飞信客户端V7架构技术方案

**桌面团队web研发项目组**

|  |  |
| --- | --- |
| *版本:* | *1.00* |
| *修改日期:* | *2019-02-11* |
| *状态:* | ***编写中*** */ 已提交 / 已发布* |
| *评审：* |  |
| *批准：* |  |

**目录**

[飞信客户端V7架构技术方案 1](#_Toc524338598)

[桌面团队web研发项目组 1](#_Toc524338599)

[1 文档介绍 4](#_Toc524338600)

[1.1 文档范围与目的 4](#_Toc524338601)

[1.2 术语和缩写 4](#_Toc524338602)

[2 web 架构设计 6](#_Toc524338603)

[2.1 web 架构简述 6](#_Toc524338604)

[2.1.1 Web系统架构图 6](#_Toc524338605)

[2.1.2 WebIM架构设计特点 7](#_Toc524338606)

[2.1.2.1 Component 组件化开发 7](#_Toc524338607)

[2.1.2.2 Communication通信 7](#_Toc524338608)

[2.1.2.3 User interface交互设计模式 7](#_Toc524338609)

[2.1.2.4 Agent代理 7](#_Toc524338610)

[2.1.2.5 WebSocket协议 7](#_Toc524338611)

[2.1.2.6 Process独立进程 7](#_Toc524338612)

[2.1.3 WebIM技术特点 8](#_Toc524338613)

[2.1.3.1 概述 8](#_Toc524338614)

[2.1.3.2 兼容 8](#_Toc524338615)

[2.1.3.3 技术关键字 8](#_Toc524338616)

[2.1.3.4 数据 8](#_Toc524338617)

[2.1.3.5 交互视觉 8](#_Toc524338618)

[2.1.3.6 版本控制 8](#_Toc524338619)

[2.1.3.7 通知 9](#_Toc524338620)

[2.1.3.8 消息 9](#_Toc524338621)

[2.1.3.9 预加载 9](#_Toc524338622)

[2.2 web业务模块 9](#_Toc524338623)

[2.2.1 设计说明 9](#_Toc524338624)

[2.2.2 登录模块 10](#_Toc524338625)

[2.2.3 好友列表模块 11](#_Toc524338626)

[2.2.4 群列表模块 12](#_Toc524338627)

[2.2.5 最近联系人模块 13](#_Toc524338628)

[2.2.6 会话业务模块 14](#_Toc524338629)

[2.2.7 消息订阅模块 15](#_Toc524338630)

[2.2.8 个人通信录模块 16](#_Toc524338631)

[2.2.9 企业通信录模块 17](#_Toc524338632)

[2.2.10 profile模块 18](#_Toc524338633)

[2.2.11 群发助手模块 19](#_Toc524338634)

[2.2.12 搜索模块 20](#_Toc524338635)

[2.3 技术特性 20](#_Toc524338636)

[2.3.1 V7架构设计目标 21](#_Toc524338637)

[2.3.2 Angular 22](#_Toc524338638)

[2.3.3 Webpack 23](#_Toc524338639)

[2.3.4 IndexedDB 27](#_Toc524338640)

[2.3.5 localStorage 28](#_Toc524338641)

[2.3.6 ES6 29](#_Toc524338642)

[2.3.7 CMD 30](#_Toc524338643)

[2.3.8 CS架构 30](#_Toc524338644)

[2.3.9 WebSocket 31](#_Toc524338645)

[2.3.10 轮询 32](#_Toc524338646)

[2.3.11 长连接 33](#_Toc524338647)

[2.3.12 浏览器引擎 33](#_Toc524338648)

[2.3.13 Map的内存分配机制分析 33](#_Toc524338649)

# 文档介绍

## 文档范围与目的

本文档描述了飞信WEB版-V7架构设计的原因、目的、路线，将V7架构分解，并分别进行其概要设计，对V7版本的开发，以及飞信web架构的发展过程具有指导作用，并且对新架构下各开发团队的日常开发工作具有指导作用。

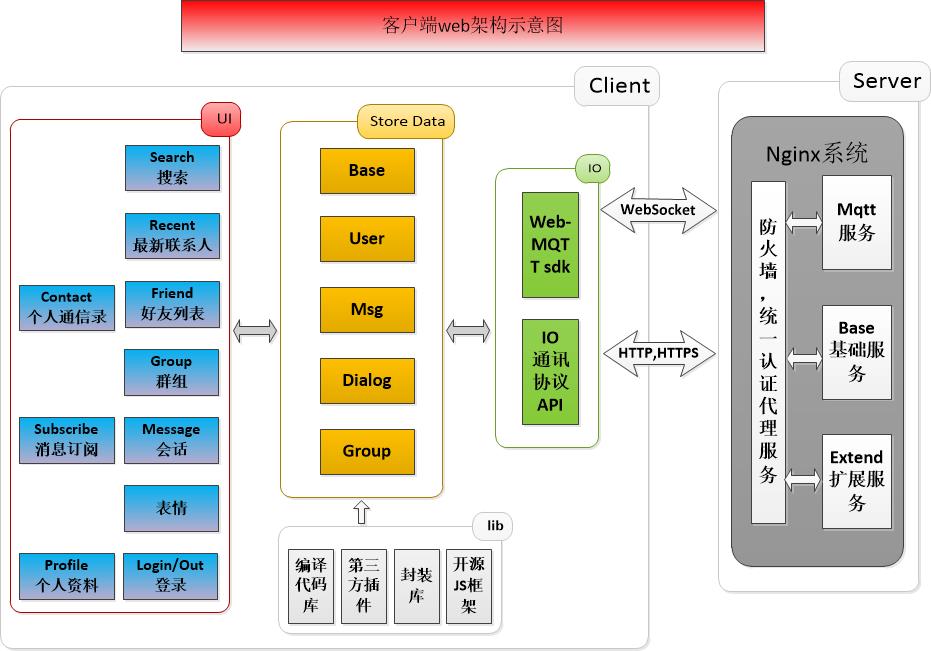
## 术语和缩写

|  |  |
| --- | --- |
| **术语、缩写** | **解释** |
| SDK | 软件开发工具包（外语首字母缩写：SDK、外语全称：Software Development Kit）一般都是一些软件工程师为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件时的开发工具的集合。 |
| JAVASCRIPT | JavaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在[HTML](https://baike.baidu.com/item/HTML" \t "_blank)（标准通用标记语言下的一个应用）网页上使用，用来给HTML网页增加动态功能。 |
| MVVM | C++是C语言的继承，它既可以进行C语言的过程化程序设计，又可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计. |
| 框架 | 网页中经常使用的页面设计方式，框架的作用就是把网页在一个浏览器窗口下分割成几个不同的区 域，实现在一个浏览器窗口中显示多个HTML页面。使用框架可以非常方便的完成导航工作，让网站的结构更加清晰，而且各个框架之间决不存在干扰问题。利用框架最大的特点就是使网站的风格一致。通常把一个网站中页面相同的部分单独制作成一个页面，作为框架结构的一个子框架的内容给整个网站公用。 。 |
| WEBSOCKET | WebSocket协议是基于TCP的一种新的网络协议。它实现了浏览器与服务器全双工(full-duplex)通信——允许服务器主动发送信息给客户端。 |
| WEBVIEW | webview 标签用于集成 'guest（访客）' 内容（比如外部的网页）在你的应用内。每个 webview 运行在独立的进程中。 作为页面它拥有不一样的权限并且所有的嵌入的内容和你应用之间的交互都将是异步的。这将保证你的应用对于嵌入的内容的安全性。 |

# web 架构设计

## web 架构简述

### Web系统架构图



### WebIM架构设计特点

#### Component 组件化开发

底层功能组件以二进制模块方式提供，框架(angtular)、模块(Module)、组件

(Component)等之间松耦合关联，UI采用组件化设计。

#### Communication通信

组件开发期间，减少代码依赖，增强开发人员的独立编码和测试；即各组件之间通过调用广播监听的形式通信，数据采用SDK—API。包含必要的通信接口信息；

#### User interface交互设计模式

主框架模块设计采用MVVM设计模式。MVVM是Model-View-ViewModel的简写。微软的WPF带来了新的技术体验，如Silverlight、音频、视频、3D、动画……，这导致了软件UI层更加细节化、可定制化。同时，在技术层面，WPF也带来了 诸如Binding、Dependency Property、Routed Events、Command、DataTemplate、ControlTemplate等新特性。MVVM（Model-View-ViewModel）框架的由来便是MVP（Model-View-Presenter）模式与WPF结合的应用方式时发展演变过来的一种新型架构框架。它立足于原有MVP框架并且把WPF的新特性糅合进去，以应对客户日益复杂的需求变化。

#### Agent代理

基于node服务的代理模块，部署到服务端，为web做代理服务。基础服务、扩展服务、增值服务的顺序进行设计与开发，顺序接入底层服务。同时提供安全策略相关的能力实现。如同源，协议等。

#### WebSocket协议

WebSocket是HTML5开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

在WebSocket API中，浏览器和服务器只需要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。

浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求，连接建立以后，客户端和服务器端就可以通过 TCP 连接直接交换数据。

当你获取 Web Socket 连接后，你可以通过 send() 方法来向服务器发送数据，并通过 onmessage 事件来接收服务器返回的数据。

#### Process独立进程

业务线程分离，互不影响

 UI主进程与UI渲染进程分离，渲染进程的异常不会影响应用的崩溃

 业务数据业务线程与消息服务线程分离，在类似登录环节业务请求与消息服务数据量同时增多时，很大程度上提升了用户的交互体验与消息的及时送达。

### WebIM技术特点

#### 概述

基于H5新技术的即时通信产品WebIM着重考虑跨平台兼容性与多端通用性。有效的提高了IM产品的适用范围。让桌面应用更好的引用基于页面的即时通信化产品。

#### 兼容

研发适用HTML5，CSS3，JAVASCRIPT，WebSocket新技术。让应用有更强的适应性。暂时对浏览器版本有一定的要求。需要支持PB数据结构，WebSocket通信协议。

#### 技术关键字

IM即时通信，C/S架构，TCP协议，NODE防火墙代理，Angular交互架构。

#### 数据

用户数据轻量化数据存储机制，NOSQL数据库存储，数据访问更加高效，包括索引，事务处理和健壮的查询功能。

#### 交互视觉

UI组件渲染数据采用双向绑定模式，随着业务数据的更新，UI组件会自动监听数据变更实现渲染更新。

#### 版本控制

业务数据的版本号控制，实现增量更新，保证数据最小集更新。

#### 通知

在线业务数据变更的实时通知，通知内只带有增量变更的必要业务数据，既保证了变更的实时性，又避免了与服务器的大数据量交互。

#### 消息

推拉结合的离线消息机制，既保证了在少量离线消息时的及时推送，又能保证在大量离线消息时，客户端可以有优先级的进行消息拉取及有序的消息展示，消息拉取完成后清理远程消息队列，并且增加的了本地的消息排重机制，既保证了消息送达不丢失，又可以避免本地消息的重复展示。

#### 预加载

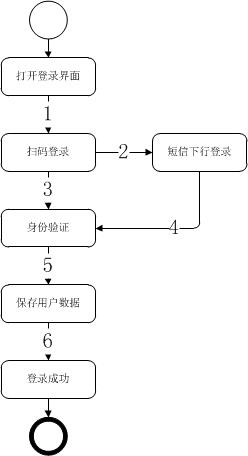
头像滚动批量预加载模式，有效避免了首次登录时大量头像数据下载造成的网络阻塞。

## web业务模块

### 设计说明

1. 模块设计遵循CS机构。一个模块内实现一个或多个组件，一个组件内实现一个或多个子组件或事件。
2. 模块、组件和命令在系统内都有一ID标识。模块ID在系统内唯一，组件ID在管理子系统内唯一，命令ID在组件内唯一。
3. 组件对外提供接口IpcComponentApi。
4. 框架管理的Controller和Service实现为启动时加载，组件实现为启动加载和运行时加载两种模式。管理模块的配置文件使用webpack描述，包含以下信息：模块(组件)名称、dll路径、是否运行时时加载。
5. 管理组件的代码同质性较高，代码量较少，Appmain,package内部采用模板实现默认的管理组件，避免太多的管理component，以减少维护复杂度。在需要特定处理的情况下，可独立实现管理component，该选项在package中配置。

### 登录模块



1. 打开登录界面，默认登录方式为扫码登录

2. 扫码登录和短信下行登录之间可相互切换

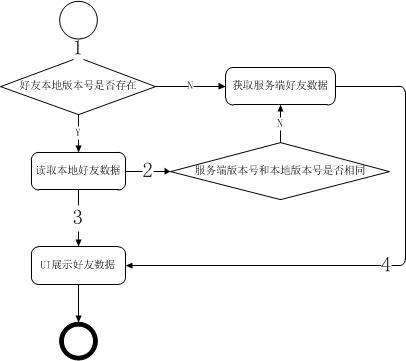
3. 手机扫二维码之后发起身份认证请求

4. 输入手机号和验证码后，点击“登录”发起身份认证请求

5. 验证用户身份的合法性并返回数据

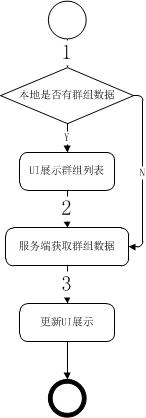
6. 登录成功后进入主界面

### 好友列表模块



1. 获取本地缓存的版本号
2. 获取服务端的好友版本号，和本地版本作对比
3. 本地获取好友数据之后根据拼音首字母排序
4. 服务端获取好友数据之后根据拼音首字母排序。

### 群列表模块

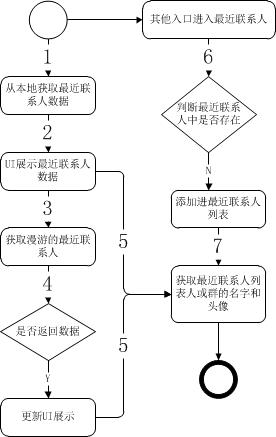


1. 获取本地群组数据

2. 获取本地群组数据版本号，根据版本号获取服务端群组数据

3. 获取的服务端数据和本地数据作对比，更新UI的展示

### 最近联系人模块



1. 从本地获取最近联系人数据

2. UI展示该数据

3. 从本地获取版本号去服务端获取最新的最近联系人数据

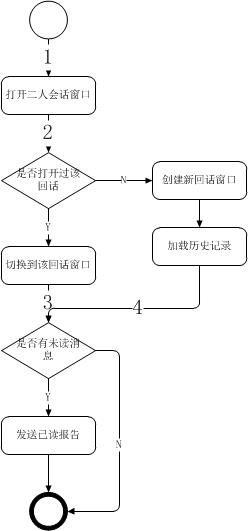
4. 判断是否有返回数据，有则更新最近联系人数据

5. 展示最近联系人列表后获取最近联系人的名字和头像并展示

6. 判断在最近联系人列表中是否存在

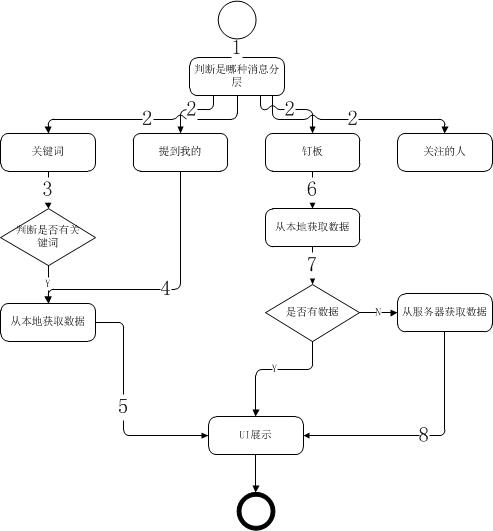
7. 如果不存在则添加进最近联系人列表

### 会话业务模块



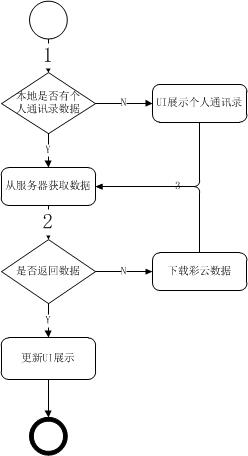
1. 点击最近联系人列表打开会话
2. 打开会话判断是否该回话是否已经打开过了
3. 判断是否有未读消息。

### 消息订阅模块

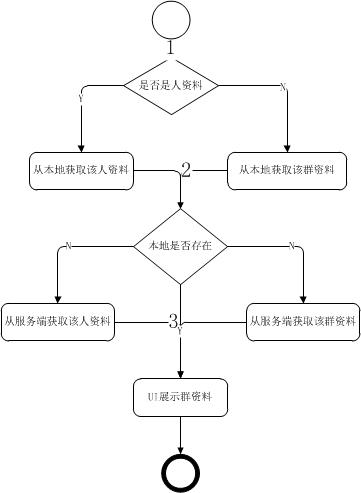


1. 判断属于哪种消息分层
2. 消息分层类型不一样所对应的业务逻辑也不一样
3. 如果是关键词类型，则从本地获取关键词，如果有关键词则返回匹配的聊天记录
4. 如果是提到我的类型，则返回聊天记录中提到我的记录
5. 把获取到的数据展示在UI
6. 如果是钉板类型，则从本地获取数据
7. 如果有钉板数据就展示在UI
8. 如果没有钉板数据则从服务端获取数据展示在UI

### 个人通信录模块



### profile模块

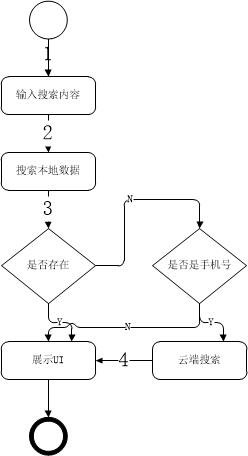


1. 判断是个人资料还是群资料

2. 从本地获取该个人或者群资料，有就展示

3. 从服务端获取该个人或者群资料，展示UI

### 搜索模块



1. 输入名字、拼音或者数字

2. 搜索本地数据，判断是否有本地搜索结果，有则显示UI，没有则判断是否是手机号，如果是手机号则从云端搜索，没有则不搜索

3. 把云端搜索的结果显示在UI

## 技术特性

1. **支持平台**

支持FdcpPlatform，设计可扩展的灵活接入模式，使其能够适应平台模型变化。

1. **组件化、松耦合**

功能组件以二进制模块方式提供，框架(FdeFramework)、模块(FdeModule)、组件(FdeComponent)等之间松耦合关联。

1. **开发独立、代码少依赖**

组件开发期间，减少代码依赖，增强开发人员的独立编码和测试；即各组件只提供一个文件即可，文件中包含纯虚基类和必要的预定义信息。

1. **支持运营、灵活配置、增量更新**

支持各种运营模式，功能组件既可以静态配置，也可以动态配置(包括用户操作和运营的push指令)；系统可增量更新。

1. **可扩展、可接入**

支持扩展组件的下载、加载与卸载，支持第三方组件接入系统。

### V7架构设计目标

针对飞信版本WebIM架构的瓶颈，并参考业界类似规模桌面软件的先进经验，我们确定了V7架构的技术设计

1. **支持平台**

支持FdcpPlatform，设计可扩展的灵活接入模式，使其能够适应平台模型变化。

1. **组件化、松耦合**

功能组件以二进制模块方式提供，框架(FdeFramework)、模块(FdeModule)、组件(FdeComponent)等之间松耦合关联。

1. **开发独立、代码少依赖**

组件开发期间，减少代码依赖，增强开发人员的独立编码和测试；即各组件只提供一个h文件即可，文件中包含纯虚基类和必要的预定义信息。

1. **支持运营、灵活配置、增量更新**

支持各种运营模式，功能组件既可以静态配置，也可以动态配置(包括用户操作和运营的push指令)；系统可增量更新。

1. **可扩展、可接入**

支持扩展组件的下载、加载与卸载，支持第三方组件接入系统。

1. **固化核心、步进发布**

主框架模块完成所有预定义模块的功能及数据流转，框架开发完成后，做到在后续的开发中，基本不再修改主框架模块。按基本功能、高级功能、增值功能的顺序进行设计与开发，顺序接入系统。不能实现的组件以空实现的方式接入系统。

1. **业务驱动界面模型，支持灵活流畅、任意组合的界面需求**

界面使用DirectUI技术，可以很容易的支持各种各样的界面需求。界面方案的变更，只需更改与界面对应的UI适配器即可，无需重构其他子系统。界面与业务完全无关，业务层只需定义本业务需要的界面支持即可，界面的实现完全在UI层控制。

1. **层次分明，切割合理**

纵向按业务切割模块，并考虑开发工作量。

### Angular

1. **简述**

AngularJS是Google开源的一款JavaScript MVC的前端框架，弥补了HTML在构建应用方面的不足，其通过使用指令（directives）结构来扩展HTML词汇，使开发者可以使用HTML来声明动态内容，从而使得Web开发和测试工作变得更加容易。

AngularJS 是一个为动态WEB应用设计的结构框架，提供给大家一种新的开发应用方式，这种方式可以让你扩展HTML的语法，以弥补在构建动态WEB应用时静态文本的不足，从而在web应用程序中使用HTML声明动态内容。Angular可以帮助你组织JavaScript代码，可以创建响应式网站（会对用户的请求产生快速的反应），Angular可以和JQuery很好的协调、方便测试（搭建可维护的应用）。

简单的解释Angular就是一个可以给HTML加上互动性的客户端JS框架。

1. **特点**

AngularJS有五个主要核心特性，如下介绍：

双向数据绑定 —— 实现了把model与view完全绑定在一起，model变化，view也变化，反之亦然。

模板 —— 在AngularJS中，模板相当于HTML文件被浏览器解析到DOM中，AngularJS遍历这些DOM，也就是说AuguarJS把模板当做DOM来操作，去生成一些指令来完成对view的数据绑定。

MVVM —— 吸收了传统的MVC设计模式针但又并不执行传统意义上的MVC，更接近于MVVM(Moodel-View-ViewModel)。

依赖注入 —— AngularJS拥有内建的依赖注入子系统，可以帮助开发人员更容易的开发，理解和测试应用。

指令 —— 可以用来创建自定义的标签，也可以用来装饰元素或者操作DOM属性。

### Webpack

**1 简介**

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

现今的很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的JavaScript代码和一大堆依赖包。为了简化开发的复杂度，前端社区涌现出了很多好的实践方法

A 模块化，让我们可以把复杂的程序细化为小的文件;

B 类似于TypeScript这种在JavaScript基础上拓展的开发语言：使我们能够实现目前版本的JavaScript不能直接使用的特性，并且之后还能转换为JavaScript文件使浏览器可以识别；

C Scss，less等CSS预处理器

这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别,而手动处理又是非常繁琐的，这就为WebPack类的工具的出现提供了需求。

**2 特点**

生成Source Maps (使调试更容易)

开发总是离不开调试，方便的调试能极大的提高开发效率，不过有时候通过打包后的文件，你是不容易找到出错了的地方，对应的你写的代码的位置的，Source Maps就是来帮我们解决这个问题的。

通过简单的配置，webpack就可以在打包时为我们生成的source maps，这为我们提供了一种对应编译文件和源文件的方法，使得编译后的代码可读性更高，也更容易调试。

在webpack的配置文件中配置source maps，需要配置devtool，它有以下四种不同的配置选项，各具优缺点，描述如下：

使用webpack构建本地服务器

想让你的浏览器监听你的代码的修改，并自动刷新显示修改后的结果，其实Webpack提供一个可选的本地开发服务器，这个本地服务器基于node.js构建，可以实现你想要的这些功能，不过它是一个单独的组件，在webpack中进行配置之前需要单独安装它作为项目依赖

npm install --save-dev webpack-dev-serverdevserver

作为webpack配置选项中的一项，以下是它的一些配置选项：

**3 Loaders**

通过使用不同的loader，webpack有能力调用外部的脚本或工具，实现对不同格式的文件的处理，比如说分析转换scss为css，或者把下一代的JS文件（ES6，ES7)转换为现代浏览器兼容的JS文件，对React的开发而言，合适的Loaders可以把React的中用到的JSX文件转换为JS文件。

Loaders需要单独安装并且需要在webpack.config.js中的modules关键字下进行配置，Loaders的配置包括以下几方面：

test：一个用以匹配loaders所处理文件的拓展名的正则表达式（必须）

loader：loader的名称（必须）

include/exclude:手动添加必须处理的文件（文件夹）或屏蔽不需要处理的文件（文件夹）（可选）；

query：为loaders提供额外的设置选项（可选）注 由于webpack3.\*/webpack2.\*已经内置可处理JSON文件，这里我们无需再添加webpack1.\*需要的json-loader。

**4 Babel**

Babel其实是一个编译JavaScript的平台，它的强大之处表现在可以通过编译帮你达到以下目的：

使用下一代的JavaScript代码（ES6，ES7...），即使这些标准目前并未被当前的浏览器完全的支持；  
使用基于JavaScript进行了拓展的语言，比如React的JSX；

**5 模块**

Webpack有一个不可不说的优点，它把所有的文件都当做模块处理，JavaScript代码，CSS和fonts以及图片等等通过合适的loader都可以被处理。

**6 插件**

插件（Plugins）是用来拓展Webpack功能的，它们会在整个构建过程中生效，执行相关的任务。  
 Loaders和Plugins常常被弄混，但是他们其实是完全不同的东西，可以这么来说，loaders是在打包构建过程中用来处理源文件的（JSX，Scss，Less..），一次处理一个，插件并不直接操作单个文件，它直接对整个构建过程其作用。

Webpack有很多内置插件，同时也有很多第三方插件，可以让我们完成更加丰富的功能。

常用插件：

1. HtmlWebpackPlugin：这个插件的作用是依据一个简单的index.html模板，生成一个自动引用你打包后的JS文件的新index.html。这在每次生成的js文件名称不同时非常有用（比如添加了hash值）。

2. Hot Module Replacement：(HMR）也是webpack里很有用的一个插件，它允许你在修改组件代码后，自动刷新实时预览修改后的效果。

**7 优势**

A 代码拆分：  
 Webpack支持同步/异步两种组织模块依赖的方式

B loader：  
 Webpack只支持原生的JavaScript模块 但是loader转换器可以将各种类型的资源转换 成JavaScript模块资源 这样Webpack可以处理各种类型的资源

C 智能解析  
 Webpack有个智能解析器 可以解析任何第三方库 无论他们的模块形式是CommonJS AMD还是原生的JS模块

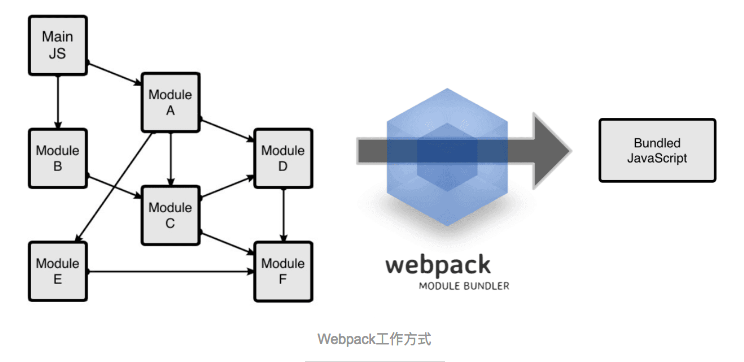
D 插件系统   
 丰富的插件系统 方便开发和使用开源的插件 满足各种需求

E 快速运行  
 Webpack使用异步I/O和多级缓存提高效率 使得其可以快速增量编译

**8 对比**

**9 工作方式**

Webpack的工作方式是：把你的项目当做一个整体，通过一个给定的主文件（如：index.js），Webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包为一个（或多个）浏览器可识别的JavaScript文件。



### IndexedDB

**1 简介**

IndexedDB是在浏览器中保存结构化数据的一种数据库，IndexedDB的思想是创建一套API，方便保存和读取JavaScript对象，同时支持查询和搜索。IndexedDB的最大特色是使用对象保存数据。一个IndexedDB数据库，就是一组位于相同命名空间下的对象的集合。

**2 特性**

（1） 键值对储存（Key-Value）

IndexedDB内部采用对象仓库（object store）存放数据。所有类型的数据都可以直接存入，包括JavaScript对象。在对象仓库中，数据以“键值对”的形式保存，每一个数据都有对应的键名，且键名必须是独一无二的，不能有重复，否则会抛出错误。

（2） 异步API（asynchronous API ）

IndexedDB数据库在执行增、删、改和查的操作时不会锁死浏览器，用户依然可以进行其它操作。相比之下，localStorage的操作都是同步的。异步设计是为了防止大量数据的读写时拖慢网页，而影响用户的网站体验。

（3） 支持事务（transaction）

IndexedDB支持事务（transaction），这意味着一系列操作步骤之中，只要有一步失败，整个事务就都取消，数据库回到事务发生之前的状态，不存在只改写一部分数据的情况。

（4） 同域限制

IndexedDB也受到同域限制，每一个数据库对应创建该数据库的域名。来自不同域名的网页，只能访问自身域名下的数据库，而不能访问其他域名下的数据库。

（5） 存储空间大

IndexedDB的存储空间比localStorage大得多，一般来说不少于250MB。IE的储存上限是250MB，Chrome和Opera是硬盘剩余空间的某个百分比，Firefox则没有上限。

（6） 支持二进制储存

IndexedDB不仅可以储存字符串，还可以储存二进制数据。

### localStorage

**1 简介**

在HTML5中，新加入了一个localStorage特性，这个特性主要是用来作为本地存储来使用的，解决了cookie存储空间不足的问题(cookie中每条cookie的存储空间为4k)，localStorage中一般浏览器支持的是5M大小，这个在不同的浏览器中localStorage会有所不同。

localStorage的使用也是遵循同源策略的，所以不同的网站直接是不能共用相同的localStorage

**2 特性**

localStorage的优势

1、localStorage拓展了cookie的4K限制

2、localStorage可以将第一次请求的数据直接存储到本地，这个相当于一个5M大小的针对于前端页面的数据库，相比于cookie可以节约带宽，但是这个却是只有在高版本的浏览器中才支持的

localStorage的局限

1、浏览器的大小不统一，并且在IE8以上的IE版本才支持localStorage这个属性

2、目前所有的浏览器中都会把localStorage的值类型限定为string类型，这个在对我们日常比较常见的JSON对象类型需要一些转换

3、localStorage在浏览器的隐私模式下面是不可读取的

4、localStorage本质上是对字符串的读取，如果存储内容多的话会消耗内存空间，会导致页面变卡

5、localStorage不能被爬虫抓取到

localStorage与sessionStorage的唯一一点区别就是localStorage属于永久性存储，而sessionStorage属于当会话结束的时候，sessionStorage中的键值对会被清空。

3 应用

### ES6

ECMAScript 6（以下简称ES6）是JavaScript语言的下一代标准。因为当前版本的ES6是在2015年发布的，所以又称ECMAScript 2015。也就是说，ES6就是ES2015。

ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（另外的 ECMAScript 方言还有 Jscript 和 ActionScript）。日常场合，这两个词是可以互换的。

最常用的ES6特性

let, const, class, extends, super, arrow functions, template string, destructuring, default, rest arguments

### CMD

CMD又叫做命令提示符，在操作系统中提示进行命令输入的一种工作提示符。在不同的操作系统环境下，命令提示符各不相同。

web端主要用于启动程序和打包项目。

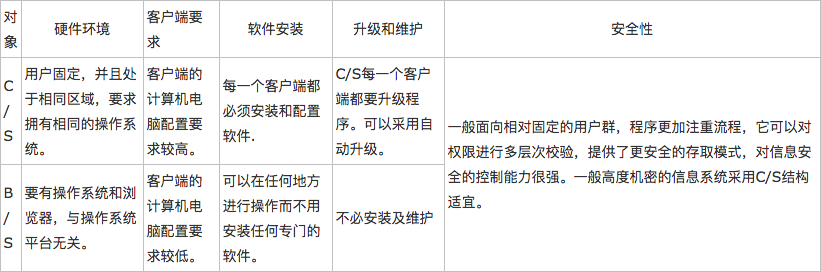
### CS架构

1 简介

CS（Client/Server）：客户端----服务器结构。C/S结构在技术上很成熟，它的主要特点是交互性强、具有安全的存取模式、网络通信量低、响应速度快、利于处理大量数据。因为客户端要负责绝大多数的业务逻辑和UI展示，又称为胖客户端。它充分利用两端硬件，将任务分配到Client 和Server两端，降低了系统的通讯开销。C/S结构的软件需要针对不同的操作系统系统开发不同版本的软件，加之产品的更新换代十分快，已经很难适应百台电脑以上局域网用户同时使用。

　　C/S 架构是一种典型的两层架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是Socket服务器端，服务器端的程序通过[**Socket**](http://blog.csdn.net/fightfaith/article/details/50055581)与客户端的程序通信。

2 C/S 与 B/S 的比较



### WebSocket

1 简介

WebSocket API是下一代客户端-服务器的异步通信方法。该通信取代了单个的TCP套接字，使用ws或wss协议，可用于任意的客户端和服务器程序。WebSocket目前由W3C进行标准化。WebSocket已经受到Firefox 4、Chrome 4、Opera 10.70以及Safari 5等浏览器的支持。

WebSocket API最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；XHR受到域的限制，而WebSocket允许跨域通信。

Ajax技术很聪明的一点是没有设计要使用的方式。WebSocket为指定目标创建，用于双向推送消息。

2 特点

事件驱动

异步

使用ws或者wss协议的客户端socket

能够实现真正意义上的推送功能

3 语法

只专注于客户端的API，因为每个服务器端语言有自己的API。下面的代码片段是打开一个连接，为连接创建事件监听器，断开连接，消息时间，发送消息返回到服务器，关闭连接。

// 创建一个Socket实例

var socket = new WebSocket('ws://localhost:8080');

// 打开Socket

socket.onopen = function(event) {

// 发送一个初始化消息

socket.send('I am the client and I'm listening!');

// 监听消息

socket.onmessage = function(event) {

console.log('Client received a message',event);

};

// 监听Socket的关闭

socket.onclose = function(event) {

console.log('Client notified socket has closed',event);

};

// 关闭Socket....

socket.close()

};

让我们来看看上面的初始化片段。参数为URL，ws表示WebSocket协议。onopen、onclose和onmessage方法把事件连接到Socket实例上。每个方法都提供了一个事件，以表示Socket的状态。

### 轮询

轮询：客户端定时向服务器发送Ajax请求，服务器接到请求后马上返回响应信息并关闭连接。

优点：后端程序编写比较容易。

缺点：请求中有大半是无用，浪费带宽和服务器资源。

实例：适于小型应用。

长轮询：客户端向服务器发送Ajax请求，服务器接到请求后hold住连接，直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

优点：在无消息的情况下不会频繁的请求，耗费资小。

缺点：服务器hold连接会消耗资源，返回数据顺序无保证，难于管理维护。 Comet异步的ashx，

### 长连接

长连接：在页面里嵌入一个隐蔵iframe，将这个隐蔵iframe的src属性设为对一个长连接的请求或是采用xhr请求，服务器端就能源源不断地往客户端输入数据。

优点：消息即时到达，不发无用请求；管理起来也相对方便。

缺点：服务器维护一个长连接会增加开销。

### 浏览器引擎

引擎-----HTML渲染引擎 JS交互引擎

是套成熟的快速构建UI界面的工具。其主要功能与特性如下：

UI界面编辑拖拽式操作，所见即所得；

采用平台＋插件的体系构架，平台即DirectUi界面编辑工具，界面中的各个控件采用插件的方式导入到平台中；

架构优秀且久经考验，有利于UI开发人员更加专注于交互特效的开发，构建绚烂多彩的客户端交互界面；

可扩展图形引擎，基本图形引擎的实现采用GDI和GDI+的方式，可自由扩充使用DirectX、OpenGL等方式；

自动控件布局处理；

支持多语种功能；

内存最小化资源管理；

支持多种方式的控件消息处理；

DirectUI界面工具共约20万行源代码，由于其成熟性与通用性，并且在业界已得到广泛应用与验证。

### Map的内存分配机制分析

功能说明：

map的内存分配机制分析。

代码说明：

map所管理的内存地址可以是不连续的。如果key是可以通过<排序的，那么，map最后的结果是有序的。它是通过一个平衡二叉树来保存数据。所以，其查找效率极高。

实现方式：

分析代码如下

#include <iostream>

#include <string>

#include <map>

#include <windows.h>

using namespace std;

class CData

{

public:

CData()

{

sequence = 0;

this->remark = "default string";

cout << "CData()\t" << toString() << "\t" << this << endl;

}

CData(int i, string &s)

{

this->sequence = i;

this->remark = s;

cout << "CData(int i,string &s)\t" << toString() << "\t" << this << endl;

}

CData(const CData &data)

{

this->sequence = data.sequence;

this->remark = data.remark;

cout << "CData(const CData &data)\t" << toString() << "\t" << this << endl;

}

CData operator = (const CData &data)

{

this->sequence = data.sequence;

this->remark = data.remark;

cout << "CData operator = (const CData &data)\t" << toString() << "\t" << this << endl;

return \*this;

}

void setSequence(const int i)

{

this->sequence = i;

}

void setRemark(const string &s)

{

this->remark = s;

}

string toString() const

{

char tmp[2048] = { 0 };

sprintf(tmp, "[sequence:%d | remark:%s]", this->sequence, this->remark.c\_str());

//此处应该有内存复制操作，所以不用担心返回后tmp数组所指向的内存被修改或者不存在的情况。

return tmp;

}

~CData()

{

cout << "~CData()\t" << this << endl;

}

protected:

private:

int sequence;

string remark;

};

int main(int argc, char \*\*argv)

{

cout << "process begin at " << (void\*)&main << endl;

string str = "baby\_test";

CData data1(1, str);

CData data2(2, str);

CData data3(3, str);

CData data4(4, str);

CData data5(5, str);

CData data6(6, str);

CData data7(7, str);

CData data8(8, str);

CData data9(9, str);

CData data10(10, str);

cout << "push CData to map" << endl;

map<int,CData> map\_data;

cout << "max size of list<CData> is " << map\_data.max\_size() << endl;

cout << "size of list<CData> is " << map\_data.size() << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data1)" << endl;

map\_data.insert(make\_pair(1, data1));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data2)" << endl;

map\_data.insert(make\_pair(2, data2));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data3)" << endl;

map\_data.insert(make\_pair(3, data3));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data4)" << endl;

map\_data.insert(make\_pair(4, data4));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data5)" << endl;

map\_data.insert(make\_pair(5, data5));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data6)" << endl;

map\_data.insert(make\_pair(6, data6));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data7)" << endl;

map\_data.insert(make\_pair(7, data7));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data8)" << endl;

map\_data.insert(make\_pair(8, data8));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data9)" << endl;

map\_data.insert(make\_pair(9, data9));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data10)" << endl;

map\_data.insert(make\_pair(10, data10));

Sleep(2 \* map\_data.size());

// 程序到此为止，可以看到在每次push的时候，对象的复制构造函数会调用2次。

cout << "===============show map by iterator" << endl;

for (map<int,CData>::iterator itr = map\_data.begin(); itr != map\_data.end(); itr++)

{

// 显示的地址信息，是每次push的时候所复制的对象的地址。

cout<<"key:"<< itr->first<<"\t" << "value:address of itr is " << &(itr->second) << " and msg is " << itr->second.toString() << endl;

}

cout << "===============clear map 1" << endl;

// map中的元素不是指针，此处的清理，会调用析构函数。

map\_data.clear();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data6)" << endl;

map\_data.insert(make\_pair(6, data6));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data2)" << endl;

map\_data.insert(make\_pair(2, data2));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data3)" << endl;

map\_data.insert(make\_pair(3, data3));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data1" << endl;

map\_data.insert(make\_pair(1, data1));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data10)" << endl;

map\_data.insert(make\_pair(10, data10));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data4)" << endl;

map\_data.insert(make\_pair(4, data4));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data7)" << endl;

map\_data.insert(make\_pair(7, data7));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data8)" << endl;

map\_data.insert(make\_pair(8, data8));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data9)" << endl;

map\_data.insert(make\_pair(9, data9));

Sleep(2 \* map\_data.size());

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*map\_data.push\_back(data5)" << endl;

map\_data.insert(make\_pair(5, data5));

Sleep(2 \* map\_data.size());

// 程序到此为止，可以看到在每次push的时候，对象的复制构造函数会调用2次。

cout << "===============show map by iterator 2" << endl;

for (map<int, CData>::iterator itr = map\_data.begin(); itr != map\_data.end(); itr++)

{

// 显示的地址信息，是每次push的时候所复制的对象的地址。

// 可以看到，不管插入的数据是如何的乱序，map的结果是排好序的。

cout << "key:" << itr->first << "\t" << "value:address of itr is " << &(itr->second) << " and msg is " << itr->second.toString() << endl;

}

cout << "===============clear map 2" << endl;

// map中的元素不是指针，此处的清理，会调用析构函数。

map\_data.clear();

cout << "======================end of process" << endl;

return 0;