**通用H5架构设计**

**(内部资料 请勿外传)**

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **时间** | **修订内容** | **修订人** | **审批人** |
| 1 | | 2017.7.7 | 初建 | 王龙 |  |
| 2 | |  |  |  |  |
|  | |  |  |  |  |

目录

目录

[目录 3](#_Toc487213182)

[1 背景 5](#_Toc487213183)

[2 名词和术语解释 5](#_Toc487213184)

[2.1 JSSDK 5](#_Toc487213185)

[2.2 SDK Adapter 5](#_Toc487213186)

[3 总体设计 5](#_Toc487213187)

[3.1 设计目标 5](#_Toc487213188)

[3.1.1 标准化 5](#_Toc487213189)

[3.1.2 可配置性 5](#_Toc487213190)

[3.1.3 一次开发多处运行 6](#_Toc487213191)

[3.2 设计思想 6](#_Toc487213192)

[3.2.1 组件化 6](#_Toc487213193)

[3.3 系统架构 7](#_Toc487213194)

[3.4 H5 UI 8](#_Toc487213195)

[3.4.1 总体结构 8](#_Toc487213196)

[3.4.2 根组件 8](#_Toc487213197)

[3.4.3 UI组件 9](#_Toc487213198)

[3.4.4 功能组件 9](#_Toc487213199)

[3.4.5 Page组件 9](#_Toc487213200)

[3.5 JS SDK 9](#_Toc487213201)

[3.5.1 流程 9](#_Toc487213202)

[3.5.2 接口 9](#_Toc487213203)

[3.6 Adapter 10](#_Toc487213204)

[4 开发 10](#_Toc487213205)

[4.1 构建工具 10](#_Toc487213206)

[4.2 ES6 module 10](#_Toc487213207)

[5 部署 10](#_Toc487213208)

[6 系统性能 11](#_Toc487213209)

[7 系统开发环境 11](#_Toc487213210)

# 背景

基于产品标准化的思维，需要为每一类产品提供出标准的H5界面，以便用户进行设备的调试和控制。

同时为更好地服务于 DNA kit 导入的产品，实现对客户的快速服务（独立APP、京东、国美、苏宁等），提升客户满意度，必须实现一套快速开发的 UI 机制。

# 名词和术语解释

## JSSDK

DNA kit平台提供的一套标准JS接口，为上层 UI 提供统一的调用接口，屏蔽底层平台的差异性

## SDK Adapter

JSSDK在不同平台（京东、国美、苏宁等）上的实现。

# 总体设计

## 设计目标

### 标准化

为每一类设备开发标准版的H5 UI，H5 UI的开发是基于标准化的参数、交互和互斥关系的。

### 可配置性

每一类产品的标准UI需要提供一定的可配置性，比如某可选参数的是否启用、某参数枚举值的增加删除（如风速的档位）等等。

标准UI应根据产品profile一定程度上动态生成界面。

### 一次开发多处运行

标准UI（或者其他使用JS SDK开发出的UI），应可以经过简单打包处理，不需更改代码即可上其他平台使用。

## 设计思想

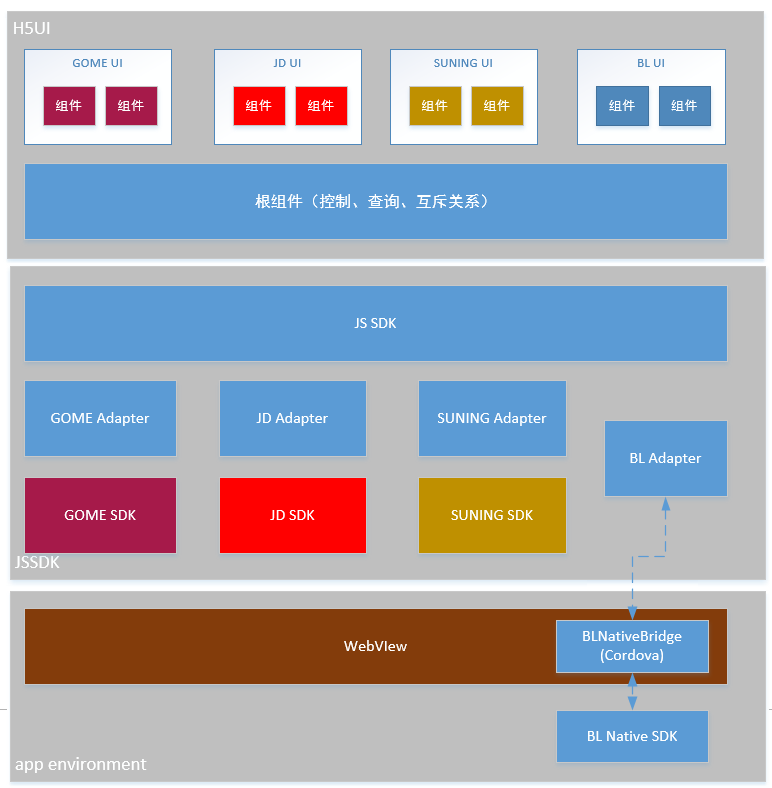
### 组件化

组件即将一段或几段完成各自功能的代码段封装为一个或几个独立的部分。

组件可接受参数，并输出HTML DOM节点，在浏览器中渲染显示。

H5 UI中所有的部分都可以抽象为组件，如UI组件、功能组件、Page组件等等，各种组件的互相组合便成为H5 APP（H5 UI）

## 系统架构

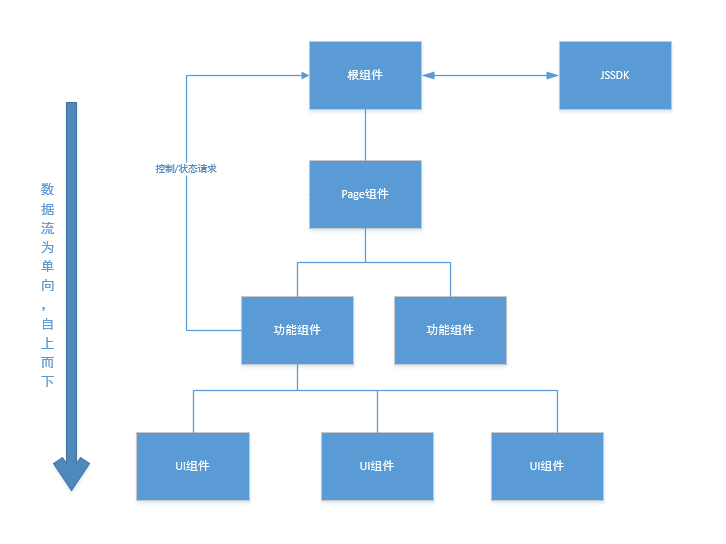


架构图

H5UI部分，各平台的H5UI只有界面显示上的差别，功能上都是通用的。所以在不考虑各平台的UI设计标准的情况下，这些UI都是可以在任意平台上使用的。

## H5 UI

### 总体结构



### 根组件

根组件储着整个程序的状态，并提供跨平台跨设备的基础逻辑。如：设备控制、状态查询、互斥逻辑等等。

所有其他组件需要进行设备状态的操作时，都必须发送消息给根组件，由根组件去调用JSSDK，获得数据后传递给下层组件。

根组件的定义可以有多个，如同步根组件（每次操作都会有loading界面出现）、异步根组件（操作完成界面立刻改变，不理会设备实际状态）。但在运行时有且只能有一个根组件的存在。

**根组件是跨平台和跨设备类型的**。

### UI组件

从UI视觉的角度开发的组件，注重于显示效果，无具体的业务逻辑。如button、loading（加载框）、dialog（消息框）、toast等。

UI组件是高于设备类型的，在一个平台中是通用的。

### 功能组件

功能组件代表着设备中的一个功能，一般由一个或者多个的UI组件组成。功能组件中还有着该功能的业务逻辑和互斥逻辑。

基本设计原则为：一个参数代表着一个功能组件。功能组件能根据参数中枚举值动态组合UI组件。

功能组件接受的参数为：

参数名，如：pwr

参数值，如：1

禁用值，如: [2,4] //2,4这两个值不可用

功能组件为设备类型特有的，一般不能跨设备类型而使用。

### Page组件

多个功能组件和UI组件便组成了一个Page。Page的本身也是一个组件。

界面组件最重要的功能提供布局逻辑。

### 平台特定组件

平台特定组件是一类特殊的组件，在每个平台（JD、AL、国美）中会提供些平台特定的功能，如云定时、云菜谱、历史数据等等。

这些功能和接口数量多、特异性大，adapter无法进行适配，也无法在JS SDK接口中体现。所以针对每个平特的特有功能，对其开发特定的组件，如JD云定时组件、JD云菜谱组件、国美云定时组件等等。

在打包部署时根据所要部署的平台加入对应的组件即可。

## JS SDK

### 流程

JSSDK在原来的sdk基础上进行简化，会提供一个统一的入口方法，删除繁琐的多步事件依次触发的流程。

JSSDK中会自动引入（require或者import）对应的adapter文件，将adapter对开发者屏蔽

### 接口

JSSDK提供的所有接口统一返回promise对象

1. *Init*
2. *setDeviceStatus*
3. *getDeviceStatus*

## Adapter

Adapter使用各平台提供的SDK，来实现JS SDK中定义的接口。

适配原则：

1. 设备控制类接口一一适配，如个别特殊接口为某个平台特有，那么其他平台的adapter中则提供一个空方法
2. 平台特有接口，如云菜谱、云定时、历史数据等接口。Adapter中不做适配也不体现。我们为这些功能在每个平台上进行定制开发，开发出各平台通用的组件。

## 互斥逻辑

### 定义

每个设备的互斥条件是跨平台的（一个产品互斥条件是与平台无关的），因此我们在JS SDK中提供一个通用接口:getMutex。

getMutex(status)

功能: 根据当前设备状态（快照），返回被禁用的参数和其被禁用的值。

输人项:

* status: 当前设备状态（快照）

返回：

一个对象，代表被禁用的功能。如{ WdSpd:[1,6]}代表WdSpd的1和6不可用（风速的强劲和静音不可用）

### 使用

根组件在设备状态发生变化时会自动调用getMutex方法，将其返回值传递给对应的功能组件。如{ WdSpd:[1,6]}，根组件会将[1,6]传递给“风速调节组件”，让其禁用或者屏蔽“静音”和“强劲”这两个选项

### 默认实现

该接口的默认实现是基于解析互斥配置文件“mutex.json”,该配置文件以json方式提供出设备的互斥条件。格式如下：

{

param1: {

val1: {expression}, //在expression运算值为true的情况下，val1值不可用

val2: {expression},

},

param2: {

val1: {expression},

val2: {expression},

},

}

以格力空调的互斥条件举例：

{

WdSpd: {

'6': {"Mod==='3'||Mod==='0'||SwhSlp==='1'"},

//自动、送风模式或者睡眠开启，风速中的“强劲”不可用

},

AssHt: {

'1': {"Mod!=='4'"},

//除制热模式，辅热开启功能都不可用

},

}

Expression是一个简单的JS表达式，getMutex方法会使用eval()或者new Function()这些途径在一个特定的context下（沙箱）执行。

如果正在开发的设备与标准版设备的互斥条件有稍许不同，开发者只要更改“mutex.json”配置文件即可。

### 自定义实现

如果设备互斥功能过于复杂，开发者只要根据接口文档，覆盖该方法的实现即可。如：

JSSDK.getMutex = function(){

//code

}

# 开发

## 构建工具

通用UI要适配多个平台，逻辑比较复杂，灵活的较大。原生的JS+HTML不能满足开发需求，需要引入构建工具（webpack、gulp等等）

## ES6 module

借助构建工具可以使用 ES6 module的语法，避免了HTML中js文件引用的过多的问题。同时可以将多个js打包成一个，减少http请求。

# 调试

UI 系统提供了一个虚拟设备，以 adapter 的形式提供。

在构建工具的配置文件中，开发模式设置为debug即可。

虚拟设备读取 profile 并模拟真实设备，用户通过 JSDK 和虚拟设备交互，可以获得和调试真实设备相同的效果。它的主要优势是可以在 PC 上用现代浏览器来调试 UI 代码，能够借助现代浏览器提供的强大调试工具，如 firebug、DOM inspector 等，极大的简化开发流程。

虚拟设备暴露接口，用户可以选择让设备：

* 离线
* 对指令返回错误信息
* 不响应用户指令

这样能够调试程序对错误的处理。

# 部署

使用我们自己开发的构建工具插件，将使用到的adapter、html、js打包并压缩成为每个平台所需要的格式，并生成在特定的目录下。

开发者需要将其手动部署到各大平台。

# 系统性能