> tags = new HashMap<>();
tags.put("button_id", "submit");
Map<String, Object> properties = new HashMap<>();
properties.put("page", "/checkout");

tdem.track("button_click", tags, properties);
```

自定义测速

Kotlin

```
// 直接上报耗时（毫秒）
tdem?.measure("api_latency", 320, tags = mapOf("api" to "/user/info"))

// 或使用计时器
tdem?.startMeasure("render")
// ... 执行操作 ...
val duration = tdem?.endMeasure("render")  // 返回耗时（ms），未找到 start 返回 -1
```

Java

```
// 直接上报耗时（毫秒）
Map<String, String> tags = new HashMap<>();
tags.put("api", "/user/info");
tdem.measure("api_latency", 320, tags, new HashMap<String, Object>());

// 或使用计时器
tdem.startMeasure("render");
// ... 执行操作 ...
long duration = tdem.endMeasure("render");  // 返回耗时（ms），未找到 start 返回 -1
```

`measure` / `endMeasure` 支持省略末尾可选参数的短重载，因此 Java 侧也可写成 `tdem.measure("api_latency", 320)` 或 `tdem.endMeasure("render")`。

日志与异常上报

Kotlin

```kotlin
tdem?.captureMessage("用户完成注册", level = "info", tags = mapOf("step"
to "register"))

try {
    riskyOperation()
} catch (e: Exception) {
    tdem?.captureException(e, tags = mapOf("module" to "payment"))
}
```

Java

```java
Map<String, Object> tags = new HashMap<>();
tags.put("step", "register");
tdem.captureMessage("用户完成注册", "info", tags);

Map<String, Object> crashTags = new HashMap<>();
crashTags.put("module", "payment");
try {
    riskyOperation();
} catch (Exception e) {
    tdem.captureException(e, crashTags);
}
```

`captureMessage` / `captureException` 未提供短重载，Java 侧需显式传出全部参数（`level` 传 `null` 等同于不写该字段）。

用户标识管理

```kotlin
tdem?.setUser("user-456")    // 设置用户
tdem?.clearUser()            // 清除用户
```

隐私标记

通过 `TDEMPrivacyMetadata` 主动声明某个控件为敏感，用于自动识别覆盖不到的场景（例如订单金额、收货地址这类字符串本身无语义特征的文案）。

```
import com.tencent.tdem.common.privacy.TDEMPrivacyMetadata
import com.tencent.tdem.common.privacy.TDEMReplayMasking

// 标记为敏感：行为监控不采集该控件文本，Session Replay 遮罩该控件及其子树
TDEMPrivacyMetadata.setSensitive(orderAmountText, true)
TDEMPrivacyMetadata.setSensitive(orderAmountText, false)    // 取消标记

// 只调 Session Replay 遮罩，不影响文本采集
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.MASKED)
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.UNMASKED)
TDEMPrivacyMetadata.setReplayMasking(qrCodeView,
TDEMReplayMasking.INHERIT)  // 恢复默认判定
```

`TDEMReplayMasking` 三个取值：`INHERIT`（默认，按下面的遮罩判定链走）、`MASKED`（强制遮罩）、`UNMASKED`（显式不遮罩）。

两条通道作用范围与继承语义不同：

通道	影响	对子控件是否继承
`setSensitive`	行为监控文本采集 + Session Replay 遮罩	行为监控不继承，Session Replay 继承
`setReplayMasking`	仅 Session Replay 遮罩	不继承

标记父容器时：Session Replay 会把标记向下传递，**等价于遮罩整棵子树**；但行为监控**只判定被标记的那个控件本身**，子控件文本仍会被采集，需逐个标记。

Session Replay 的遮罩判定按下列次序，命中即返回：

1. 业务敏感标记（`setSensitive`，含父容器继承）：最高优先级，**不可被任何标记撤销**。
2. `sentry-unmask` Tag → 不遮罩
3. `sentry-mask` Tag → 遮罩
4. `setReplayMasking(MASKED)` → 遮罩
5. `setReplayMasking(UNMASKED)` → 不遮罩（次序在显式遮罩之后，但仍受第 1 条压制）。

6. `unmaskViewClasses` 匹配 → 不遮罩
7. `maskViewClasses` 匹配 → 遮罩

因此对已标记敏感的控件调用 `setReplayMasking(UNMASKED)` 不会放开遮罩。`TDEMPrivacyMetadata.isSensitive(view)` / `replayMasking(view)` 可查询当前取值，后者未设置时返回 `INHERIT`。

行为元数据

`BehaviorViewMetadata` 用于在自动识别拿不到稳定语义时，为单个控件补充行为侧元数据。**点击或长按事件仍会正常上报**，这些标记只影响事件属性与下游判定。

Kotlin

```kotlin
import com.tencent.tdem.behavior.BehaviorViewMetadata

// 该控件的点击仍会上报，但不计入「无响应点击」判定
BehaviorViewMetadata.setDeadClickIgnored(buyButton, true)
BehaviorViewMetadata.isDeadClickIgnored(buyButton)    // 查询，未设置返回 false

// 把行为事件与业务反馈键关联
BehaviorViewMetadata.setStruggleFeedbackKey(buyButton, "coupon_refresh")
BehaviorViewMetadata.struggleFeedbackKey(buyButton)   // 查询，未设置返回 null
```

Java

```java
import com.tencent.tdem.behavior.BehaviorViewMetadata;

BehaviorViewMetadata.setDeadClickIgnored(buyButton, true);
BehaviorViewMetadata.isDeadClickIgnored(buyButton);

BehaviorViewMetadata.setStruggleFeedbackKey(buyButton,
"coupon_refresh");
BehaviorViewMetadata.struggleFeedbackKey(buyButton);
```

两个标记各自写入行为事件的 `event_properties`：

方法	写入的事件属性	说明

`setDeadClickIgnored(view, true)`	`dead_click_ignored=1`	该控件的点击不计入「无响应点击」判定；传 `false` 撤销
`setStruggleFeedbackKey(view, key)`	`struggle_feedback_key`	把行为事件与业务反馈键关联；`key` 为空白或 `null` 时撤销

标记挂在 `View` 实例上，内部使用弱引用，控件被回收后自动释放，不需要手动清理。

> ⚠ **注意：**
> 页面 ID 与控件 ID 由 SDK 自动解析（优先取 `resource-id`，取不到时回落控件层级路径），Android 没有手动指定 page ID / element ID 的接口。若只是想在检测阶段跳过一批控件、不写事件属性，应改用 `StrugglePluginConfig` 的 `deadClickIgnoredViewIds` / `ignoreDeadClickSelectors` 配置项。

设备标识管理

```
tdem?.setDeviceId("device-abc")    // 设置设备标识，由宿主提供
tdem?.clearDeviceId()              // 清除设备标识，回落 not_set
```

设备标识由业务侧提供，SDK 不会自行读取 `ANDROID_ID` 等系统标识；未设置或传空白时上报字面量 `not_set`。也可在 `TDEMConfiguration` 的 `deviceId` 中于初始化时一并设置。

全局标签管理

```
tdem?.setTags(mapOf("role" to "admin", "team" to "dev"))   // 设置/追加标签
tdem?.removeTags(listOf("team"))                           // 移除指定标签
tdem?.clearTags()                                          // 清空所有标签
```

页面管理

```
// 手动逻辑页（LIFO，覆盖自动 Activity/Fragment/Compose 页）
tdem?.startPage("checkout")
tdem?.leavePage()    // LIFO pop，栈空时幂等 no-op

// 替换当前逻辑页并发 page_view（navigation_type=replace，RN SPA 切页用）
tdem?.setPage("/modal/confirm", pageTitle = "确认弹窗")
```

上下文与状态查询

除写入接口外，SDK 也提供读取当前运行状态的接口，便于接入自检与问题定位。

Kotlin

```kotlin
tdem?.getTags()                // 当前全局标签快照，Map<String, Any>
tdem?.currentPageUrl           // 当前页面标识
tdem?.getCurrentSessionId()    // 当前会话 ID，会话未就绪返回 ""
tdem?.getCurrentReplayId()     // 当前回放 ID，未启用回放返回 ""
tdem?.userId                   // 当前用户标识，未设置返回 null
tdem?.deviceId                 // 当前设备标识，未设置返回 not_set
tdem?.initialized              // SDK 是否已进入采集态
```

Java

```java
import java.util.Map;

Map<String, Object> tags = tdem.getTags();
String pageUrl = tdem.getCurrentPageUrl();
String sessionId = tdem.getCurrentSessionId();
String replayId = tdem.getCurrentReplayId();
String userId = tdem.getUserId();
String deviceId = tdem.getDeviceId();
```

- `getCurrentSessionId()` 在会话尚未就绪时返回空串；
- `getCurrentReplayId()` 在未启用回放时返回空串；
- `currentPageUrl` 的取值为「手动页面栈顶 > 自动识别的 Activity / Fragment / Compose 页 > `unknown`」。
- `resolveReplayLink(timestampMs)` 按事件时间戳反查所属回放段，返回 `replayId` 与相对段起点的 `offsetMs`，无命中窗口时返回 `null`：

Kotlin

```kotlin
val link = tdem?.resolveReplayLink(eventTimeMs)
if (link != null) {
    println("replay=${link.replayId}, offset=${link.offsetMs}ms")
}
```

Java

```
import com.tencent.tdem.core.replay.ReplayAssociationStore;

ReplayAssociationStore.Link link = tdem.resolveReplayLink(eventTimeMs);
if (link != null) {
    System.out.println("replay=" + link.getReplayId() + ", offset=" +
link.getOffsetMs() + "ms");
}
```

以下两个接口属进阶用法，一般由插件或桥接层使用：

- `isPluginEnabled(pluginEnabled, remoteEnabled)` **做插件挂载的两级决策（远程配置明确关闭时返回** `false`**，否则取本地配置）。**
- `putContextField(key, value)` **向 Protocol v2 批量上下文追加自定义字段（** `key` **空白或** `value` **为空时忽略）。**

挣扎事件上报

```
// 业务主动上报自定义挣扎事件（无需理解或填写 owner）
tdem?.trackStruggle(
    "payment_declined",
    severity = TDEMStruggleSeverity.HIGH,    // LOW(1) / MEDIUM(3) /
HIGH(5)，默认 MEDIUM
    properties = mapOf("reason" to "risk_rejected", "retry_count" to 2),
    tags = mapOf("channel" to "checkout"),
)
```

原始事件上报

```
// 直接上报 TDEMEvent 格式的事件（高级用法，SDK 会自动补齐 page_url、session_id
与 tags）
tdem?.reportTDEMEvent(
    TDEMEvent(
        eventType = EventType.CUSTOM,
        eventCategory = EventCategory.CUSTOM,
        pageUrl = currentPageUrl,
        sessionId = tdem.getCurrentSessionId(),
        data = JSONObject().put("event_name", "my_event"),
```

```
    )
)
```

启动封口

业务就绪时结束启动测量。需先配置 `LaunchPluginConfig(manualEndEnabled = true)`：

```
TDEM.endLaunch()
```

配置修改与销毁

```
tdem?.setUser("new-user")    // 动态修改用户标识
tdem?.destroy()              // 销毁实例
```

版本与日志

通过 `TDEM.VERSION` 可获取 SDK 版本号字符串。日志由全局单例 `TDEMLogger` 控制：

Kotlin

```
import com.tencent.tdem.common.log.TDEMLogger

TDEMLogger.enabled = true             // 全局日志开关，修改时同步下推 native
TDEMLogger.tag = "TDEM-Android"       // Logcat tag，默认 TDEM-Android
TDEMLogger.level = TDEM.LEVEL_DEBUG   // 级别：DEBUG / INFO / WARN /
ERROR，默认 DEBUG
```

Java

```
import com.tencent.tdem.common.log.TDEMLogger;

TDEMLogger.INSTANCE.setEnabled(true);
TDEMLogger.INSTANCE.setTag("TDEM-Android");
TDEMLogger.INSTANCE.setLevel(TDEM.LEVEL_DEBUG);
```

`TDEMLogger` 是 Kotlin `object`，Java 侧通过 `TDEMLogger.INSTANCE` 访问，属性读写为 `getEnabled()` / `setEnabled(...)`、`getTag()` / `setTag(...)`、`getLevel()` / `setLevel(...)`（`enabled` 为普通声明式属性，Java 侧不是 `isEnabled()`）。日志级别常量 `TDEM.LEVEL_*` 也可直接用于 `TDEMConf`

`iguration.Builder.logLevel(...)` 。SDK 初始化时会按配置同步一次日志开关与级别，通常无需手动设置。

环境枚举

用于标识应用部署所属环境，区分不同环境的数据上报、日志采集与监控策略，枚举定义如下：

- production：生产环境。
- development：开发环境。
- gray：灰度环境。
- pre：预发布环境。
- daily：日发布环境。
- local：本地环境。
- test：测试环境。
- others：其他环境（未知值归一于此）。

iOS SDK 接入指引
SDK 集成

最近更新时间：2026-09-28 17:02:02

用户体验监控 iOS SDK 由腾讯云可观测团队提供技术支持。支持 Swift Package Manager 自动集成和手动集成两种方式。

版本升级提醒

因 SDK 中存在缓存数据等，旧版本无法做到对新版本缓存数据的全兼容，**故一般情况下升级后的 SDK 强烈不推荐进行降级处理**，可能存在缓存读取异常的情况。若存在降级的需求，请联系我们。

前提条件

用户体验监控 iOS SDK 要求使用 Xcode 15.3 或更高版本，最低支持 iOS 12.0。

自动集成（推荐）

1. 接入 iOS SDK，支持两种方式接入。
   - 方式1：在 Xcode 中通过 **File > Add Package Dependencies** 添加 SDK 仓库地址，然后在 Dependency Rule 选择 **Up to Next Major Version**，起始版本 `0.2.0` 。

     ```
     https://cnb.cool/tencent-dem/ios-sdk
     ```

   - 方式2：在工程的 `Package.swift` 中声明依赖。

     ```
     dependencies: [
         .package(url: "https://cnb.cool/tencent-dem/ios-sdk.git",
     from: "0.2.0")
     ]
     ```

2. 宿主代码只允许导入这一个模块。

   ```
   import TDEMiOSSDK
   ```

手动集成

若工程不使用 Swift Package Manager，可把 `Frameworks/TDEM.xcframework` 直接拖入工程。

1. 下载用户体验监控 iOS SDK 的 xcframework 文件。

2. 拖拽 `TDEM.xcframework` 文件到 Xcode 工程内，在 **General** > **Frameworks, Libraries, and Embedded Content** 中添加，选择 **Embed & Sign**（SDK 为动态库）。
3. 确认 **Build Settings** > **Swift Compiler – General** > **Objective-C Interop Mode** 为 `objc`（Xcode 默认值，无需改动）。

> ⚠ **注意：**
> 仅当工程显式把 `SWIFT_OBJC_INTEROP_MODE` 设成了 `objcxx`、且存在 `.mm`（Objective-C++）文件时，才会触发编译错误。因为该 xcframework 的 `TDEMiOSSDK-Swift.h` 已烘入 C++ 互操作内联代码；消费侧若为 `objcxx`，`.mm` 文件会按 Objective-C++ 展开这些区块并失败。
> 解法：把消费侧改为 `objc`，或避免用 `.mm` 文件 `@import TDEMiOSSDK`。

SDK 初始化

最近更新时间：2026-09-28 17:02:02

本文将为您介绍如何初始化用户体验监控 iOS SDK。

操作步骤

参考以下代码初始化用户体验监控 iOS SDK，在 App 启动后初始化监控框架，一般推荐在 `application:didFinishLaunchingWithOptions:delegate` 中进行初始化。

1. 在 AppDelegate 实现文件中引入对应的头文件。

Swift

```
import TDEMiOSSDK
```

Objective-C

```
@import TDEMiOSSDK;
```

2. 创建配置对象并启动 SDK。`id` / `url` 为必填，缺失会导致初始化失败（`invalidConfiguration`）；`userId` / `deviceId` 为业务必填，缺失不会导致初始化失败，但会使控制台的用户、设备维度无法聚合（`version` 已自动取 `CFBundleShortVersionString`，可不填）。其余配置项均有默认值，按需补充即可，详见 完整配置项说明 。

Swift

```
let configuration = TDEMConfiguration(
    id: "<project-key>",                                    // 必填：项目标识
    url: URL(string: "https://dem.rumt-zh.com")!            // 必填：采集服务基础地址（国内站）
)
configuration.version = "1.0.0"                             // 建议显式配置：默认取 CFBundleShortVersionString，回落 1.0.0
configuration.env = "production"                            // 运行环境，默认 production
configuration.userId = "<user-id>"                          // 必填：预置用户标识（用户维度聚合）
```

```
configuration.deviceId = "<device-id>"                    // 必填：设备标识（设备异常率统计）
configuration.debug = false                               // 可选：诊断日志开关，默认 false

TDEM.start(configuration: configuration)
```

Objective-C

```
// id 与 url 为必填：项目标识 / 采集服务基础地址（国内站）
TDEMConfiguration *configuration =
    [[TDEMConfiguration alloc] initWithId:@"<project-key>"
                                      url:[NSURL
URLWithString:@"https://dem.rumt-zh.com"]];

configuration.version = @"1.0.0";                         // 建议显式配置：默认取 CFBundleShortVersionString，回落 1.0.0
configuration.env = @"production";                        // 运行环境，默认 production
configuration.userId = @"<user-id>";                      // 必填：预置用户标识（用户维度聚合）
configuration.deviceId = @"<device-id>";                  // 必填：设备标识（设备异常率统计）
configuration.debug = NO;                                 // 可选：诊断日志开关，默认 NO

[TDEM startWithConfiguration:configuration];
```

初始化后即可自动采集崩溃、卡顿、启动、网络等数据，无需额外打点。用户行为监控（`behavior`）与会话回放（`replay`）默认关闭，需显式开启。

> ⚠ **注意：**
>
> - `id` 与 `url` 为必填。`url` 是采集服务基础地址，支持 `http` / `https`（生产环境请使用 HTTPS），SDK 会自行拼接具体协议端点。国内站填 `https://dem.rumt-zh.com`，详见 上报域名。
> - `userId` 与 `deviceId` 需一并传入。二者缺失不会导致初始化失败，但会使控制台无法按用户 / 设备维度聚合：`userId` 为空时该字段不会写入上报体；`deviceId` 为空或空白时上报字面量 `not_set`，设备异常率统计会失真。

- 设备 ID 非常重要，用户体验监控使用设备 ID 来计算设备异常率。请通过 `TDEMConfiguration.deviceId` 或 `TDEM.setDeviceId(_:)` 传入稳定的业务设备标识；**未设置时上报 `not_set`**，按设备维度的异常率统计会失真。SDK 不读取 IDFV 等系统标识，需由业务侧提供。
- 建议在用户授权个人信息保护规则后再初始化 SDK。
- 切勿把真实 project key、采集凭证或生产环境地址提交进代码仓库。

上报域名

请根据项目所在地域选择对应的上报域名。不同站点的数据相互隔离，跨站填写会导致数据无法入库。

站点	上报域名
国内站	`https://dem.rumt-zh.com`
新加坡站	`https://dem.rumt-sg.com`
美国站	`https://dem.rumt-us.com`

接入时填写站点根地址即可，无需拼接路径，SDK 会自行拼接具体协议端点。

完整配置项说明

以下为 `TDEMConfiguration` 的全部配置项，均需在 `TDEM.start(configuration:)` 之前设置。其中 `id` / `url` / `userId` / `deviceId` 需按要求传入，其余为可选配置。

基础配置

下表中 `id` / `url` / `userId` / `deviceId` 为必填，其余为可选（均有默认值或可不配置）。

- `id` 与 `url` 由 SDK 强制校验，缺失会直接初始化失败。
- `userId`、`deviceId` 为业务必填项，缺失虽不影响 SDK 运行，但会导致控制台对应的用户 / 设备维度无法聚合。
- `version` 已默认取 `CFBundleShortVersionString`，通常无需手工传入。

配置项	说明
`id`	**必填**。项目标识，对应请求体中的 `project_key`。为空会导致初始化失败（`invalidConfiguration`）。
`url`	**必填**。采集服务基础地址，需为 `http` / `https` 且带 host，SDK 会自行拼接具体协议端点（如 `/api/v1/config`、`/api/v1/collect/events`）。国内站填 `https://dem.rumt-zh.com`，详见「上报域名」。

`version`	应用版本号，对应上报体的 `app.version`，用于版本对比与版本分布分析。**默认自动取** `CFBundleShortVersionString`，取不到或为空时回落 `1.0.0`，因此可不手工传入；若业务版本号与 Bundle 版本不一致，建议显式配置。
`env`	运行环境标识，允许值 `production` / `development` / `gray` / `pre` / `daily` / `local` / `test` / `others`，默认 `production`；未知值（含 `staging`）归一为 `others`。
`userId`	**必填。**业务用户标识，对应上报体的 `context.user_id`；不填时该字段不会写入上报体，控制台无法按用户维度聚合分析。也可在初始化后调用 `TDEM.setUser` 动态设置。
`deviceId`	**必填。**宿主提供的设备标识（`String?`）。**由业务侧设置，SDK 不会自动生成 IDFV 或其他系统设备标识**；未设置或传空白时，上报字面量 `not_set`，按设备维度的异常率统计会失真。也可在运行期调用 `TDEM.setDeviceId(_:)` 设置、`TDEM.clearDeviceId()` 清除。
`defaultTags`	全局默认标签（`[String: String]`），注入每个事件。也可通过 `TDEM.setTags` 增删。
`debug`	诊断日志开关，默认 `false`。开启后输出脱敏的本地诊断日志，不打印事件正文。
`requestCaptureEnabled`	是否抓取事件与回放上报的 HTTP 请求 / 响应原始报文用于本地排查，默认 `false`。与 `debug` 相互独立，仅供测试环境使用。
`requestCaptureDirectory`	原始报文落盘目录。**必须与** `requestCaptureEnabled` **同时设置才生效**，二者缺一则不抓取。

初始化时 SDK 会先请求 `POST {url}/api/v1/config` 拉取远程配置，仅在远程返回 `enabled: true` 且通过采样时才启动采集；拉取失败时沿用本地缓存，无缓存则保守不上报（进入 `suppressed` 状态）。需要感知结算结果时，可实现 `TDEMInitListener` 并通过 `TDEM.addInitListener` 注册。

崩溃监控（crash，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。开启后自动捕获崩溃，并接收 `TDEM.captureException` 上报的已捕获异常。

网络监控（network，默认开启）

配置项	说明

`enabled`	总开关，默认 `true` 。
`reportAllSuccessfulRequests`	是否上报全部成功请求，默认 `false` （仅慢成功与错误 / 取消上报）。
`slowThresholdMs`	慢请求阈值（ms），默认 1000。
`allowedHosts`	域名白名单。非空时仅采集命中白名单的请求；匹配规则为精确域名或其子域（填 `example.com` 可命中 `a.example.com` ）。
`deniedHosts`	域名黑名单。仅在 `allowedHosts` 为空时生效，命中即不采集。
`additionalSensitiveHeaderNames`	在默认敏感头之外追加需要脱敏的请求头名（不区分大小写）。

SDK 自身与采集服务之间的请求（与 `url` 同源）会被自动排除，不会产生递归上报。

启动监控（launch，默认开启）

配置项	说明
`enabled`	是否启用，默认 `true` 。
`manualEndEnabled`	是否等待业务调用 `TDEM.endLaunch()` 才封口，默认 `false` （首个页面展示后自动上报）。

冷启动 / 温启动的慢启动判定阈值由 SDK 内部维护（默认4000ms / 2000ms），接入方不可修改。

卡顿监控（stall，默认开启）

配置项	说明
`enabled`	总开关，默认 `true` 。
`stallThresholdMs`	卡顿判定阈值（ms），默认1000，取值需大于0且小于5000，否则初始化失败（ `stall_invalid_configuration` ）。

除卡顿外，SDK 同时检测主线程无响应（阈值5000ms，内部固定），两类事件都会采集代表性堆栈与掉帧指标。

用户行为监控（behavior，默认关闭）

配置项	说明
`enabled`	总开关，默认 `false` （合规整改起默认 opt-in，需显式开启）。

用户挣扎检测（struggle，默认开启）

配置项	说明
`enabled`	总开关，默认 `true`。
`clickRulesEnabled`	是否检测 rage / dead / error click，默认 `true`。
`formRulesEnabled`	是否检测 long focus，默认 `true`。
`navigationRulesEnabled`	是否检测 back-forward，默认 `true`。

规则阈值（rage 1s/5次、dead/error 3s、long focus 20s、back-forward 10s/3次）与 Android 对齐，SDK 内部写死，接入方不可修改。如需精确忽略某个控件的 dead click，可调用 `TDEMBehaviorMetadata.setDeadClickIgnored(_:for:)`。

Session Replay 会话录屏（replay，默认关闭）

配置项	说明
`enabled`	总开关，默认 `false`。
`sessionSampleRate`	全量会话采样率（0.0-1.0），默认 0.1。
`errorSampleRate`	错误触发采样率（0.0-1.0），默认 1.0。
`quality`	录制质量预设（`TDEMReplayQuality.low` / `.medium` / `.high`），默认 `low`。
`maskAllText`	是否遮罩所有文本内容，默认 `false`。
`maskAllImages`	是否遮罩所有图片内容，默认 `false`。

默认仅遮罩输入类控件与显式标记为敏感的内容；也可通过 `TDEMPrivacyMetadata.setReplayMasking(_:for:)` 对单个 View 强制遮罩或放开。`sessionSampleRate`、`errorSampleRate` 超出 [0, 1] 会导致初始化失败（`replay_invalid_configuration`）。

Web SDK 接入指引
SDK 集成

最近更新时间：2026-09-28 17:02:02

用户体验监控 Web SDK 支持 npm 安装（推荐）与手动拷入构建产物两种方式。

npm 安装（推荐）

1. 安装 tdem-web-sdk 包。

```
npm install tdem-web-sdk@^1 --save
```

2. 安装后，在项目入口文件中引入并初始化。

```
import TDEM from 'tdem-web-sdk';
```

3. 引入后即可通过 `new TDEM(config)` 创建实例。

手动接入（备选）

若项目不使用打包器，可将 npm 包内的构建产物 `lib/tdem.min.js` 拷入项目静态目录，通过 `<script>` 标签引入（需置于业务脚本之前）：

```
<script src="./libs/tdem.min.js"></script>
```

引入后全局会暴露 `TDEM` 构造函数，初始化方式与 npm 引入一致。