**如何打造一款标准的 JS SDK ？**

**简介：** 本文通过 SDK 的实际案例，介绍如何基于 JavaScript 来开发 SDK，并分享一些设计原则以及实现技巧。



岳鹰全景监控，是阿里UC官方出品的先进移动应用线上监控平台，为开发者及企业提供一套完整的移动应用线上质量监控解决方案。岳鹰WEB前端监控，可实时监控页面性能、JS异常、资源加载异常、API成功率、自定义错误等异常情况。本文通过岳鹰前端监控SDK的实际案例，介绍如何基于JavaScript来开发SDK，并分享一些设计原则以及实现技巧。

**SDK 是什么**

SDK全称是“Software Development Kit”，直译就是软件开发工具集。说的再通俗点就是一个面向开发者，针对特定领域的软件包。比如Java SDK（JDK），就是一个Java领域的软件包。基于它，开发人员就可以快速构建自己的Java应用。比较规范的SDK一般都会包含若干的API、开发工具集和说明文档。

JS SDK也无外于此，不过鉴于JS语言本身的特性，基于Ta封装的SDK更多常见于UI组件库、统计分析、web服务接口封装、前端稳定性和性能监控等场景。岳鹰前端监控SDK[1]即属于前端稳定性和性能监控这一领域范畴的SDK。

**设计原则**

如何设计SDK，其实更多取决于你的场景，或者SDK最终的用途。比如实现一个给网页调用的SDK与用于服务端的SDK就有明显的差异，但这之间确实存在着一些共通的原则，或者方法论：

* 最小可用性原则，即用最少的代码，如无必要勿增实体。
* 最少依赖原则，即最低限度的外部依赖，如无必要勿增依赖。

进一步阐述，即我们打造的SDK要符合以下的要求。

**满足功能需求**

SDK一般都是偏于面向某个领域，所以，同时在设计和实现的时候明确职责和边界很重要，同时还应该足够精简，专注领域内的业务。

**足够稳定**

* 绝不能导致宿主应用崩溃，这是最基础也是最严格的要求。
* 较好的性能，比如SDK体积应尽量小，运行速度尽量快。
* 可测试，保障每一次变更。
* 向后兼容，不轻易出现 Breakchange。

**少依赖，易扩展**

* 最小程度的第三方依赖，尽可能自行实现，确实无法避免则最小化引入。
* 插件化，最大限度支持扩展。
* Hook机制，满足个性化诉求。

**如何实现**

下面我们将通过剖析岳鹰前端监控SDK的设计过程，来看看上述的设计原则是如何应用到实际的开发过程中的。

**明职责，定边界**

前面章节提到，岳鹰前端监控SDK是前端稳定性和性能监控的SDK，主要面向前端H5领域。因此，稍加分析即可得出以下结论：

* 前端领域，稳定性方面主要的关注点  
  + JS异常
  + 资源加载异常
  + API请求异常
  + 白屏异常
* 性能方面，核心的关注点  
  + 白屏时间
  + 可交互时间（TTI）
  + 首屏时间
  + FP / FMP / FCP 等

上述监控内容实际上都相对独立，因此我们可以把它们横向划分为如下几大部分：



明确了SDK的边界以及各部分的职责，结合前端监控的特性，我们可以开始设计SDK的整体框架了。

**筑框架，夯基础**

俗话说千里之行始于足下，因此筑牢基础十分重要。总得来说，我们需要做好下面几点：

**确定SDK的引用形式**

SDK整体而言是一个大模块，前端模块有多种表现形式：ES Module、CommonJS、AMD/CMD/UMD，而在引用方面则大体分 CDN和 NPM两种。即无论我们实现的是哪种形式的模块，最终都是通过CDN或者NPM的方式提供给用户引用。

// ES Module

import wpkReporter from 'wpkReporter'

// CommonJS

const wpkReporter = require('wpkReporter')

// AMD,requireJS引用

require.config({

paths: {

"wpk": "https://g.alicdn.com/woodpeckerx/jssdk/wpkReporter.js",

}

})

require(['wpk', 'test'], function (wpk) {

// do your business

})

乍看有点眼花，但事实上今时今日的前端工程领域，已有很多利器可以帮助我们达到目的。比如webpack，通过简单的配置就可以构建出一个UMD的bundle。

// webpack.config.js

module.exports = {

output: {

filename: '[name].js',

path: `${\_\_dirname}/dist`,

globalObject: 'this',

library: '[name]',

libraryTarget: 'umd'

}

}

综上，我们可以通过webpack将SDK构建为一个UMD bundle，这样可以自动适配所有形式的模块。同时我们也将同时提供CDN和NPM两种引用方式，给用户更多选择。

**确定SDK的版本管理机制**

现有较成熟的版本管理机制当属语义化版本号[2]，表现形式为 {主版本}.{次版本}.{补丁版本}，简单易记好管理。

一般重大的变更才会触发主版本号的更替，而且很可能新旧版本不兼容。次版本主要对应新特性或者较大的调整，因此也有可能出现breakchange。其他小的优化或bugfix就基本都是在补丁版本号体现。

看到此处，是否有点似曾相识的感觉？没错，所有NPM模块都遵循语义化版本规范，因此结合第一点，我们可以将SDK初始化为一个NPM模块，结合webpack的能力就可以实现基础的版本管理及模块构建。

**确定SDK的基础接口**

接口是SDK和用户沟通的桥梁，每一个接口对应着一个独立的SDK功能，并且有明确的输入和输出。我们可以先来看看岳鹰前端监控SDK的核心接口有哪些？

// 上报相关

wpk.report(logData)

wpk.reportJSError(error)

wpk.reportAPIError(apiData)

// 配置变更

wpk.setConfig(data)

// SDK诊断

wpk.diagnose()

// 添加插件

wpk.addPlugin(plugin)

总结接口的设计原则，如下：

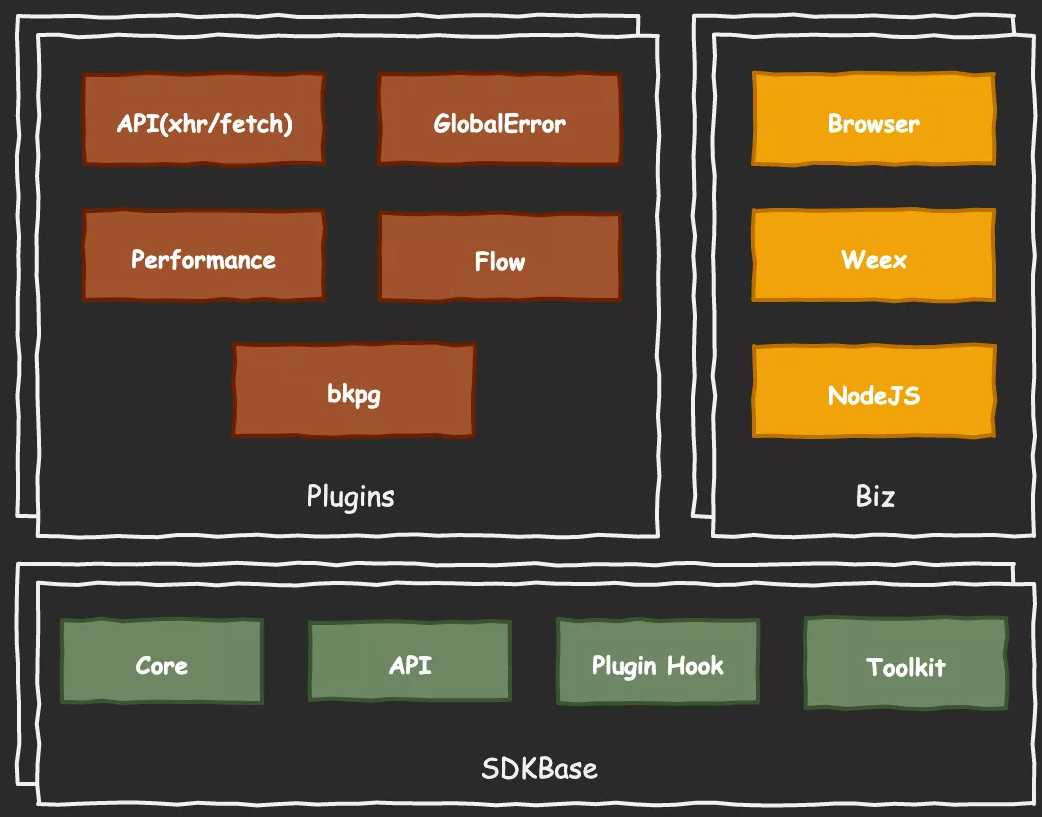
* 职责单一
  + 一个接口只做一件事情
* 命名简单清晰，参数尽量少但可扩展
  + 好的接口命名就是最好的注释，一看即明其用处
  + 参数尽可能适用Object封装
* 做好参数校验和逻辑保护

**领域分析，模块划分**

定边界的时候，我们已经清楚划分了SDK的几个关键的部分：全局异常、API异常、页面性能和白屏，实际上监控SDK通常也会内置对页面流量的监控，以方便用户对异常的影响面做出评估。这几个核心的关键组成部分，每一块都对应一个专业的领域，因此对应到SDK也是每一个独立的模块。

除了这些核心的偏领域的模块，SDK还需要有更基础的与领域无关的模块，包括SDK内核(构造方法、插件机制、与下游服务的交互、上报队列机制、不同环境的管理等等)和工具类库。

我们可以先看一下岳鹰前端监控SDK最后的整体模块划分：



* SDK底层提供基础的能力，包括上面提到的内核、插件机制的实现、工具类库以及暴露给用户的基础API。
* 可以看到，我们前面提到的所有模块都以插件的形式存在，即各领域的功能都各自松散的做实现，这样使得底层能力更具通用性，同时扩展能力也更强，用户甚至也可以封装自己的插件。
* Biz部分更多是对于不同宿主环境的多入口适配，当前支持浏览器、Weex以及NodeJS。

**测试覆盖，线上无忧**

SDK是一个基础服务，相对于前台业务而言可能更底层些。其影响面跟应用的范围是正比的关系，更多的用户意味着更大的责任。所以SDK的质量保障也是很重要的一个环节。

岳鹰前端监控SDK的质量保障策略很简单，只有两条：

* 核心接口100%的单元测试覆盖率
* 发布卡点：再小的版本发布也需要走集成测试回归

事实上，除了核心接口，工具类库的所有功能我们都实现了100%的单元测试覆盖，我们采用的前端测试工具是轻量好用的Jest[3]。

// 小巧精炼的 Jest，笔者力荐

test('isError: real error', function () {

var err = new Error('this is an error')

expect(util.isError(err)).toBeTruthy()

})

**细节打磨，极致体验**

* 快捷引入
  + 极尽所能提高用户引用的效率
  + 一行代码，快速引入，享用监控全家桶功能

<script>

!(function(c,i,e,b){var h=i.createElement("script");var f=i.getElementsByTagName("script")[0];h.type="text/javascript";h.crossorigin=true;h.onload=function(){c[b]||(c[b]=new c.wpkReporter({bid:"dta\_1\_203933078"}));c[b].installAll()};f.parentNode.insertBefore(h,f);h.src=e})(window,document,"https://g.alicdn.com/woodpeckerx/jssdk/wpkReporter.js","\_\_wpk");

</script>

* 动态采样
  + 即通过云端下发数据采样率的方式，控制客户端上报数据的频率
  + 更好的保护监控下游
* 自我诊断
  + 除了接口，SDK整体对用户而言就是一个黑盒，因此用户在遇到问题时很容易蒙圈 (如：为啥没有上报数据)
  + SDK可以提供一个自我诊断的接口，快速排除基础问题。  
    比如，SDK是否已正常初始化、关键参数是否正常设置等。
  + 增加调试模式，输出更详细的过程日志，方便定位问题
* 渐进式的指引文档
  + 图文并茂，循序渐进
  + 入门，一步步引导用户初识SDK，领略概貌，学会基本的使用
  + 进阶，安利SDK的深度用法，帮助用户更好的使用SDK

**结语**

实际在SDK的设计和开发过程中，要处理的问题还远不止本文所述的内容，比如NPM模块开发时本地如何引用，构建的bundle大小如何调优等等。不过还是希望阅完此文，对你有所启发。同时文中若有不对之处，还望不吝赐教。

相关链接

[1]https://yueying-docs.effirst.com/api-usage.html

[2]https://semver.org/

[3]https://jestjs.io/