**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

很多web系统只是在用各种方式（折线图、柱状图、表格）显示服务器或者远程接口中的数据。

鉴于这类系统的共同点，本文档提供一种将数据层与UI层隔离的架构，由此获得UI的高度可复用性和可配置性。

# 名词和术语解释

## 展示型（Presentational）组件

展示型组件通常不知道数据如何获取，更不知道怎么去操作这些数据，他们一般只负责页面的渲染，把数据放到对应的位置。

* 关注UI呈现
* 通过接口或者参数被动接收渲染UI所需要的数据
* 不知道数据如何加载
* 不依赖于系统的其余部分

## 容器型（Container）组件

容器型组件通常会负责和服务端的沟通，还有一些业务逻辑的处理。他们通常只负责获取数据，处理数据，处理状态，但一般不知道如何去展示页面。

* 为展示型组件提供数据与行为。比如：定时刷新、增加监听等
* 通常是有状态的，往往被作为数据源使用。
* 在本系统中，都由框架统一生成

# 总体设计

## 设计思路与原则

### 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。

### 可配置性

每个展示型组件要包括更可能多的配置项（针对于显示的），如大小、颜色、背景、分组等。这样才能保持其高度的跨项目的可复用行

### 单一数据源

为整个应用的 state 被储存在一棵 object tree 中。

这让应用开发变得非常容易和信息。来自服务端的 state 可以在无需编写更多代码的情况下被序列化并注入到客户端中。

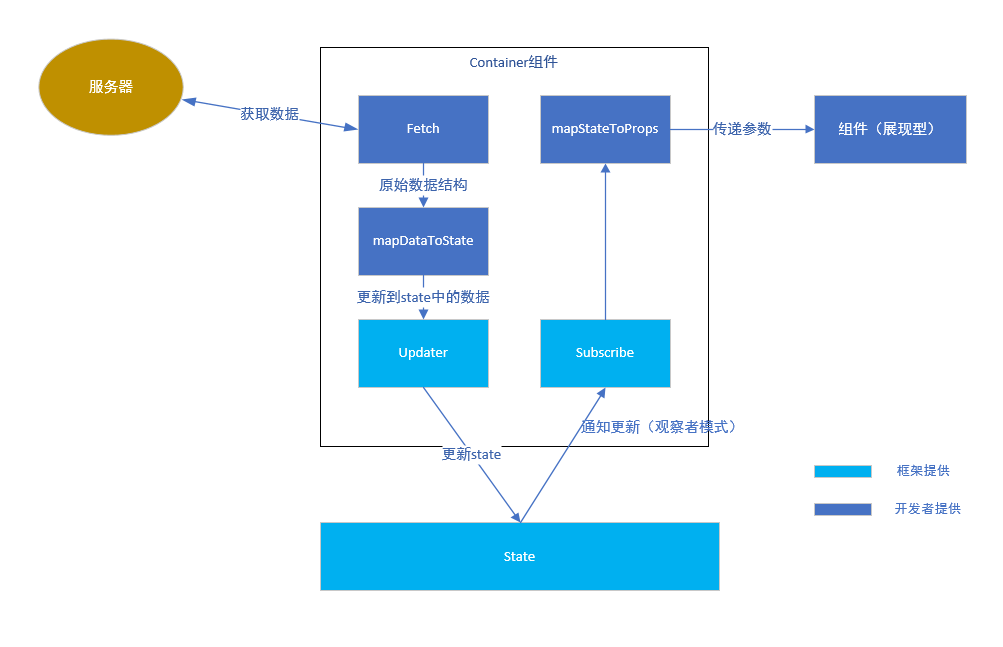
由于是单一的 state tree ，调试也变得非常容易。在开发中，你可以把应用的 state 保存在本地，从而加快开发速度。

此外，受益于单一的 state tree ，使用长连接如websocket，接收服务器的主动推送更新界面也变得轻而易举。

### State 是只读的

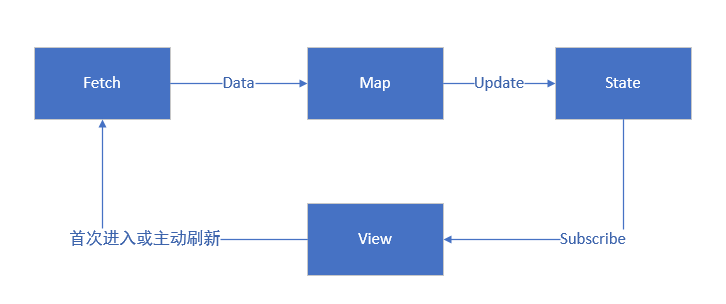
State无法被代码直接修改，只能通过系统提供的统一接口进行修改。

## 系统架构



架构图

## 系统流程



## Container Factory

Factory方法是系统提供给开发者生成Container组件的方法。接受参数如下:

展现型组件

* 展示型（Presentational）组件
* Fetch方法 （获取远程数据的方法）
* MapDataToState方法 （将数据转换为存入state tree的结构）
* MapStateToProps方法 (将state中的数据转换成组件所需的props)

## MapDataToState

MapDataToState在对应的Fetch方法返回成功后，被系统自动调用，同时注入参数：

* 当前state tree
* 对应的Fetch方法的结果

MapDataToState的返回值

* 新的state tree 对象

系统会将state tree进行更新，并通知所用的监听的组件（Container组件）

## Container组件

系统中由Factory生成的contianer组件统一提供以下几个功能：

1. Reload按钮 （重新发送fetch请求）
2. 定时刷新 （周期行发送fetch请求）
3. 重新加载组件后是否重新fetch （页面切换的场景）

# 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。

# 部署

# 系统性能