HTML5面试题

基础强化

1. 聊一聊前端存储。

（1）cookie

（2）短暂的sessionStorage

（3）简易强大的localStorage

（4）websql与indexeddb

详细参见： <https://segmentfault.com/a/1190000005927232>

1. BFC

(1) w3c规范中的BFC定义：

浮动元素和绝对定位元素，非块级盒子的块级容器（例如 inline-blocks, table-cells, 和 table-captions），以及overflow值不为“visiable”的块级盒子，都会为他们的内容创建新的BFC（块级格式上下文）。

在BFC中，盒子从顶端开始垂直地一个接一个地排列，两个盒子之间的垂直的间隙是由他们的margin 值所决定的。在一个BFC中，两个相邻的块级盒子的垂直外边距会产生折叠。

在BFC中，每一个盒子的左外边缘（margin-left）会触碰到容器的左边缘(border-left)（对于从右到左的格式来说，则触碰到右边缘）。

(2) BFC的通俗理解：

首先BFC是一个名词，是一个独立的布局环境，我们可以理解为一个箱子（实际上是看不见摸不着的），箱子里面物品的摆放是不受外界的影响的。转换为BFC的理解则是：BFC中的元素的布局是不受外界的影响（我们往往利用这个特性来消除浮动元素对其非浮动的兄弟元素和其子元素带来的影响。）并且在一个BFC中，块盒与行盒（行盒由一行中所有的内联元素所组成）都会垂直的沿着其父元素的边框排列。

详细参见：

<http://www.w3cplus.com/css/understanding-bfc-and-margin-collapse.html>

<https://www.zhihu.com/question/28433480>

前端工程化

1. 场景：你是第一天来公司上班的，项目代码托管在GitLab，项目地址：[git@lab.com:org/project.git](mailto:git@lab.com:org/project.git),现在有一处代码需要你修改。请下完成此项任务中，与git/gitlab相关的操作步骤。

第一步：$> ssh-keygen -t rss -C [zhangsan@abc.com](mailto:zhangsan@abc.com)

第二步：拷贝公钥到gitlab

第三步：

$> git config —global user.name zhangsan

$> git config —global user.email [zhangsan@abc.com](mailto:zhangsan@abc.com)

第四步：$> git clone [git@lab.com:org/project.git](mailto:git@lab.com:org/project.git)

第五步：$> git checkout -b project-20170227-zhangsan-bugfix

第六步：修改代码

第七步：git status

第八步：git add .

第九不：git commit -am ‘bugfix’

第八步：git push --set-upstream origin project-20170227-zhangsan-bugfix

1. CSS，JS代码压缩，以及代码CDN托管，图片整合。

（1）CSS,JS 代码压缩：

可以应用gulp的gulp-uglify，gulp-minify-css模块完成；可以应用webpack的 UglifyJsPlugin压缩插件完成。

（2）CDN：

内容分发网络(CDN)是一个经策略性部署的整体系统，包括分布式存储、负载均衡、网络请求的重定向和内容管理4个要件。主要特点有：本地Cache加速，镜像服务，远程加速，带宽优化。关键技术有：内容发布，内容路由，内容交换，性能管理。CDN网站加速适合以咨询为主的网站。CDN是对域名加速不是对网站服务器加速。CDN和镜像站比较不需要访客手动选择要访问的镜像站。CDN使用后网站无需任何修改即可使用CDN获得加速效果。如果通过CDN后看到的网页还是旧网页，可以通过URL推送服务解决，新增的网页和图片不需要URL推送。使用动态网页可以不缓存即时性要求很高的网页和图片。CDN可以通过git或SVN来管理。

（3）图片整合

减少网站加载时间的最有效的方式之一就是减少网站的HTTP请求数。实现这一目标的一个有效的方法就是通过CSS Sprites——将多个图片整合到一个图片中，然后再用CSS来定位。缺点是可维护性差。可以使用百度的fis/webpack来自动化管理sprite。

1. 如何利用webpack把代码上传服务器以及转码测试？

（1）代码上传：

可以使用sftp-webpack-plugin，但是会把子文件夹给提取出来，不优雅。可以使用gulp+webpack来实现。

（2）转码测试

webpack应用babel来对ES6转码，开启devtool: “source-map" 来进行浏览器测试。应用karma或mocha来做单元测试。

1. 项目上线流程是怎样的？

（1）流程建议

- 模拟线上的开发环境

本地反向代理线上真实环境开发即可。（apache，nginx，nodejs均可实现）

- 模拟线上的测试环境

模拟线上的测试环境，其实是需要一台有真实数据的测试机，建议没条件搭daily的，就直接用线上数据测好了，只不过程序部分走你们的测试环境而已，有条件搭daily最好。

- 可连调的测试环境

可连调的测试环境，分为2种。一种是开发测试都在一个局域网段，直接绑hosts即可，不在一个网段，就每人分配一台虚拟的测试机，放在大家都可以访问到的公司内网，代码直接往上布即可。

- 自动化的上线系统

自动化的上线系统，可以采用Jenkins。如果没有，可以自行搭建一个简易的上线系统，原理是每次上线时都抽取最新的trunk或master，做一个tag，再打一个时间戳的标记，然后分发到cdn就行了。界面里就2个功能，打tag，回滚到某tag，部署。

- 适合前后端的开发流程

开发流程依据公司所用到的工具，构建，框架。原则就是分散独立开发，互相不干扰，连调时有hosts可绑即可。

（2）简单的可操作流程

- 代码通过git管理，新需求创建新分支，分支开发，主干发布

- 上线走简易上线系统，参见上一节

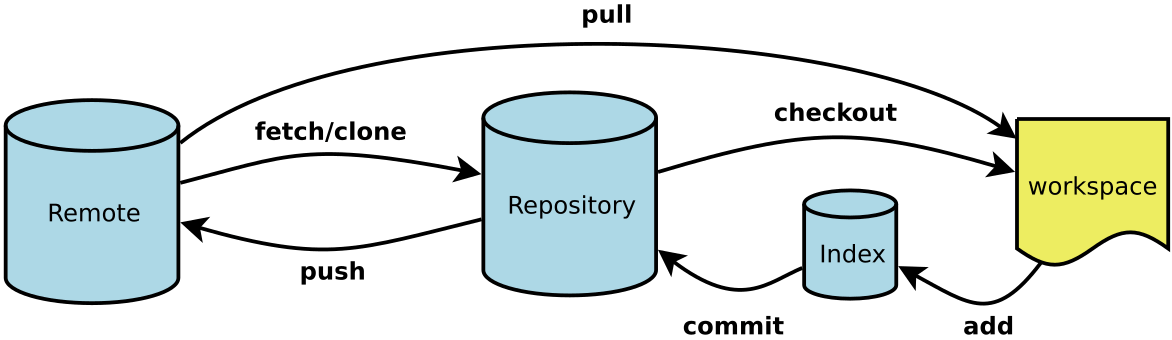
- 通过gulp+webpack连到发布系统，一键集成，本地只关心原码开发

- 本地环境通过webpack反向代理的server

- 搭建基于linux的本地测试机，自动完成build+push功能

1. 工程化怎么管理的?

gulp和webpack

1. git常用命令

Workspace：工作区

Index / Stage：暂存区

Repository：仓库区（或本地仓库）

Remote：远程仓库

详细参见： http://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html

1. webpack 和 gulp对比

Gulp 就是为了规范前端开发流程，实现前后端分离、模块化开发、版本控制、文件合并与压缩、mock数据等功能的一个前端自动化构建工具。说的形象点，“Gulp就像是一个产品的流水线，整个产品从无到有，都要受流水线的控制，在流水线上我们可以对产品进行管理。” 另外，Gulp是通过task对整个开发过程进行构建。

Webpack 是当下最热门的前端资源模块化管理和打包工具。它可以将许多松散的模块按照依赖和规则打包成符合生产环境部署的前端资源。还可以将按需加载的模块进行代码分隔，等到实际需要的时候再异步加载。通过 loader的转换，任何形式的资源都可以视作模块，比如 CommonJs 模块、AMD 模块、ES6 模块、CSS、图片、JSON、Coffeescript、LESS 等。

Gulp和Webpack功能实现对比：从基本概念、启动本地Server、sass/less预编译、模块化开发、文件合并与压缩、mock数据、版本控制、组件控制八个方面对Gulp和Webpack进行对比。

详细参见： <http://www.tuicool.com/articles/e632EbA>

1. webpack打包文件太大怎么办?

webpack 把我们所有的文件都打包成一个 JS 文件，这样即使你是小项目，打包后的文件也会非常大。可以从去除不必要的插件，提取第三方库，代码压缩，代码分割，设置缓存几个方面着手优化。

详细参见： http://www.jianshu.com/p/a64735eb0e2b

1. 不想让别人盗用你的图片，访问你的服务器资源该怎么处理？

目前常用的防盗链方法主要有两种：

（1）设置Referer：适合不想写代码的用户，也适合喜欢开发的用户

（2）签名URL：适合喜欢开发的用户

详细参见： https://yq.aliyun.com/articles/57931

1. 精灵图和base64如何选择？

css精灵，用于一些小的图标不是特别多，一个的体积也稍大，比如大于10K（这个没有严格的界定）。

base64，用于小图标体积较小（相对于css精灵），多少都无所谓。字体图标，用于一些别人做好的图标库（也有少数自己去做的）用起来比较方便，他的图标只能用于单色，图标用只能于一种颜色。

1. webpack怎么引入第三方的库？

拿jQuery为例：

entry: {

page: 'path/to/page.js',

jquery: 'node\_modules/jquery/dist/jquery.min.js'

}

new HtmlWebpackPlugin({

filename: 'index.html',

template: 'index.html',

inject: true,

chunks: ['jquery', 'page'] // 按照先后顺序插入script标签

})

1. 如果线上出现bug git怎么操作？

参见第1题。

1. 用过Nginx吗？都用过哪些？

nginx是一个高性能的HTTP和反向代理服务器。

常使用场景：

（1）反向代理

（2）网站负载均衡

详细参见： http://www.cnblogs.com/hobinly/p/6023883.html

移动端布局与适配

1. iscroll安卓低版本卡顿，如何解决？

方案一：iScroll v5.1.3 设置momentum: true

方案二：配置probeType

方案三：开启硬价加速：给scroll元素增加css样式：-webkit-transform:translate3d(0,0,0);

方案四：判断手机版系统版本，应用原生CSS：overflow-y:scroll

1. 移动布局自适应不同屏幕的几种方式

（1）响应式布局  
（2）100%布局（弹性布局）

（3）等比缩放布局（rem）

1. 请说下移动端常见的适配不同屏幕大小的方法?

参照第15题

1. 移动端屏幕适配?

参照第15题

1. 你们做移动端平时在什么浏览器上测试？

Chrome，Safari，微信X5，UC，其他手机自带浏览器

1. 说说移动端是如何调试的？

移动端调试：

（1）模拟手机调试chrome://inspect

（2）真机调试之android手机+Chrome

（3）真机调试之iphone + safari

（4）UC浏览器

（5）微信内置浏览器调试

（6）debuggap

（7）抓包

详细参考： <https://segmentfault.com/a/1190000005964730>

1. 说说ICONFONT是如何用的？

从以下几个方面做答：

（1）font-face

（2）什么是iconfont,iconfont怎么用

（3）iconfont怎么做

（4）iconfont的利和弊

详细参见： <https://segmentfault.com/a/1190000005904616?_ea=953657>

1. 说说移动端Web分辨率

从以下几个方面做答：

（1）PC到移动，渲染的变迁

（2）可以更改的布局宽度

（3）再次变迁的像素

（4）又一次变迁

（5）是时候说说安卓了

详细参见： https://segmentfault.com/a/1190000005884985

性能和效率

1. 你平时如何评测你写的前端代码的性能和效率。

Chrome DevTools的Timeline: 是用来排查应用性能瓶颈的最佳工具。

Chrome DevTools的Audits: 对页面性能进行检测，根据测试的结果进行优化。

第三方工具Yslow。

详细参见：

<http://www.cnblogs.com/-simon/p/5883336.html>

<http://blog.csdn.net/ivan0609/article/details/45508365>

http://www.wtoutiao.com/p/1305TZW.html

1. 如何优化页面，加快页面的加载速度(至少5条)

（1）优化图片资源的格式和大小

（2）开启网络压缩

（3）使用浏览器缓存

（4）减少重定向请求

（5）使用CDN存储静态资源

（6）减少DNS查询次数

（7）压缩css和js内容

详细参见： http://www.mahaixiang.cn/wyzz/1589.html

1. 怎么保证多人开发进行内存泄漏的检查（内存分析工具）

使用xcode里面的Analyze进行静态分析，为避免不必要的麻烦，多人开发的时候尽量使用ARC。

详细参见： <http://www.jianshu.com/p/424c15b9b73a>

1. 前后端性能如何调优？

参照第20题

1. 浏览器http请求过多怎么解决?

（1）合并JS、CSS文件

（2）合并图片css sprite

（3）使用Image maps

（4）data嵌入图片：如base64

（5）使用CDN，减少http请求头

Web安全

1. 你所了解到的Web攻击技术

（1）XSS攻击

（2）CSRF攻击

（3）网络劫持攻击

（4）控制台注入代码

（5）钓鱼

详细参见： http://blog.csdn.net/fengyinchao/article/details/52303118

1. 如何防止XSS攻击?

（1）将前端输出数据都进行转义

（2）将输出的字符串中的\反斜杠进行转义

（3）从url中获取的信息，防止方法是由后端获取，在前端转义后再行输出

（4）使用cookie的HttpOnly属性，保护好cookie

详细参见： http://blog.csdn.net/fengyinchao/article/details/52303118

1. 项目中有没有用过加密，哪种加密算法？

项目中没有用过，但我了解几个加密算法：

（1）RSA加密

（2）MD5加密

（3）SHA256加密

1. 聊一聊网页的分段传输与渲染

从下面几个方面说：

（1）CHUNKED编码

（2）BIGPIPE

（3）分段传输与bigpipe适用场景

详细参见： <https://segmentfault.com/a/1190000005989601?_ea=984496>

1. 百度移动端首页秒开是如何做到的？

从几个方面优化：

（1）静态文件放置

（2）缓存

（3）外链

（4）缓存DOM

（5）使用iconfont

（6）卡片的异步加载与缓存

（7）不在首屏的就要异步化

（8）少量静态文件的域名

详细参见： https://segmentfault.com/a/1190000005882953

1. 前端速度统计（性能统计）如何做？

回答下面的两个问题：

（1）网站都有哪些指标？

（2）如何统计自己网站的这些指标？

详细参见： https://segmentfault.com/a/1190000005869953

架构

1. 如果让你来制作一个访问量很高的大型网站，你会如何来管理所有css、js文件、图片？

（1）遵循自定的一套CSS，JS和图片文件和文件夹命名规范

（2）依托采用的前端工程化工具，依照工具脚手架规范 (gulp, webpack, grunt, yeoman)

（3）依据采用的框架规范（Vue, React, jQuery）

1. 如果没有框架、怎么搭建你的项目

应用原生JS自己尝试搭建一个MVC架构：

(1) 基本模块

common：公共的一组件，下面的各模块都会用到

config：配置模块，解决框架的配置问题

startup：启动模块，解决框架和Servlet如何进行整合的问题

plugin：插件模块，插件机制的实现，提供IPlugin的抽象实现

routing：路由模块，解决请求路径的解析问题，提供了IRoute的抽象实现和基本实现

controller：控制器模块，解决的是如何产生控制器

model：视图模型模块，解决的是如何绑定方法的参数

action：action模块，解决的是如何调用方法以及方法返回的结果，提供了IActionResult的抽象实现和基本实现

view：视图模块，解决的是各种视图引擎和框架的适配

filter：过滤器模块，解决是执行Action，返回IActionResult前后的AOP功能，提供了IFilter的抽象实现以及基本实现

（2）扩展模块

filters：一些IFilter的实现

results：一些IActionResult的实现

routes：一些IRoute的实现

plugins：一些IPlugin的实现

详细参见： http://www.cnblogs.com/lovecindywang/p/4444915.html

1. 在选择框架的时候要从哪方面入手

影响团队技术选型有很多因素，如技术组成，新技术，新框架，语言及发布等。为了更好的考量不同的因素，需要列出重要的象限，如开发效率、团队喜好，依次来决定哪个框架更适合当前的团队和项目。上线时间影响框架选择，不要盲目替换现有框架。

（1）jQuery

项目功能比较简单。并不需要做成一个单页面应用，就不需要 MV\* 框架。项目是一个遗留系统。与其使用其他框架来替换，不如留着以后重写项目。

（2）AngularJS

当我们在制作一个应用，它对性能要求不是很高的时候，那么我们应该选择开发速度更快的技术栈AngularJS，她拥有混合开发能力的ionic框架。对于复杂的前端应用来说，基于 Angular.js 应用的运行效率，仍然有大量地改进空间。Angular2需要学习新的语言，需慎重选择。

（3）React

选择React有两个原因，一是通过Virtual DOM提高运行效率，二是通过组件化提高开发效率。大型项目首选。选择 React 还有一个原因是：React Native、React VR 等等，可以让 React 运行在不同的平台之上。我们还能通过 React 轻松编写出原生应用，还有 VR 应用。

令人遗憾的是 React 只是一个 View 层，它是为了优化 DOM 的操作而诞生的。为了完成一个完整的应用，我们还需要路由库、执行单向流库、web API 调用库、测试库、依赖管理库等等，为了完整搭建出一个完整的 React 项目，我们还需要做大量的额外工作。

（4）Vue.js

对于使用 Vue.js 的开发者来说，我们仍然可以使用 熟悉的 HTML 和 CSS 来编写代码。并且，Vue.js 也使用了 Virtual DOM、Reactive 及组件化的思想，可以让我们集中精力于编写应用，而不是应用的性能。

对于没有 Angular 和 React 经验的团队，并且规模不大的前端项目来说，Vue.js 是一个非常好的选择。

详细参见： https://zhuanlan.zhihu.com/p/25194137

1. 聊一聊前端模板与渲染

（1）页面级的渲染，后端模板

如smarty，这种方式的特点是展示数据快，直接后端拼装好数据与模板，展现到用户面前，对SEO友好。

（2）异步的请求与新增模板，前端模板

如Mustache，ArtTemplate，前端解析模板的引擎的语法，与后端解析模板引擎语法一致。这样就达到了一份HTML前后端一起使用的效果。

详细参见： https://segmentfault.com/a/1190000005916423

混合开发

1. UIWebView和JavaScript之间是怎么交互的?

UIWebView是[iOS](http://lib.csdn.net/base/ios) SDK中渲染网面的控件，在显示网页的时候，我们可以hack网页然后显示想显示的内容。其中就要用到[JavaScript](http://lib.csdn.net/base/javascript)的知识，而UIWebView与javascript交互的方法就是stringByEvaluatingJavaScriptFromString:

有了这个方法我们可以通过objc调用javascript,可以注入javascript。

UIWebView是[iOS](http://lib.csdn.net/base/ios) SDK中渲染网面的控件，在显示网页的时候，我们可以hack网页然后显示想显示的内容。其中就要用到[JavaScript](http://lib.csdn.net/base/javascript)的知识，而UIWebView与javascript交互的方法就是stringByEvaluatingJavaScriptFromString，有了这个方法我们可以通过objc调用javascript,可以注入javascript

Js调用OC方法原理就是利用UIWebView重定向请求，传一些命令到我们的UIWebView,在UIWebView的delegate的方法中接收这些命令，并根据命令执行相应的objc方法。这样就相当于在javascript中调用objc的方法。

在android中，我们有固有组件webview，经过设置可以让它支持我们的js的渲染，然后在代码中设置（WebViewClient/WebChromeClient）让应用跳转页面时在本webview中跳转，通过webview.loadurl（String str）方法可以在需要的地方加载我们前端的页面或者调用前端所定义的方法（wv.loadUrl("javascript:sendDataToAndroid('我是来自js的呦，你看到了吗')");），我们再通过JavascriptInterface接口设置我们前端和android通讯的标识，

wv.addJavascriptInterface(new MJavascriptInterface(getApplicationContext()), "WebViewFunc");

这样前端就可以在页面上调用我们的方法了，fun1方法是在android中定义的

Window.WebViewFunc.fun1();

总之，前端和android或者ios进行结合开发，我们称之为混合开发，原理就是在原生的开发语言中，我们提供了一个组件webview，这个组件就是我们的原生语言的浏览器，但是我们得自行设置让其能够完美支持我们的应用，需要设置对应的标识，然后连接起来，我们称之为JavascriptInterfac。

1. 混合开发桥接api是怎么调用的，需要引入类库嘛?调用的对象是什么?

Hybrid框架结构

HyBrid App = H5 App + Native框架

H5App 用来实现功能逻辑和页面渲染

Native框架 提供WebView和设备接口供H5调用

方案一 重混合应用， 在开发原生应用的基础上，嵌入WebView但是整体的架构使用原生应用提供，一般这样的开发由Native开发人员和Web前端开发人员组成。Native开发人员会写好基本的架构以及API让Web开发人员开发界面以及大部分的渲染。保证到交互设计，以及开发都有一个比较折中的效果出来，优化得好也会有很棒的效果。   
Hybrid App技术发展的早期， Web的运行性能成为主要瓶颈！  
为解决性能问题Hybrid App走向“ 重混”。

通过多WebView：实现流畅的多页加载和专场动画。

使用Navtive UI 组件：框架、菜单、日期等。

“重混”的优缺点

优点：

– 提升了运行性能  
 – 增强了交互体验

缺点：

– Web和Native技术交叉混杂  
 – 需要同时掌握Web和Native技术， 学习难度增加  
 – 一个页面有Web组件也有Native组件， 编程调试困难

需要引入各自需要的各种依赖工具

方案二：轻混合应用， 使用PhoneGap、AppCan之类的中间件，以WebView作为用户界面层，以Javascript作为基本逻辑，以及和中间件通讯，再由中间件访问底层API的方式，进行应用开发。这种架构一般会非常依赖WebView层的性能。  
随着时代的发展， 手机硬件、浏览器技术、无线网络技术都得到了大幅的提升，H5已经可以支持复杂应用， 并拥有良好的运行性能。使用轻混方案的App也越来越多。

目前我们要学习的Hybrid App开发就是方案二， 使用H5+Js+Native框架开发当前轻混合应用。

Phonegap引入phonegap.js或者cordova.js，对象为navigator

Dcloud引入引入mui.js或者其他的js组件，对象为plus

apiloud引入各种第三方插件，对象为api

顺变提一下， 2012年8月， 微信公众平台的上线，重新定义了移动应用： 移动应用 = Iphone App + Android App + 微信App

1. 说一下你对支付，推送（远程，本地）的理解

消息的推送主要有两种：

一种是本地推送，主要应用在系统的工具中，例如：闹钟，生日提醒等；实现本地推送需要以下三个步骤，

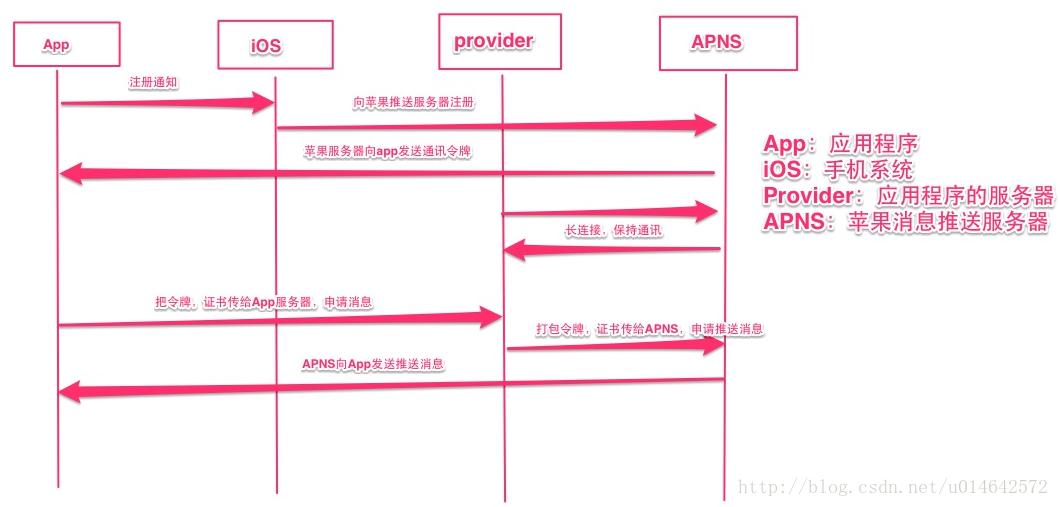
1. 实例化一个本地推送对象
2. 设置通知对象的各个属性
3. 添加本地推送对象

一种是远程消息推送，主要应用联网设备的信息推送，例如：邮件，各种软件的广告或优惠信息的推送。远程推送比较复杂，需要使用开发者账号进行申请证书，获得实现推送功能的配置文件，所以想要实现远程推送功能，必须要有开发者账号并且生成配置文件

1. 完成证书的申请和Xcode的配置
2. 在Demo中注册远程服务对象，并设置其代理
3. 找一个简单的App服务器进行消息推送（推荐使用:PushMeBaby，gitup网站上就有）
4. 运行PushMeBaby

参考网址：http://blog.csdn.net/u014642572/article/details/26857717

远程推送流程图如下



1. 什么是代理和通知，写一下他们基本的实现方式

通知： 一对一  一对多  传值

四个步骤:   
   1.发送通知  
 2.创建监听者  
 3.接收通知  
 4.移除监听者

使用场景:

1- 很多控制器都需要知道一个事件，应该用通知；  
 2 - 相隔多层的两个控制器之间跳转

注意事项:  
  1, 一旦接收消息的对象多了，就难以控制了，可能有你不希望的对象接受了消息并做了处理  
  2,创建了观察者，在dealloc里面一定要移除；

代理：“一对一”，对同一个协议，一个对象只能设置一个代理delegate  
 六个步骤:  
 1.声明一个协议,定义代理方法  
 2. 遵循协议  
 3.设置一个代理对象  
 4.调用代理方法  
 5.给代理赋值  
 6.实现代理方法  
  
 注意事项:  
 1,单例对象不能用代理；  
 2,代理执行协议方法时要使用 respondsToSelector检查其代理是否符合 协议(检查对象能否响应指定的消息),以避免代理在回调时因为没有实现方法而造成程 序崩溃  
  使用场景:

公共接口，方法较多也选择用delegate进行解耦   
 iOS最常用tableViewDelegate，textViewDelegate   
  iOS有很多例子比如常用的网络库AFNetwork，ASIHTTP库，UIAlertView类。

Block:Block是iOS4.0+ 和Mac OS X 10.6+ 引进的对C语言的扩展，用来实现匿名函数的特性。Blocks语法块代码以闭包得形式将各种内容进行传递，可以是代码，可以是数组无所不能。闭包就是能够读取其它函数内部变量的函数。就是在一段请求连续代码中可以看到调用参数（如发送请求）和响应结果。所以采用Block技术能够抽象出很多共用函数，提高了代码的可读性，可维护性，封装性。

使用场景：  
 一：动画  
 二：数据请求回调  
 三：枚举回调  
 四：多线程gcd

注意事项:

block需要注意防止循环引用

参考网址：http://www.cnblogs.com/wenboliu/articles/5422033.html

1. UIViewController的生命周期
2. 通过alloc init 分配内存,初始化controller.
3. loadView（loadView方法默认实现[super loadView]  
   如果在初始化controller时指定了xib文件名,就会根据传入的xib文件名加载对应的xib文件,如果没传xib文件名,默认会加载跟controller同名的xib文件,如果没找到相关联的xib文件,就会创建一个空白的UIView,然后赋給controller的view）
4. viewDidLoad（当loadView创建完view之后，此时view已经完成加载了，会调用viewDidLoad方法；一般我会在这里做界面上的初始化操作，比如添加按钮,子视图,等等.）
5. viewWillAppear（当view在load完之后,将要显示在屏幕之前会调用这个方法，在重写这些方法时候最好先调用一下系统的方法之后在做操作。）
6. viewDidAppear（当view已经在屏幕上显示出来之后,会调用这个方法，当一个视图被移除屏幕并且销毁的时候）
7. viewWillDisappear（当视图将要从屏幕上移除时候调用）
8. viewDidDisappear（当视图已经从屏幕上移除时候调用）
9. Dealloc（view被销毁时候调用，如果是手动管理内存的话，需要释放掉之前在init和viewDidLoad中分配的内存(类似alloc,new,copy)；dealloc方法不能由我们主动调用,必须等引用计数为0时候由系统调用.）

参考网址：<http://www.cnblogs.com/wujy/p/5822329.html>

1. rem布局字体太大怎么处理?

一般情况下我们设置了html根节点的字体大小作为rem单位的一个基本标准，那么我们可以紧接着在body标签内设置一个字体大小为该应用的基本字体大小

针对于一些机型如果一开始就显示的字体不正常，我们可以通过判断机型然后加载不同的样式

<script language="javascript">

window.onload = function() {

alert("1");

var u = navigator.userAgent;

if(u.indexOf('Android') > -1 || u.indexOf('Linux') > -1) { //安卓手机

alert("安卓手机");

} else if(u.indexOf('iPhone') > -1) { //苹果手机

alert("苹果手机");

} else if(u.indexOf('Windows Phone') > -1) { //winphone手机

alert("winphone手机");

}

}

</script>

1. 如何调用原生的接口？

首先你得选择一个合适的框架作为自己的基础，以Dcloud为例，页面中一定要存在一个事件，plusready， plusready实际上是原生将桥接js注入到页面中的容器，进行任何方法调用的时候都在plusready之后。所有api方法全部都托管在了一个plus对象中。使用语法plus.模块名称.具体方法（参数，callback）

当我们需要打开系统相册的时候，可以这样做：

Gallery模块管理系统相册，支持从相册中选择图片或视频文件、保存图片或视频文件到相册等功能。通过plus.gallery获取相册管理对象。打开相册plus.gallery.pick进行打开 ，选取多个图片{multiple:true,maximum:9,system:false}

1. 微信支付怎么做？说说流程
2. 申请微信公众号及支付功能申请：根据公众号申请流程申请即可。
3. 获取商户支付配置信息及支付[测试](http://lib.csdn.net/base/softwaretest)配置：

支付授权目录最多可以配置三个域名，测试授权目录只可以一个，这里需要注意的是域名大小写必须要网站URL一致，否则会无法通过授权，提示支付请求的URL不合法。另外，测试支付的微信号必须加到测试白名单，否则无法进行支付测试。

1. H5页面发起支付请求，请求生成支付订单

获取用户授权（获取用户的openid）

1. 调用统一下单API，生成预付单
2. 生成JSAPI页面调用的支付参数并签名，注意时间戳timeStamp是32位字符串
3. 返回支付参数prepay\_id,paySign参数的html文本给前端。
4. 微信浏览器自动调起支付JSAPI接口支付，提示用户输入密码。
5. 确认支付，输入密码，提交支付。
6. 步通知商户支付结果，商户收到通知返回确认信息。
7. 返回支付结果，并发微信消息提示。
8. 展示支付信息给用户，跳转到支付结果页面。
9. 混合开发的注意点

增强WebView：原生WebView基本是PC平台浏览器内核的移植，但对于移动场景并不完全适合，各种硬件API得不到HTML5原生支持。因此对于WebView的种种Hack、增强应运而生，甚至出现了基于增强WebView提供第三方服务的。

路由：应用内跳转由于加入了WebView而变得复杂起来，同时由于组件化、模块化带来的问题，路由也成为人们讨论的重点。

缓存：移动网络条件差，为了用户体验，必须要做资源缓存和预加载。

通信：即HTML5和Native之间的通信。利用系统提供的桥接API可以实现，不过在应用上还有着一些坑点和安全问题。

1. 说说你对手机平台的安装包后缀的理解

Android：\*\*.apk

Ios:\*\*.ipa

Windows: wp7 wp8的是xap

wp8.1以后用8.1的sdk开发的是appx

NodeJS

1. 谈谈你对Socket编程的理解，及实现原理，Socket之间是怎么通讯的

A、Socket定义

Socket是进程通讯的一种方式，即调用这个网络库的一些API函数实现分布在不同主机的相关进程之间的数据交换。几个定义：IP地址：即依照TCP/IP协议分配给本地主机的网络地址，两个进程要通讯，任一进程首先要知道通讯对方的位置，即对方的IP。端口号：用来辨别本地通讯进程，一个本地的进程在通讯时均会占用一个端口号，不同的进程端口号不同，因此在通讯前必须要分配一个没有被访问的端口号。连接：指两个进程间的通讯链路。

B、实现原理

在TCP/IP网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器（Client/Server, C/S）模式，即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。客户/服务器模式的建立基于以下两点：首先，建立网络的起因是网络中软硬件资源、运算能力和信息不均等，需要共享，从而造就拥有众多资源的主机提供服务，资源较少的客户请求服务这一非对等作用。其次，网间进程通信完全是异步的，相互通信的进程间既不存在父子关系，又不共享内存缓冲区，因此需要一种机制为希望通信的进程间建立联系，为二者的数据交换提供同步，这就是基于客户/服务器模式的TCP/IP。

C、通讯过程

服务器端：其过程是首先服务器方要先启动，并根据请求提供相应服务：（1）打开一通信通道并告知本地主机，它愿意在某一公认地址上的某端口（如FTP的端口可能为21）接收客户请求；（2）等待客户请求到达该端口；（3）接收到客户端的服务请求时，处理该请求并发送应答信号。接收到并发服务请求，要激活一新进程来处理这个客户请求（如UNIX系统中用fork、exec）。新进程处理此客户请求，并不需要对其它请求作出应答。服务完成后，关闭此新进程与客户的通信链路，并终止。（4）返回第（2）步，等待另一客户请求。（5）关闭服务器客户端：（1）打开一通信通道，并连接到服务器所在主机的特定端口；（2）向服务器发服务请求报文，等待并接收应答；继续提出请求......（3）请求结束后关闭通信通道并终止。从上面所描述过程可知：（1）客户与服务器进程的作用是非对称的，因此代码不同。（2）服务器进程一般是先启动的。只要系统运行，该服务进程一直存在，直到正常或强迫终止。

详细参见：

https://www.zhihu.com/question/29637351

http://blog.csdn.net/panker2008/article/details/46502783?ref=myread

1. WEB应用从服务器主动推送Data到客户端有哪些方式?

一般的服务器Push技术包括：

（1）基于 AJAX 的长轮询（long-polling）方式，服务器Hold一段时间后再返回信息；

（2）HTTP Streaming，通过iframe和<script>标签完成数据的传输；

（3）TCP 长连接

（4）HTML5新引入的WebSocket，可以实现服务器主动发送数据至网页端，它和HTTP一样，是一个基于HTTP的应用层协议，跑的是TCP，所以本质上还是个长连接，双向通信，意味着服务器端和客户端可以同时发送并响应请求，而不再像HTTP的请求和响应。

（5）nodejs的http://socket.io，它是websocket的一个开源实现，对不支持websocket的浏览器降级成comet / ajax 轮询，http://socket.io的良好封装使代码编写非常容易。

上述的1和2统称为comet技术。comet详细参考：http://www.ibm.com/developerworks/cn/web/wa-lo-comet/

1. 简述Node.js的适用场景？

IO 密集而非计算密集的情景；高并发微数据（比如账号系统）的情景。特别是高并发，Node.js 的性能随并发数量的提高而衰减的现象相比其他 server 都有很明显的优势。

一篇外文：

**Bad Use Cases**：

CPU heavy appsSimple

CRUD / HTML apps

NoSQL + Node.js + Buzzword Bullshit

**Good Use Cases**：

JSON APIs

Single page apps

Shelling out to unix tools

Streaming data

Soft Realtime Applications

详细参见：

http://nodeguide.com/convincing\_the\_boss.html

1. 什么是HTTPS，做什么用的呢？如何开启HTTPS？

（1）什么是HTTPS

https是http的加密版本，是在http请求的基础上，采用ssl进行加密传输。

（2）做什么用

加密数据，反劫持，SEO

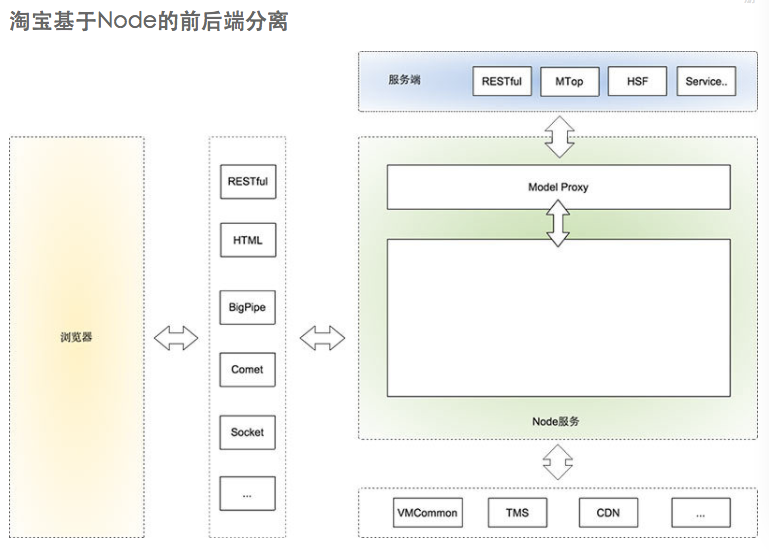
（3）如何开启

生成私钥与证书，配置nginx，重启nginx看效果

详细参见： https://segmentfault.com/a/1190000006199237?utm\_source=tuicool&utm\_medium=referral

1. 你们原来公司如何发送的新消息推送？

参见第47第48题。

1. 如何用NodeJS搭建中间层？

最上端是服务端，就是我们常说的后端。后端对于我们来说，就是一个接口的集合，服务端提供各种各样的接口供我们使用。因为有Node层，也不用局限是什么形式的服务。对于后端开发来说，他们只用关心业务代码的接口实现。

服务端下面是Node应用。

Node应用中有一层Model Proxy与服务端进行通讯。这一层主要目前是抹平我们对不同接口的调用方式，封装一些view层需要的Model。

Node层还能轻松实现原来vmcommon,tms（引用淘宝内容管理系统）等需求。

Node层要使用什么框架由开发者自己决定。不过推荐使用express+xTemplate的组合，xTemplate能做到前后端公用。

怎么用Node大家自己决定，但是令人兴奋的是，我们终于可以使用Node轻松实现我们想要的输出方式:JSON/JSONP/RESTful/HTML/BigPipe/Comet/Socket/同步、异步，想怎么整就怎么整，完全根据你的场景决定。

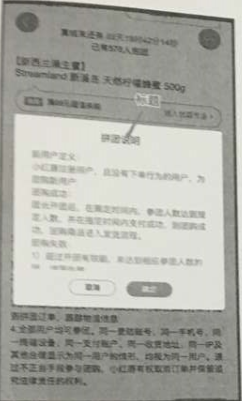
浏览器层在我们这个架构中没有变化，也不希望因为引入Node改变你以前在浏览器中开发的认知。

引入Node，只是把本该就前端控制的部分交由前端掌控。

详细参见：<http://blog.csdn.net/u011413061/article/details/50294263>

组件设计

1. 设计一个弹框组件，组件宽度为屏幕高度的50%，宽度为屏幕宽度的80%，水平垂直居中。组件有header，body，footer三部分，header中有标题，可定制，body区域，footer区域有确定和取消按钮，可定制两个按钮的文字内容，组件外的内容有遮罩，点击遮罩和取消按钮时关闭弹框，参照下图。



使用面向对象封装插件较为合适

构造函数的参数有header的标题及body内容和按钮文字内容

封装的方法应该有 show，hide，在点击遮罩和取消按钮时调用hide方法

并且hide和show方法应该有返回值以供判断。

1. 实现一个手势滑动轮播图组件。效果参考：[https://static.xiaohongchun.com/goods/4514(请在手机里打开)](https://static.xiaohongchun.com/goods/4514(%E8%AF%B7%E5%9C%A8%E6%89%8B%E6%9C%BA%E9%87%8C%E6%89%93%E5%BC%80))

详细参考：<http://www.jb51.net/article/65177.htm>

1. 设计基于观察者模式的事件绑定机制

观察者模式的定义：Observer的意图是定义对象之间的一种一（被观察者）对多（观察者）的关系，当一个对象的状态发生改变时，所有依赖它的对象得到通知，并且会自动更新自己

代码：<http://blog.csdn.net/phker/article/details/6880371>

<http://www.cnblogs.com/LuckyWinty/p/5796190.html>

1. jq自己扩展过什么插件?

弹出层插件、pagination插件、瀑布流插件、模态框插件等

参考：Jquery插件库 jquery之家

1. 侧滑菜单如何实现?

主要依靠两个大的容器来模拟侧滑菜单界面和主界面，把侧滑菜单放到页面右侧看不到的地方，在操作的同时，使用css3过渡、动画或者jq来使两个容器相对运动，实现侧滑菜单效果

参考：<http://www.111cn.net/wy/js-ajax/99687.htm>

1. 权限管理如何实现?

详细参见：代码：<http://blog.csdn.net/liuweidagege/article/details/42497731>

1. 一个大数组，可能存了100万个数字，要从其中取出来第二大的数的下标，有什么快速的方法？

用两个变量max,max2,其中max储存最大值,max2储存第二大值  
初始化的时候,将数组中的第一个元素中较大的存进max中,较小的存进max2中  
然后从第三个元素(下标为2)的元素开始,如果遇到的数比max大,就让max2=max;max等于遇到的数  
一直循环,直到数组尾部  
最后输出max2

单元测试

1. 单个组件怎么测试性能

React 组件测试框架用mocha, 测试库用官方的测试工具库，也可使用第三方库Enzyme，建议使用第三方的。

详细参见： http://www.ruanyifeng.com/blog/2016/02/react-testing-tutorial.html

Vue使用Unit和e2e测试工具：

详细参见： http://www.tuicool.com/articles/6vuINvR

React

1. redux中间件

中间件提供第三方插件的模式，自定义拦截 **action -> reducer**的过程。变为**action -> middlewares -> reducer** 。这种机制可以让我们改变数据流，实现如异步 action ，action 过滤，日志输出，异常报告等功能。

常见的中间件：

redux-logger：提供日志输出

redux-thunk：处理异步操作

redux-promise：处理异步操作，actionCreator的返回值是promise

1. redux有什么缺点

1.一个组件所需要的数据，必须由父组件传过来，而不能像flux中直接从store取。

2.当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新render，可能会有效率影响，或者需要写复杂的shouldComponentUpdate进行判断。

1. react组件的划分业务组件技术组件？

根据组件的职责通常把组件分为UI组件和容器组件。

UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。

两者通过React-Redux 提供connect方法联系起来。

具体使用可以参照如下链接：

http://www.ruanyifeng.com/blog/2016/09/redux\_tutorial\_part\_three\_react-redux.html

1. react生命周期函数

这个问题要考察的是组件的生命周期

一、初始化阶段：

　　getDefaultProps:获取实例的默认属性

　　getInitialState:获取每个实例的初始化状态

　　componentWillMount：组件即将被装载、渲染到页面上

　　render:组件在这里生成虚拟的DOM节点

　　componentDidMount:组件真正在被装载之后

二、运行中状态：

　　componentWillReceiveProps:组件将要接收到属性的时候调用

　　shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回false，接收数据后不更新，阻止render调用，后面的函数不会被继续执行了）

　　componentWillUpdate:组件即将更新不能修改属性和状态

　　render:组件重新描绘

　　componentDidUpdate:组件已经更新

三、销毁阶段：

　　componentWillUnmount:组件即将销毁

1. react性能优化是哪个周期函数？

shouldComponentUpdate 这个方法用来判断是否需要调用render方法重新描绘dom。因为dom的描绘非常消耗性能，如果我们能在shouldComponentUpdate方法中能够写出更优化的dom diff算法，可以极大的提高性能。

详细参考：<https://segmentfault.com/a/1190000006254212>

1. 为什么虚拟dom会提高性能?

虚拟dom相当于在js和真实dom中间加了一个缓存，利用dom diff算法避免了没有必要的dom操作，从而提高性能。

具体实现步骤如下：

* 1. 用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中
  2. 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异
  3. 把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了。

参考链接：<https://www.zhihu.com/question/29504639?sort=created>

1. diff算法?
   1. 把树形结构按照层级分解，只比较同级元素。
   2. 给列表结构的每个单元添加唯一的key属性，方便比较。
   3. React 只会匹配相同 class 的 component（这里面的class指的是组件的名字）
   4. 合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.
   5. 选择性子树渲染。开发人员可以重写shouldComponentUpdate提高diff的性能。

参考链接：<https://segmentfault.com/a/1190000000606216>

1. react性能优化方案

（1）重写shouldComponentUpdate来避免不必要的dom操作。

（2）使用 production 版本的react.js。

（3）使用key来帮助React识别列表中所有子组件的最小变化。

参考链接：

<https://segmentfault.com/a/1190000006254212>

<http://blog.csdn.net/limm33/article/details/50948869>

1. 简述flux 思想

Flux 的最大特点，就是数据的"单向流动"。

1.用户访问 View

2.View 发出用户的 Action

3.Dispatcher 收到 Action，要求 Store 进行相应的更新

4.Store 更新后，发出一个"change"事件

5.View 收到"change"事件后，更新页面

参考链接：<http://www.ruanyifeng.com/blog/2016/01/flux.html>

1. React项目用过什么脚手架？Mern? Yeoman?

Mern：MERN是脚手架的工具，它可以很容易地使用Mongo, Express, React and NodeJS生成同构JS应用。它最大限度地减少安装时间，并得到您使用的成熟技术来加速开发。

参考链接：http://www.open-open.com/lib/view/open1455953055292.html

Vue.js

1. vue与react的对比,如何选型？从性能，生态圈，数据量，数据的传递上，作比较

（1）React 和 Vue 有许多相似之处，它们都有：

使用 Virtual DOM

提供了响应式（Reactive）和组件化（Composable）的视图组件。

将注意力集中保持在核心库，伴随于此，有配套的路由和负责处理全局状态管理的库。

（2）性能：

到目前为止，针对现实情况的测试中，Vue 的性能是优于 React 的。

（3）生态圈

Vue.js: ES6+Webpack+unit/e2e+Vue+vue-router+单文件组件+vuex+iVew

React: ES6+Webpack+Enzyme+React+React-router+Redux

（4）什么时候选择Vue.js

如果你喜欢用（或希望能够用）模板搭建应用，请使用Vue

如果你喜欢简单和“能用就行”的东西，请使用Vue

如果你的应用需要尽可能的小和快，请使用Vue

如果你计划构建一个大型应用程序，请使用React

如果你想要一个同时适用于Web端和原生App的框架，请选择React

如果你想要最大的生态圈，请使用React

详细参考：

http://cn.vuejs.org/v2/guide/comparison.html#React

http://blog.csdn.net/yzh\_2017/article/details/54909166

1. vue slot是做什么的?

简单来说，假如父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。

详细参考：[http://cn.vuejs.org/v2/guide/components.html#使用-Slot-分发内容](http://cn.vuejs.org/v2/guide/components.html#%E4%BD%BF%E7%94%A8-Slot-%E5%88%86%E5%8F%91%E5%86%85%E5%AE%B9)

1. vue和angular的优缺点以及适用场合?

参见：《 在选择框架的时候要从哪方面入手》一题

1. vue路由实现原理?

以官方仓库下 examples/basic 基础例子来一点点具体分析整个流程。

和流程相关的主要需要关注点的就是 components 、 history 目录以及 create-matcher.js 、 create-route-map.js、index.js 、 install.js。

从入口，作为插件，实例化 VueRouter，实例化 History，实例化 Vue，defineReactive 定义 \_route，router-link 和 router-view 组件等几个方面展开分析

必须参见： http://www.tuicool.com/articles/jQRnIrF

1. 你们vue项目是打包了一个js文件，一个css文件，还是有多个文件？

根据vue-cli脚手架规范，一个js文件，一个CSS文件。

详细参见：

http://blog.csdn.net/lx376693576/article/details/54911340

https://zhuanlan.zhihu.com/p/24322005

1. vue遇到的坑，如何解决的？

Vue1.0升级2.0有很多坑: 生命周期；路由中引入静态js，全局组件，全局变量，全局function；v-for循环的key，value值互换了位置，还有track-by；filter过滤器；遍历数组时，key值不能做model；父子通信等。

其他坑详见：

http://blog.csdn.net/lemon\_zhao/article/details/55510589

<https://segmentfault.com/a/1190000008347498>

http://www.tuicool.com/articles/aUrmumV

1. vue的双向绑定的原理，和angular的对比

在不同的 MVVM 框架中，实现双向数据绑定的技术有所不同。

AngularJS 采用“脏值检测”的方式，数据发生变更后，对于所有的数据和视图的绑定关系进行一次检测，识别是否有数据发生了改变，有变化进行处理，可能进一步引发其他数据的改变，所以这个过程可能会循环几次，一直到不再有数据变化发生后，将变更的数据发送到视图，更新页面展现。如果是手动对 ViewModel 的数据进行变更，为确保变更同步到视图，需要手动触发一次“脏值检测”。

VueJS 则使用 ES5 提供的 Object.defineProperty() 方法，监控对数据的操作，从而可以自动触发数据同步。并且，由于是在不同的数据上触发同步，可以精确的将变更发送给绑定的视图，而不是对所有的数据都执行一次检测。

详细参见： http://www.jianshu.com/p/d3a15a1f94a0

1. vue-cli，脚手架

安装：$ npm install -g vue-cli

使用：$ vue init <template-name> <project-name>

webpack配置详解： https://zhuanlan.zhihu.com/p/24322005

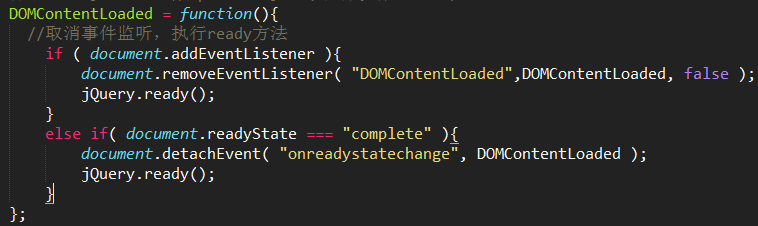
1. Vue里面router-link在电脑上有用，在安卓上没反应怎么解决？

Vue路由在Android机上有问题，babel问题，安装babel polypill 插件解决。

框架底层

1. jQuery源码中值得借鉴的？
2. 使用模块化思想，模块间保持独立，不会导致多个开发人员合作时产生的冲突。
3. 在设计程序时，要结构清晰，高内聚，低耦合。
4. 利用多态的方式，实现方法的重载，提高代码的复用率
5. jQuery的链式调用以及回溯
6. jQuery.fn.extend 与 jQuery.extend方法来实现扩展静态方法或实例方法
7. $.ready是怎么实现的?

jquery.ready()和window.onload，window.onload事件是在页面所有的资源都加载完毕后触发的. 如果页面上有大图片等资源响应缓慢, 会导致window.onload事件迟迟无法触发.所以出现了DOM Ready事件. 此事件在DOM文档结构准备完毕后触发, 即在资源加载前触发.

jQuery中的ready方法实现了当页面加载完成后才执行的效果，但他并不是window.onload或者doucment.onload的封装，而是使用 标准W3C浏览器DOM隐藏api和IE浏览器缺陷来完成的 。 

1. 懒加载的实现原理?

意义： 懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

实现原理：先加载一部分数据，当触发某个条件时利用异步加载剩余的数据，新得到的数据不会影响原有数据的显示,同时最大程度上减少服务器端的资源耗用。

实现方式：

1.第一种是纯粹的延迟加载，使用setTimeOut或setInterval进行加载延迟.

2.第二种是条件加载，符合某些条件，或触发了某些事件才开始异步下载。

3.第三种是可视区加载，即仅加载用户可以看到的区域，这个主要由监控滚动条来实现，一般会在距用户看到某图片前一定距离便开始加载，这样能保证用户拉下时正好能看到图片。

1. 双向数据绑定和单向数据的区别?

（1）单向数据流中，父组件给子组件传递数据，但反过来不可以传递，也就是说单向数据流是从最外层节 点传递到子节点，他们只需从最外层节点获取props渲染即可，如果顶层组件的某个prop改变了，React会递归的向下便利整棵组件树，重新 渲染所有使用这个属性的组件，React组件内部还具有自己的状态，这些状态只能在组件内修改；双向数据绑定是数据与视图双向绑定，数据发生改变时，视图也改 变，视图发生改变时，数据也会发生改变。

（2）双向数据绑定的各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到；单向数据流的数据流动方向可以跟踪，流动单一，追查 问题的时候可以更快捷，缺点是写起来不太方便，要使视图发生改变就得创建各种action来维护state。

（3）双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。而在践行单向数据流的flux系的实现中，其实不过是在全局搞了一个单例的事件分发器 (dispatcher)，开发者必须显式地通过这个统一的事件机制做数据变更通知。

1. 怎么实现一个类似于const功能的方法？

es6中const相当于声明常量不可更改，我们利用defineProperty可以模拟实现；我们把 writable 设置为 false 的时候，该属性就成了只读，也就满足了常量了性质，我们把常量封装在CONST命名空间里面,但是因为我们依然可以通过修改属性writable为true修改属性值,所以configurable设置为false，不能修改属性；

模拟：

如下代码CONST.a相当于es6中cont a=2；CONST.a是不可以更改得常量；

var CONST = {};

Object.defineProperty(CONST, 'a', {

value: 2,

writable: false,

configurable: false,

enumerable: true //可枚举

});

console.log(CONST.a); //2

CONST.a = 3;

console.log(CONST.a); //2

1. 使用原生js模拟一个apply方法  
   //bind()方法就是为被选元素绑定事件，并规定事件触发时执行的函数;但是是jQuery事件，我们可以通过给Function的prototype原型进行扩展，为IE6-8自定义bind方法

Function.prototype.bind = function(target){

var self = this;//存this，指向事件触发时应该执行的函数

var args = Array.prototype.slice.call(arguments);

//因为arguments是伪数组，而我们需要用到数组的slice方法，所以通过call方法，使arguments拥有slice方法

//console.log(args);

//使用闭包，传入外部变量。

return function(){

return self.apply(target,args.slice(1));

//核心。也就是说 当事件触发时 target应该执行此时self指向的函数，该函数接受一个实参 args.slice(1);

}

}

var btn = document.getElementById('btn');

var box = document.getElementById('box');

btn.onclick = function(color){

//console.log(this);

this.style.backgroundColor = color;

}.bind(box,'skyblue')

1. Object.create()和直接创建对象有什么区别？

Object.create()方法创建一个拥有指定原型和若干个指定属性的对象

//Object.create(proto,[propertiesObject])

该方法创建一个对象，其接受两个参数，第一个参数是这个对象的原型对象proto,

第二个是一个可选参数，用以对对象的属性做进一步描述

如果proto 参数不是null或一个对象值，则抛出一个TypeError异常

var obj1 = Object.create({

x: 1,

y: 2

}); //对象obj1继承了属性x和y

var obj2 = Object.create(null); //对象obj2没有原型

对象字面量是创建对象最简单的一种形式，

目的是在于简化创建包含大量属性的对象的过程。

对象字面量由若干属性名(keys)和属性值(values)成对组成的映射表，

key和value中间使用冒号(:)分隔，

每对key/value之间使用逗号(,)分隔，

整个映射表用花括号({})括起来。

在用字面量来创建对象的时候，对象中的property定义可以用单引号或双引号来包括，也可以忽略引号。不过，当property中出现空格、斜杠等特殊字符，或者使用的property与JS关键词冲突时，则必须使用引号。

var obj = {

property\_1: value\_1, // property\_# 可能是一个标识符...

2: value\_2, // 或者是一个数字

"property n": value\_n // 或是一个字符串

}

通过对象字面量创建的对象复用性较差，

使用Object.create()创建对象时不需要定义一个构造函数就允许你在对象中选择其原型对象。

1. 使用for in 遍历对象和使用Object.keys来遍历对象有什么区别？

1、for in 主要用于遍历对象的可枚举属性，包括自有属性、继承自原型的属性

2、Object.keys 返回一个数组，元素均为对象自有的可枚举属性

3、Object.getOwnProperty 用于返回对象的自有属性，包括可枚举的和不可枚举的

var obj = {

"name":'lh',

"age":27

}

Object.defineProperty(obj,"height",{value:178,enumerable:false})

Object.prototype.prototype1 = function(){

console.log('aaa')

}

Object.prototype.prototype2 = 'bbb';

//for in

for(var i in obj){

console.log(i); // name age prototype1 prototype2

}

//Object.keys

console.log(Object.keys(obj)) //name age

//Object.getOwnProperty

console.log(Object.getOwnPropertyNames(obj)) //name age height

1. 深拷贝和浅拷贝以及应用场景

1.浅拷贝

//拷贝就是把父对像的属性，全部拷贝给子对象。

var Chinese = {

　　nation:'中国'

}

var Doctor = {

　　career:'医生'

}

function extendCopy(p) {

　　　　var c = {};

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

　　　　}

　　　　c.uber = p;

　　　　return c;

　}

//使用的时候，这样写：

Doctor = extendCopy(Chinese);

Doctor.career = '医生';

alert(Doctor.nation); // 中国

//但是，这样的拷贝有一个问题。那就是，如果父对象的属性等于数组或另一个对象，那么实际上，子对象获得的只是一个内存地址，而不是真正拷贝，因此存在父对象被篡改的可能。

//请看，现在给Chinese添加一个"出生地"属性，它的值是一个数组。

Chinese.birthPlaces = ['北京','上海','香港'];

//通过extendCopy()函数，Doctor继承了Chinese。

Doctor = extendCopy(Chinese);

//然后，我们为Doctor的"出生地"添加一个城市：

Doctor.birthPlaces.push('厦门');

//看一下输入结果

alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

alert(Chinese.birthPlaces); //北京, 上海, 香港, 厦门

//结果是两个的出生地都被改了。

//所以，extendCopy() 只是拷贝了基本类型的数据，我们把这种拷贝叫做“浅拷贝”。

2.深拷贝

//因为浅深拷有如此弊端所以我们接下来看一下深拷贝

//所谓"深拷贝"，就是能够实现真正意义上的数组和对象的拷贝。它的实现并不难，只要递归调用"浅拷贝"就行了。

var Chinese = {

　　nation:'中国'

}

var Doctor = {

　　career:'医生'

}

function deepCopy(p, c) {

　　　　var c = c || {};

　　　　for (var i in p) {

　　　　　　if (typeof p[i] === 'object') {

　　　　　　　　c[i] = (p[i].constructor === Array) ? [] : {};

　　　　　　　　deepCopy(p[i], c[i]);

　　　　　　} else {

　　　　　　　　　c[i] = p[i];

　　　　　　}

　　　　}

　　　　return c;

　　}

//看一下使用方法：

Doctor = deepCopy(Chinese);

//现在，给父对象加一个属性，值为数组。然后，在子对象上修改这个属性：

Chinese.birthPlaces = ['北京','上海','香港'];

Doctor.birthPlaces.push('厦门');

alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

alert(Chinese.birthPlaces); //北京, 上海, 香港

JavaScript 中的对象一般是可变的（Mutable），因为使用了引用赋值，新的对象简单的引用了原始对象，改变新的对象将影响到原始对象。如 `foo={a: 1}; bar=foo; bar.a=2` 你会发现此时 `foo.a` 也被改成了 `2`。

虽然这样做可以节约内存，但当应用复杂后，这就造成了非常大的隐患，Mutable 带来的优点变得得不偿失。为了解决这个问题，一般的做法是使用 shallowCopy（浅拷贝）或 deepCopy（深拷贝）来避免被修改，但这样做造成了 CPU 和内存的浪费。

Immutable 可以很好地解决这些问题。

1. 原型链，闭包与继承

闭包的好处：

1.不会污染全局环境;

2.可以进行形参的记忆,减少形参的个数,延长形参生命周期;

function add(x) {

return function(y) {

return (x+y);

}

}

var sum = add(2);

sum(5);//结果为7

3.方便进行模块化开发;

var module = (function() {

var name = '123';

function init() {

console.log(name);

}

return {

getname:init

}

})()

module.getname();//结果为123;

继承：一个构造函数继承另一个构造函数中的方法;可以省去大量的重复。

function Man(name,age) {

this.name = name;

this.age = age;

}

var person = new Man('tom',19);

function Woman(name,age) {

this.sex = 'woman';

Man.call(this,name,age);

}

Woman.prototype = Man.prototype;

var person1 = new Woman('july',20);

person1.name//结果为 july

person1.age //结果为 20

person1.sex //结果为 woman

原型链查找：进行方法调用的时候，会现在实例自身上找，如果没有就去该实例的原型上找。

function People() {

this.name = 'a People';

}

People.prototype.say = function() {

this.age = '10';

console.log(this.name,this.age);

}

var person = new People();

person.say();

AngularJS

1. 对bootstrap的掌握、为什么用angular+bootstrap搭建后台管理系统

一句话总结：

{{

bootstrap 是一个快速开发的响应式框架，主要是为了快速搭建ui界面，bootstrap的web组件和js插件对pc端开发比较友好，尤其是栅格化系统可以良好兼容浏览器，低版本浏览器可以使用 bootstrap-responsive的插件兼容，js插件有各种回调机制，可以满足自己的多样开发需求，

而且bootstrap 使用 css属性来操作样式，免去了手写原生代码的痛苦

使用angular 进行数据绑定，bootstrap来搭建界面，提升开发效率

}}

个人心得：

{{

我在实际开发中使用 ace admin这套基于boostrap的框架，可以更快速的开发，数据项通过json

结构进行配置，几乎不用手写代码，提升开发效率

}}

1. angular 中ng-if 和ng-show/hide 有什么区别？

一句话总结：

{{

ng-if 是通过布尔值的判断来移除添加某个节点，而ng-show/hide 是通过 布尔值来控制节点的显示／隐藏，都是控制了css的display属性

}}

个人心得：

{{

ng-if 添加删除节点，那么肯定回创建作用域，而ng-show/hide 则不会

}}

1. Angular 中ng-click中写的表达式，可以用js原生上的方法吗？为什么？

一句话总结：

{{

ng-click 和原生事件完成的功能是一样的，但是ng-click做了优化，而且ng-click里面可以写表达式，运算过程，click 则要单独处理，手写功能。

}}

个人心得：

{{

如果不在作用域里添加函数，可以配合ng-init初始化属性值，在ng-click里添加算法或者某一功能，虽然ng-init不推荐使用，但是侧面说明ng-click的优势

}}

一句话总结：

{{

货币过滤器 currency 大小写过滤器 toUppercase toLowercase 排序过滤器orderby 日期过滤器 date 字符串过滤器 limitto json格式化过滤器

数字过滤器 number 还有filter 筛选过滤器

}}

• 如何自定义filter?

一句话总结：

{{

在模块下挂在一个filter()方法，第一个参数传入过滤器的名字，第二个参数是回调函数，处理过滤方法的详细内容，最后返回结果，这样外部就可以根据过滤器的名字调用了

例如

myAppModule.filter("reverse",function(){

return function(input,uppercase){

var out = "";

for(var i=0 ; i<input.length; i++){

out = input.charAt(i)+out;

}

if(uppercase){

out = out.toUpperCase();

}

return out;

}

});

使用： name | reverse 通过管道符调用

}}

1. 内置filter都有哪些？

一句话总结：

{{

factory ,service ,provider都是angular提供的服务，

factory 就是原生js里的方法，一个简单的函数

service 类似原生里构造函数的过程，拥有一个构造器constructor，也就是说有new的过程，追加属性和方法都是在this上追加的

provider 是服务商 当service 需要配置的时候，需要使用provider提供服务，例如当使用angular进行跨域访问，需要配置jsonp信息的时候，就可以使用provider进行config的配置，简单理解是service的高级版本，provider提供一个$get的属性来返回$provider的实例

他们都是单例模式，只实例化一次

}}

个人心得：

{{

个人理解 provider > service > factory，

factory用来配置简单的服务

service是在factory的基础之上加入了面向对象的思想，提供更多功能的服务

provider是在service的基础上进一步改进配置信息

factory与service在底层代码上都来源于provider

例子介绍:

我可以在factory里写一个$http()请求，不做任何配置，参数写死

我可以在service里写一个$http()请求，传入请求的参数可以先配置在this的属性上传入方法

我可以在provider里写一个请求，然后在config上传入要配置的 参数，URL，method，data等信息，通过config来修改provider的参数，再将服务商提供的服务注入控制器controller

注意事项：

config里传入的参数是 nameProvider 而不是 name，也就是说你的叫做myProvider,config 里传入的参数就是myProviderProvider

而不是myProvider

}}

1. 如何自定义filter?

一句话总结：

{{

在模块下挂在一个filter()方法，第一个参数传入过滤器的名字，第二个参数是回调函数，处理过滤方法的详细内容，最后返回结果，这样外部就可以根据过滤器的名字调用了

例如

myAppModule.filter("reverse",function(){

return function(input,uppercase){

var out = "";

for(var i=0 ; i<input.length; i++){

out = input.charAt(i)+out;

}

if(uppercase){

out = out.toUpperCase();

}

return out;

}

});

使用： name | reverse 通过管道符调用

}}

1. factory、service 和 provider 是什么关系？

一句话总结：

{{

factory ,service ,provider都是angular提供的服务，

factory 就是原生js里的方法，一个简单的函数

service 类似原生里构造函数的过程，拥有一个构造器constructor，也就是说有new的过程，追加属性和方法都是在this上追加的

provider 是服务商 当service 需要配置的时候，需要使用provider提供服务，例如当使用angular进行跨域访问，需要配置jsonp信息的时候，就可以使用provider进行config的配置，简单理解是service的高级版本，provider提供一个$get的属性来返回$provider的实例

他们都是单例模式，只实例化一次

}}

个人心得：

{{

个人理解 provider > service > factory，

factory用来配置简单的服务

service是在factory的基础之上加入了面向对象的思想，提供更多功能的服务

provider是在service的基础上进一步改进配置信息

factory与service在底层代码上都来源于provider

例子介绍:

我可以在factory里写一个$http()请求，不做任何配置，参数写死

我可以在service里写一个$http()请求，传入请求的参数可以先配置在this的属性上传入方法

我可以在provider里写一个请求，然后在config上传入要配置的 参数，URL，method，data等信息，通过config来修改provider的参数，再将服务商提供的服务注入控制器controller

注意事项：

config里传入的参数是 nameProvider 而不是 name，也就是说你的叫做myProvider,config 里传入的参数就是myProviderProvider

而不是myProvider

}}

1. angular 的数据绑定采用什么机制？详述原理

一句话总结：

{{

通过 $watch 来监听 每一次dom的变化，然后$digest 来遍历循环所有的 $watch 队列，发现与原来不同的值，也就是脏值则进行修改，最后通知$apply ，$apply会进入 angular context的执行环境，通知浏览器拿回控制权，修改相应的dom节点

}}

个人心得：

{{

每一个ng指令的触发都在内部触发了一个$Watch的队列，加入 一组标签

<li ng-repeat="item in items">

{{ item }}

</li>

循环了10次，那么就触发了10个item与1个ng-repeat的 11 个$watch的队列，

$digest 会遍历循环这些队列，比较值得变化，有变化的即为脏值 过程叫做 dirty-checking，$digest修改完对应的值就会 通知$apply()准备进入 angular context的执行阶段 修改dom，没有变化则不修改 。也就是说我们在页面每次触发的操作，每次输入一个文字都会触发$watch，可见于react相比angular的劣势出现了

}}

1. 两个平级界面块 a 和 b，如果 a 中触发一个事件，有哪些方式能让 b 知道？详述原理

一句话总结：

{{

1.通过a的 子controller 将事件使用$emit传递给 父controller 再将 事件用$broadcast传递给 b controller 实现数据传递

2.也可以通过service服务，将数据保存在service之内，然后在b中调用service

}}

个人心得：

{{

像这种数据传递的方式其实有很多种，本质是不同作用于之间的数据传递，只要掌握住这一点思想有很多方式解决，比如 我可以尝试挂在 $rootScope之上进行共享，也可以用 本地存储来存储数据，实现数据共享。方法不重要，关键是如何解决的思路。

}}

1. 一个 angular 应用应当如何良好地分层？

目录结构的划分

对于小型项目，可以按照文件类型组织，比如：

css

js

controllers

models

services

filters

templates

但是对于规模较大的项目，最好按业务模块划分，比如：

css

modules

account

controllers

models

services

filters

templates

disk

controllers

models

services

filters

templates

modules 下最好再有一个 common 目录来存放公共的东西

1. angular 应用常用哪些路由库，各自的区别是什么？

一句话总结：

{{

ng-router,ui-touter ,ui-router可以嵌套子视图

}}

1. 如果通过angular的directive规划一套全组件化体系，可能遇到哪些挑战？

一句话总结：

{{

隔离作用域，ng-指令的 作用域传递

}}

1. 分属不同团队进行开发的 angular 应用，如果要做整合，可能会遇到哪些问题，如何解决？

一句话总结：

{{

可能会遇到不同模块之间的冲突。比如一个团队所有的开发在 moduleA 下进行，另一团队开发的代码在 moduleB 下

angular.module('myApp.moduleA', [])

.factory('serviceA', function(){

...

})

angular.module('myApp.moduleB', [])

.factory('serviceA', function(){

...

})

angular.module('myApp', ['myApp.moduleA', 'myApp.moduleB'])

会导致两个 module 下面的 serviceA 发生了覆盖。

}}

个人心得：

{{

没有太好的解决方案，只能约定命名规范

}}

1. angular 的缺点有哪些？

一句话总结：

{{

不适合做交互过多的项目，因为没有选择器的存在，

导致学习成本较高，对前端不友好。但遵守 AngularJS 的约定时，生产力会很高，对 Java 程序员友好

因为所有内容都是动态获取并渲染生成的，搜索引擎没法爬取

}}

1. 如何看待 angular 1.2 中引入的 controller as 语法？

一句话总结：

{{

为angular 添加 this 作用域链，使得angular更加想原声写法

}}

1. 详述 angular 的 “依赖注入”。

一句话总结：

{{

依赖注入是一个在组件中给出的替代了硬的组件内的编码它们的依赖关系的软件设计模式。这减轻一个组成部分，从定位的依赖，依赖配置。这有助于使组件可重用，维护和测试。

AngularJS提供了一个至高无上的依赖注入机制。它提供了一个可注入彼此依赖 constant value factory service provide 核心组件。

}}

1. 当你简单的动态给页面插入html时，此时html带有angular的语法不会执行的，为什么？

一句话总结：

{{

通过$compile进行处理,任何指令的生效都需要compile，这一步在app启动的时候angular先帮你做了，但你插入的html是没有经过compile这个步骤的，所以你手动compile下即可。

}}

• 使用ng-repeat出错：Error: [ngRepeat:dupes]，怎么回事？

一句话总结:

{{

因为angular 不允许数组中出现重复的值，所以会报错dupes错误，意思是重复的参数错误

在 ng-repeat="itme in items" 加入 track by $index

ng-repeat="itme in items track by $index"

就可解决

}}

1. 使用ng-repeat出错：Error: [ngRepeat:dupes]，怎么回事？

Error: [ngRepeat:dupes]这个出错提示具体意思是指数组中有2个以上的相同数字。

ngRepeat不允许collection中存在两个相同Id的对象

对于数字对象来说，它的id就是它自身的值，因此，数组中是不允许存在两个相同的数字的。为了规避这个错误，需要定义自己的track by表达式。

例如：item in items track by item.id 或者 item in items track by fnCustomId(item)。嫌麻烦的话，直接拿循环的索引变量$index来用item in items track by $index

1. ng-repeat迭代数组的时候，如果数组中有相同值，会有什么问题，如何解决？

见上题

1. 使用第三方插件或者原生的js修改angular中的model或者view的值时，相应的model或者view的值不会变化，也就是angular的双向数据绑定失效，怎么回事？

angular有自己的一个上下文，所有与angular有关的代码执行（如双向数据绑定）都在这个上下文中进行，因此如果你用第三方插件或者原生的js进行操作时，此时代码是在javascript的上下文中执行，angular无法知道你是否修改model或者view的值，自然也就无法进行双向数据绑定。解决方案时在操作之后执行$scope.$apply()或者将操作的代码放在$scope.$apply(function(){//操作的代码...})

1. angular中注入方式有推断式注入、$inject注入、内联式注入，当然这三种方式在angular中是等效的，但推断式注入对于压缩的 JavaScript 代码来说是不起作用的，为什么？

尽量不要用推断式注入，最佳是用内联式注入的方式。

1. 如何看待angular2？

相比 Angular1.x，Angular2的改动很大，几乎算是一个全新的框架。

基于 TypeScript（可以使用 TypeScript 进行开发），在大型项目团队协作时，强语言类型更有利。

组件化，提升开发和维护的效率。

还有 module 支持动态加载，new router，promise的原生支持等等。

迎合未来标准，吸纳其他框架的优点，值得期待，不过同时要学习的东西也更多了（ES next、TS、Rx等）

详细参考：

<http://www.tuicool.com/articles/ymmq2mf>

http://www.cnblogs.com/laixiangran/p/4938732.html

综合问题

1. 请列举你知道的前端框架？常用的前端开发工具？开发过哪些应用和组件？

（1）前端框架

bootstrap/jQuery/zepto/backbone/AngularJS/vue.js/React/React Native/小程序

（2）前端开发工具

gulp/webpack/git/svn/npm/linux

（3）应用和组件

根据自己做的项目对答

1. 项目测试没问题。但是放到线上就有问题了，你是怎么分析解决的？

可能的原因：

（1）后端原因：后端接口，后端服务器

（2）域名、IP和路径问题

（3）网络环境问题

（4）线上库、框架、工具的版本和本地不一致问题

（5）线上和本地数据资源不一致问题

（6）程序bug

1. ES6里面你用过什么?

（1）块作用域 - let

（2）衡量 - const

（3）解构数组 - Array Destructuring

（4）解构对象 - Object Destructuring

（5）模板字符串 - Template Strings

（6）展开操作符

（7）剩余操作符

（8）解构参数

（9）箭头函数

（10）对象表达式

（11）对象属性名

（12）对比两个值是否相等

（13）把对象的值复制到另一个对象里

（14）设置对象的 prototype

（15）\_\_proto\_\_

（16）supper

（17）迭代器

（18）class 类

（19）get set

（20）静态方法

（21）继承

（22）模块化

细节参见： <http://es6.ruanyifeng.com/>

1. 如何管理团队?

（1）区分技术和管理角色在意识上的差异

（2）时间管理

（3）同时管理自己和其他人的代码

（4）赢得团队的尊敬

详细参见： http://www.t262.com/read/187780.html

1. 你做过的你负责的最难的数据交互模块是？

根据自己做的项目对答。

1. 你平时写过什么业务逻辑？

根据自己做的项目对答。

1. 你负责的具体是什么模块？

根据自己做的项目对答。

1. 你掌握的技术栈有哪些？

说自己了解的和擅长的。

**好三面试题整理**

### sroll resize 使用函数节流实现不要频繁触发事件的需求

|  |
| --- |
| 利用定时器，让函数执行延迟500毫秒，在500毫秒内如果有函数又被调用则删除上一次调用，这次调用500毫秒后执行，其实本意只是窗口resize后页面做一些调整就可以，而window的resize事件并不是在resize结束后才触发的，具体则么个频率我也不知道，但却是在不停的调用，直到窗口大小不再变化。其实类似的机制还有鼠标的mousemove，都是在短时间内重复触发。  <script type="text/javascript">  function resizehandler(){  console.log(new Date().getTime());  }  function throttle(method,context){  clearTimeout(method.tId);  method.tId=setTimeout(function(){  method.call(context);  },500);  }  window.onresize=function(){  throttle(resizehandler,window);  };  </script>  throttle()函数接受两个参数 ，要执行的函数以及在哪个作用域中执行。上面这个函数首先清除之前设置的任何定时器。定时器 ID 是存储在函数的 tId 属性中的，第一次把方法传递给 throttle()的时候，这个属性可能并不存在。接下来，创建一个新的定时器，并将其 ID 储存在方法的 tId 属性中。 如果这是第一次对这个方法调用 throttle()的话，那么这段代码会创建该属性。定时器代码使用call()来确保方法在适当的环境中执行。如果没有给出第二个参数，那么就在全局作用域内执行该方法。 |

### webpack 打包谈谈怎么优化的

|  |
| --- |
| 参考链接：[http://www.jianshu.com/p/908c48ced871](https://github.com/hawx1993/tech-blog/issues/3)   优化方案   （以下是vuecli的webpack中的一些配置）    可以让require模块的定位更快   1. resolve.alias别名是 Webpack 的一个配置项，它的作用是把用户的一个请求重定向到另一个路径 。如果在alias中定义了值，然后在代码中require该值的时候，会重定向到alias指定的位置。 |

### 手写代码 监测数组变化 并返回数组长度

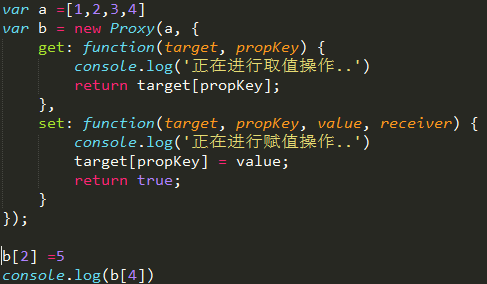
|  |
| --- |
| **利用ES6的extends实现（通过ES6的关键字extends实现继承完成Array原型方法的重写）** |

### vue react angular 怎么检测数据变化的

* 1. Angular
     1. 在angular版本里面还是采用脏值检测来检测数据的变更的，但是和angularjs不一样的是,angular引入了[zone.js](https://github.com/angular/zone.js" \t "https://segmentfault.com/q/_blank)来处理数据的变更。性能可以达到angularjs脏值检测的3到10倍。详细的文章和资料可以谷歌一下有很多介绍。
  2. Vue
     1. vue.js 则是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调
  3. React

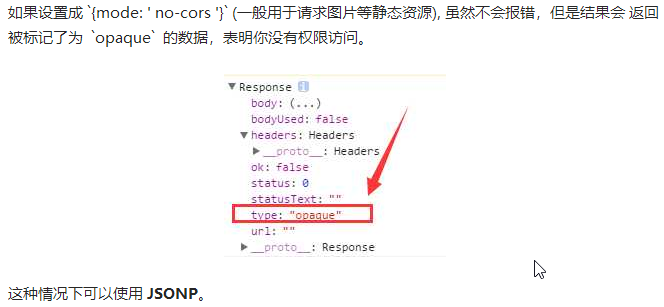
react状态变化只能通过setState,调用setState就会更新状态重新渲染dom

### ES6中，数组监测怎么实现的（代理）

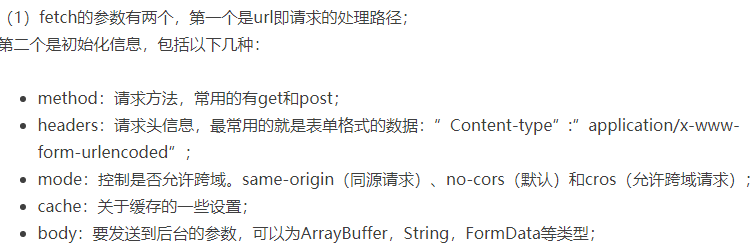


### 关于fetch跨域做了什么配置





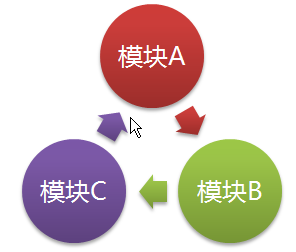






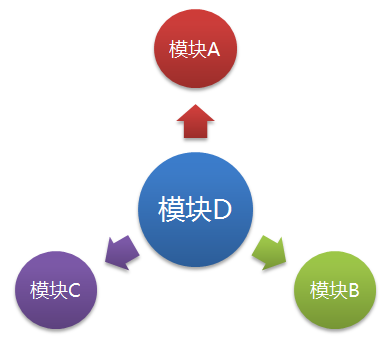
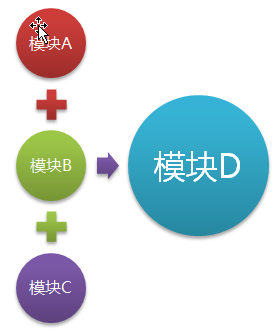
### 模块化循环依赖怎么解开

1 使用build-helper-maven-plugin插件规避。



如图所示，A,B,C互相依赖，可以合并为一个中间工程，编译为临时模块D，然后A,B,C分别依赖临时模块D进行编译。这种方法可以解决无法构建问题，但只是一个规避措施，工程关系依然是混乱的。

方法二：通过重构，从根本上消除循环依赖

88

### 深拷贝和浅拷贝

* **浅拷贝**

#### **拷贝原对象的引用**

这是最简单的浅拷贝。例：

|  |  |
| --- | --- |
| 1  2  3  4  5 | var a = [{c:1}, {d:2}];  var b = a.slice();  console.log(a === b); // 输出false，说明外层数组拷贝的是实例  a[0].c = 3;  console.log(b[0].c); // 输出 3，说明其元素拷贝的是引用 |

|  |  |
| --- | --- |
|  | 属性拷贝引用。 var a = {c:1};  var b = a;  console.log(a === b); // 输出true。  a.c = 2;  console.log(b.c); // 输出 2 |

这种情况，外层源对象是拷贝实例，如果其属性元素为复杂杂数据类型时，内层元素拷贝引用。

对源对象直接操作，不影响两外一个对象，但是对其属性操作时候，会改变两外一个对象的属性的只。

常用方法为：**Array.prototype.slice(), Array.prototype.concat(), jQury的$.extend({},obj)**，例：

        1.1

function simpleClone(initalObj) {

        var obj = {};

        for ( var i in initalObj) {

            obj[i] = initalObj[i];

        }

        return obj;

    }

    1.2Object.assign({},obj) 拷贝的是属性的引用，浅拷贝

* 2.深拷贝

(常见方法有

JSON.parse(),

JSON.stringify()，

jQury的$.extend(true,{},obj)，

lodash的\_.cloneDeep

\_.clone(value, true)。例：

)

        2.1 Json.parse()

  function deepClone(initalObj) {

        var obj = {};

        try {

         obj = JSON.parse(JSON.stringify(initalObj));

        }

        return obj;

     }

    //确定，抛弃了对象的constructor，拷贝之后只是一个Object，RegExp对象不能采用这种方法拷贝

        2.2 递归拷贝

function deepClone(initalObj, finalObj) {

        var obj = finalObj || {};

        for (var i in initalObj) {

            // 避免相互引用对象导致死循环，如initalObj.a = initalObj的情况

            if(initalObj[i] === obj) {

                continue;

            }

            if (typeof initalObj[i] === 'object') {

                obj[i] = (initalObj[i].constructor === Array) ? [] : {};

               arguments.callee(initalObj[i], obj[i]);

            } else {

                obj[i] = initalObj[i];

            }

        }

        return obj;

}

### this指向

1.函数在被直接调用的时候，其中的this指针永远指向window  
 2.匿名函数this总是指向window对象  
 3.谁执行函数，this就指向谁

4.如果函数new了一下，那么就会创建一个对象，并且this指向新创建的对象

//参考文档 <http://www.jianshu.com/p/8ad9a3b9e416>

### sass 语法编写for split 3张背景图

//定义背景图片路径这里由颜色代替

$red : #FF0000;

$orange : #FFA500;

$yellow : #FFFF00;

$green : #008000;

$bluegreen : #00FFFF;

$blue : #0000FF;

$purple : #800080;

//定义一个list储存背景颜色

$bgcolorlist: $red $orange $yellow $green $bluegreen $blue $purple;

//使用sass for循环语句为每一个li设置background

@for $i from 1 to length($bgcolorlist)+1 {

#main-container ul li:nth-child(#{$i})

{ background-color:nth($bgcolorlist,$i); }

}

### Jquery 优点 缺点

优点

1.JQuery实现脚本与页面的分离

2.最少的代码做最多的事情

3.性能支持比较好

4.它是一个“标准”

5.插件开发

6.节约学习成本

7.让JS编程变得有趣

缺点

1. 不能向后兼容
2. 插件兼容性
3. 多个插件冲突
4. Jquery的稳定性
5. 对动画和特效的支持差

//支持文档https://www.cnblogs.com/SanMaoSpace/archive/2013/06/15/3137905.html

### 防止频繁操作DOM做了哪些优化建议

以一个button的按钮为例子可以使用策略模式,点击一次后让按钮灰掉，暂时不可以 用,一段时间后再可以用。

主要的解决思想是函数节流

//参考文档https://www.cnblogs.com/dolphinX/p/3403821.html

### vue img图片路径是绝对还是相对

服务器中返回的图片路径一半都是绝对的

如果是本地静态的图片资源需要写相对的路径

### ES6 class关键字 原理 跟function什么区别

在function定义的构造函数中，其prototype.constructor属性指向构造器自身   
 在class定义的类中，constructor其实也相当于定义在prototype属性上

**重复定义**

function会覆盖之前定义的方法

class会报错

## **原型或者类中方法的枚举**

class中定义的方法不可用Object.keys(Point.prototype)枚举到

function构造器原型方法可被Object.keys(Point.prototype)枚举到，除过

constructor

所有原型方法属性都可用Object.getOwnPropertyNames(Point.prototype)访问到

## 都可通过实例的\_\_proto\_\_属性向原型添加方法

## class没有变量提升

## class定义的类没有私有方法和私有属性

// 参考文档http://blog.csdn.net/u012657197/article/details/77542404

### iframe跨域问题 页面之间怎么传值

解决方法一

同域下： 使用 window.top或者window.parent.parent获取浏览器最顶层window对象的引用。 可以使用：window.parent.parent.document最顶层元素的DOM对象 window.parent.parent.document.getElementsByTagName("html")来获取父页面的HTML元素。

解决方法二：

不同域名下： iframe在跨域访问的时候会有严格的要求，比ajax跨域请求还要难解决 浏览器判断是否跨域会根据两种情况，一个是网页的协议(protocol)，一个就是host是否相同，即，就是url的首部 如： http: (protocol协议) www.abcd.com:8080 (host) 使用：document.domain ='' 1. 对于这种状况，ifreme在做跨域的时候，可以通过在父页面和iframe子页面同时设置document.domain = 'abcd.com'实现降域。子页面和父页面同时设置才会有效果，才会跨域通信，否则会出错，而且值要相同。这种方法跨域传输数据能够得到解决。 注意： 1.设置document.doamin，也会影响到其它跟iframe有关的功能。 典型的功能如：富文本编辑器（因为是iframe来做富文本编辑器的）、ajax的前进后退（因为IE67要用到iframe，参见：IE6与location.hash和Ajax历史记录）。 2.设置document.doamin，IE678下，有时获取location.href时有异常 document.domain ="" 方法只能解决，二级域相同，使用domain方法降域可以实现，如果完全不相同的域，此方法无效

//参考文档 http://blog.csdn.net/j\_y\_x\_8/article/details/50907366

### 手写代码 实现forEach map filter reduce

**forEach**

\_.each = \_.forEach = function(obj, iteratee, context) {

iteratee = optimizeCb(iteratee, context);

var i, length;

if (isArrayLike(obj)) {

for (i = 0, length = obj.length; i < length; i++) {

iteratee(obj[i], i, obj);

}

} else {

var keys = \_.keys(obj);

for (i = 0, length = keys.length; i < length; i++) {

iteratee(obj[keys[i]], keys[i], obj);

}

}

return obj;

};

**Map**

\_.map = \_.collect = function(obj, iteratee, context) {

iteratee = cb(iteratee, context);

var keys = !isArrayLike(obj) && \_.keys(obj),

length = (keys || obj).length,

results = Array(length);

for (var index = 0; index < length; index++) {

var currentKey = keys ? keys[index] : index;

results[index] = iteratee(obj[currentKey], currentKey, obj);

}

return results;

};

**Filter**

\_.filter = \_.select = function(obj, predicate, context) {

var results = [];

predicate = cb(predicate, context);

\_.each(obj, function(value, index, list) {

if (predicate(value, index, list)) results.push(value);

});

return results;

};

**Reduce**

function createReduce(dir) {

function iterator(obj, iteratee, memo, keys, index, length) {

for (; index >= 0 && index < length; index += dir) {

var currentKey = keys ? keys[index] : index;

memo = iteratee(memo, obj[currentKey], currentKey, obj);

}

return memo;

}

return function(obj, iteratee, memo, context) {

iteratee = optimizeCb(iteratee, context, 4);

var keys = !isArrayLike(obj) && \_.keys(obj),

length = (keys || obj).length,

index = dir > 0 ? 0 : length - 1;

if (arguments.length < 3) {

memo = obj[keys ? keys[index] : index];

index += dir;

}

return iterator(obj, iteratee, memo, keys, index, length);

};

}

\_.reduce = \_.foldl = \_.inject = createReduce(1);

\_.reduceRight = \_.foldr = createReduce(-1);

参考文档underscore.js

### commonJS AMD CMD

1. CommonJS是NodeJS的模块规范，由NodeJS实现
2. AMD是浏览器端的模块规范，由require.js实现
3. CMD也是浏览器端的模块规范，由seajs实现（国内的大神玉伯写的）
4. ESModule规范，ECMAScript6官方提出的模块规范，Chrome61以上的版本支持，否则要通过Webpack使用
5. UMD规范，一种写法，支持CommonJS、AMD、原生全局的写法

(**function** (root, factory) {

**if** (**typeof** define === 'function' && define.amd) {

*// AMD*

        define(['jquery'], factory);

    } **else** **if** (**typeof** exports === 'object') {

*// Node, CommonJS之类的*

        module.exports = factory(require('jquery'));

    } **else** {

*// 浏览器全局变量(root 即 window)*

        root.returnExports = factory(root.jQuery);

    }

}(**this**, **function** ($) {

*//    方法*

**function** myFunc(){};

*//    暴露公共方法*

**return** myFunc;

}));

1. SystemJS规范，Angular在用，需要了解请查阅资料<http://web.jobbole.com/83761/>

### jquery写的页面 切换到vue的页面 有卡顿吗

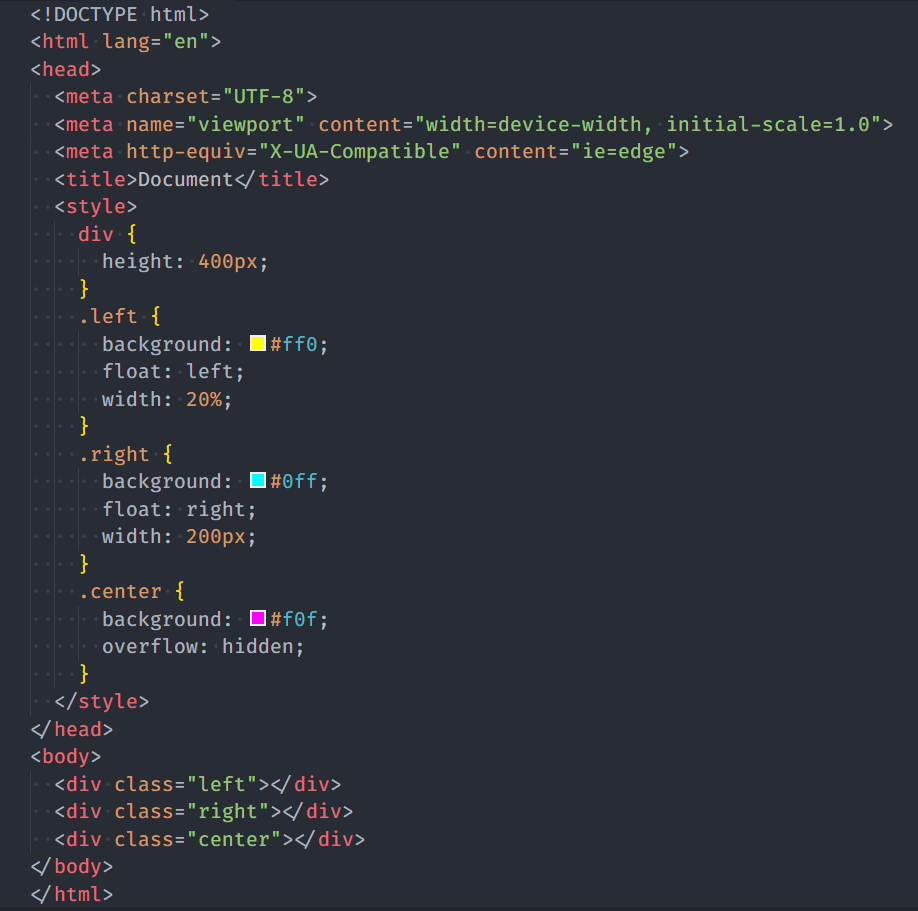
两个页面如果是独立实现的，两个html文件，不会卡顿

如果真的有，一半是jq写的，一半是Vue写的，那切换页面，首先要把jq对应的dom删掉，然后渲染vue对应的dom，可能会有性能问题，造成卡顿

1. **布局 左边20% 中间自适应 右边200px 不能用定位**

左右设置相应的宽度并分别左浮动右浮动，中间的块overflow: hidden;

div块的顺序是中间的放到最后



1. **webpack 打包后源码有读过吗 模块之间怎么依赖的**

<https://segmentfault.com/a/1190000006814420>

每个模块都会被赋予一个id（正常是从0开始的数字），模块之间通过内置的\_\_webpack\_require\_\_函数来模拟CommonJS，从而建立联系

1. **在公司有做技术分享吗 分享了什么**

开放问题，自己根据实际情况回答。

1. **Jquery 源码 为什么一个功能代码那么多 都做什么事**

哪个功能模块？代码多少行算多？为了兼容低版本的IE浏览器，为了统一各个浏览器下的表现，需要很多的hack代码来保证功能的正常运行

（可以适当向面试官询问更多的信息来回答）

1. **require.js 源码看过吗 怎么做到异步加载的**

看一下req.createNode方法

req.createNode = function(config, moduleName, url) {

var node = config.xhtml ?

document.createElementNS('http://www.w3.org/1999/xhtml', 'html:script') :

document.createElement('script');

node.type = config.scriptType || 'text/javascript';

node.charset = 'utf-8';

node.async = true; // 异步

return node;

};

当node加载完毕后会调用context.onScriptLoad，做了实例化context.Module对象，并暴露给registry供调用

onScriptLoad: function(evt) {

if (evt.type === 'load' ||

(readyRegExp.test(evt.currentTarget || evt.srcElement).readyState)) {

interactiveScript = null;

// getScriptData()找evt对应的script，提取data-requiremodule的mod名称

var data = getScriptData(evt);

context.completeLoad(data.id);

}

};

找到一篇好博客：<http://blog.csdn.net/zqjflash/article/details/43373957>

1. **关注前端消息有哪些渠道**

1. 优先关注各大技术的官网，阅读文档，关注官方GitHub的更新

2. 有个梗是知乎上有一半的程序员，知乎要定期刷，先关注前端相关的技术专栏，再关注相关的问题，关注回答者（大v，点赞数多），关注相关专栏

3. segementFault上看相关文章，同样是关注专栏，定期刷

4. 掘金、简书随便刷吧

5. 翻墙，follow一堆国外的公司Twitter，名人

6. 关注国内前端出名的团队博客：蚂蚁金服、腾讯前端团队、360奇舞团、美团、饿了么等等等

7. 关注质量高的博客等

1. **25.jenkins 上线流程**

Jenkins是基于Java开发的一种持续集成工具，用于监控持续重复的工作，功能包括：持续的软件版本发布/测试项目和监控外部调用执行的工作。

官网地址地址： [https://jenkins.io](https://link.jianshu.com/?t=https://jenkins.io)

**下载安装启动**

**CentOS 下用yum进行安装启动**

# 先更新源再安装最新版 jenkins

sudo wget -O /etc/yum.repos.d/jenkins.repo <https://pkg.jenkins.io/redhat-stable/jenkins.repo>

sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io.key>

sudo yum install jenkins

# 启动

sudo service jenkins start

# 停止

sudo service jenkins stop

# 重启

sudo service jenkins restart

# 检查

sudo chkconfig jenkins on

**下载war包放到tomcat中启动**

服务器 yum 安装速度太慢了，最终我选择了这种方式，本地下载好war包传到服务器上的tomcat容器下，然后启动

下载地址：[http://mirrors.jenkins.io/war-stable/latest/jenkins.war](https://link.jianshu.com/?t=http://mirrors.jenkins.io/war-stable/latest/jenkins.war)

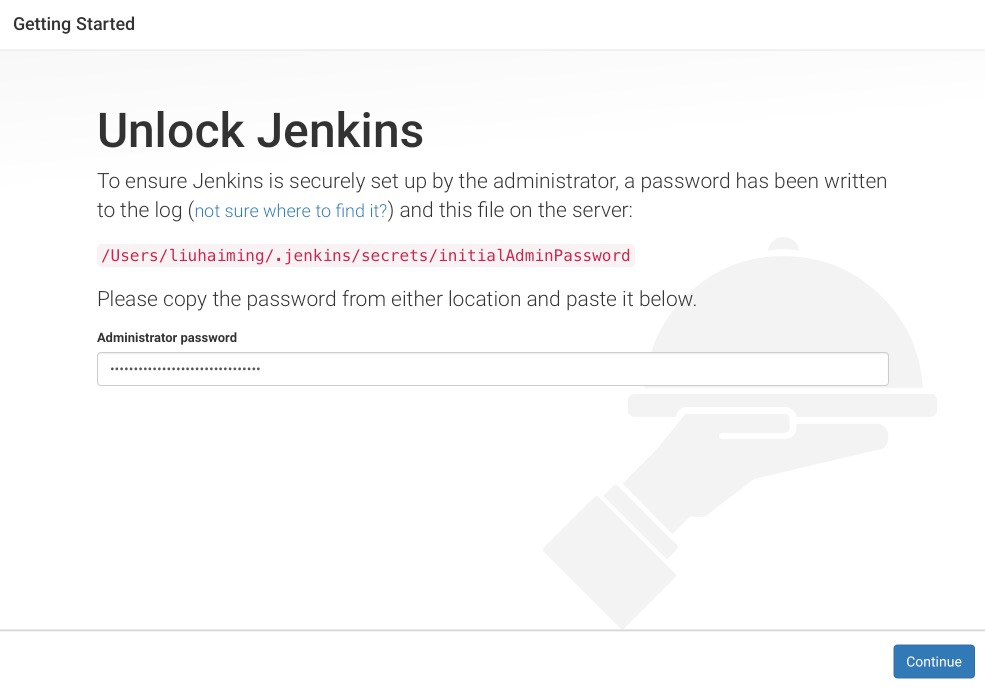
**初始化**

在浏览器中输入url打开jenkins的后台控制页面



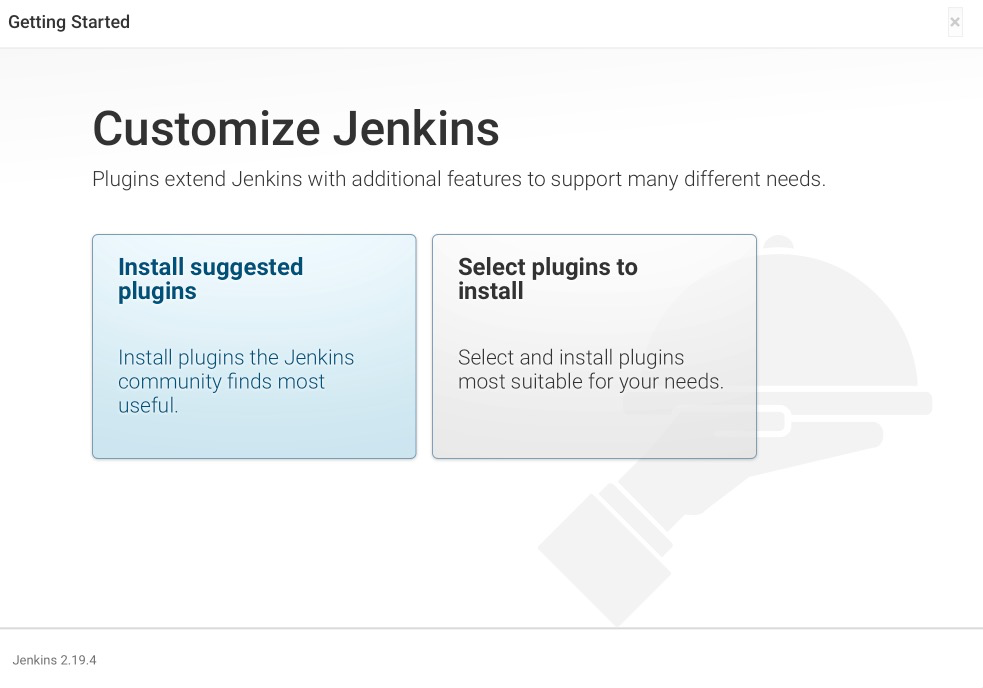
初始化界面

初始化成功后会自动生成一个管理员密码放到指定位置，根据页面提示复制密码粘贴到输入框就可以登录了



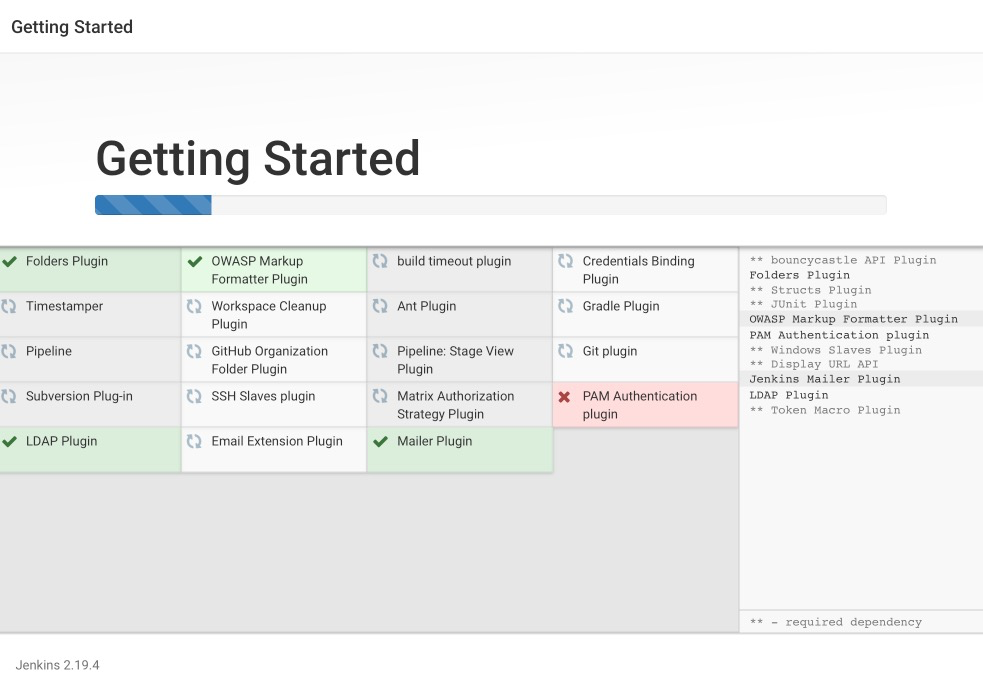
初始登录界面

登录成功后回让你选择插件的安装，可以选择建议的安装也可以自己进行选择，不清楚的话可以使用建议的安装



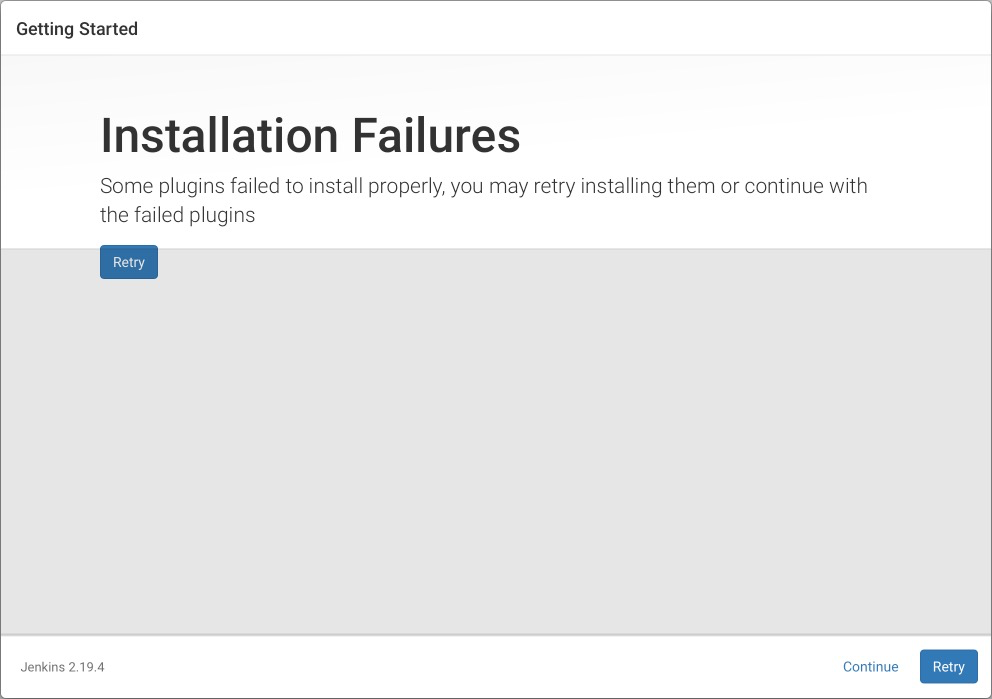
初始登录界面

由于建议安装的插件比较多，安装的过程有点慢，多等待一会



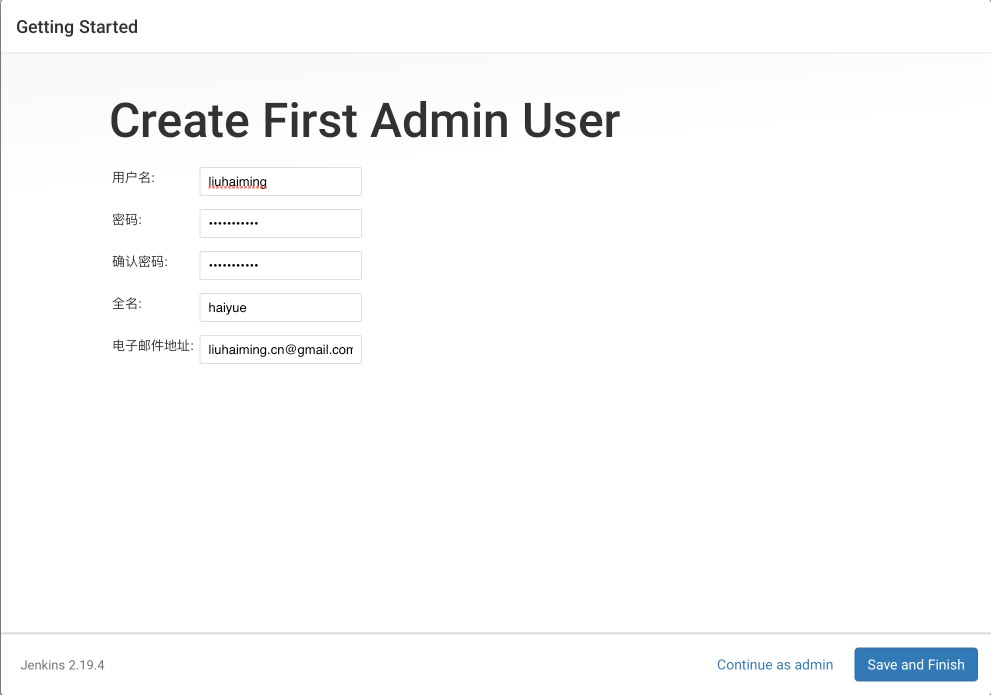
初始登录界面

安装的过程也可能因为网络等一些原因安装会失败，现在可以无视它，点击Continue，后面再进行手动的安装



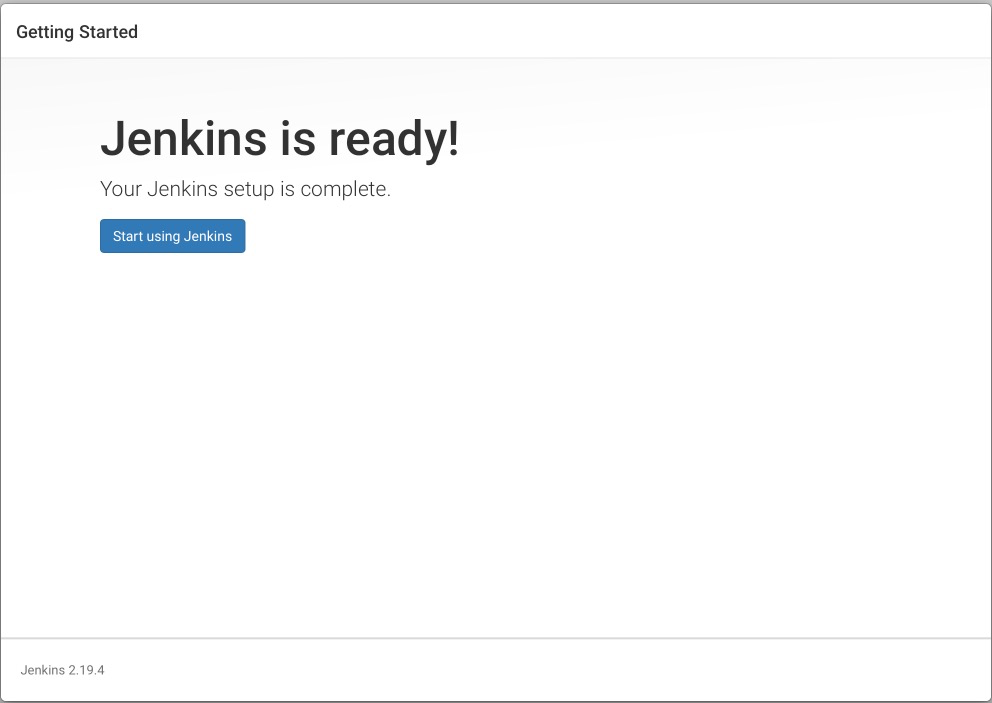
安装完成

安装完成后最好新创建一个管理员账户代替之前的临时自动生成的密码账户

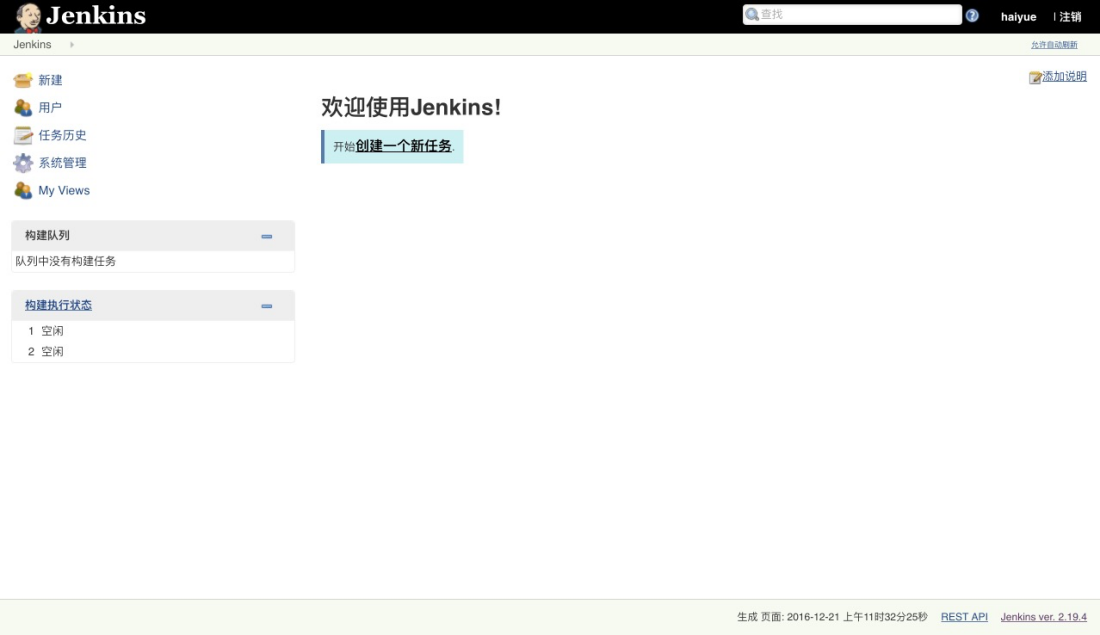


创建新的管理员账户

初始化完成，进入后台管理界面



初始化完成

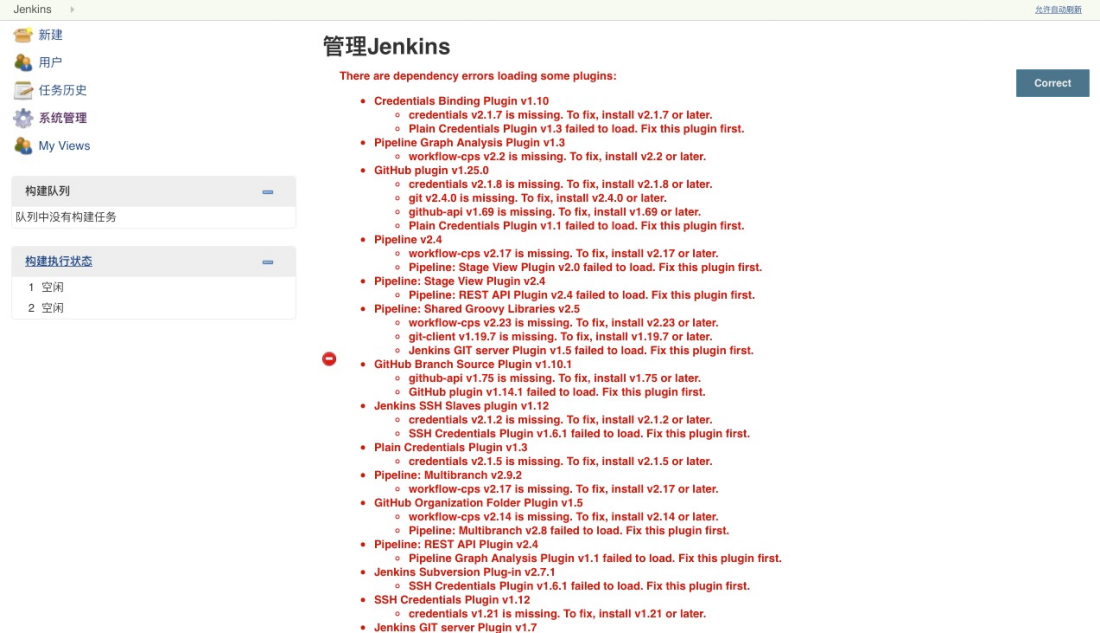


后台管理界面

**安装插件**

之前初始化的时候，有些插件安装失败，可以在用到的时候来手动修复它，没用到的话就可以暂且不理它，不影响jenkins的使用

点击左侧边栏的“系统管理”，就可以看了插件安装的一些错误信息



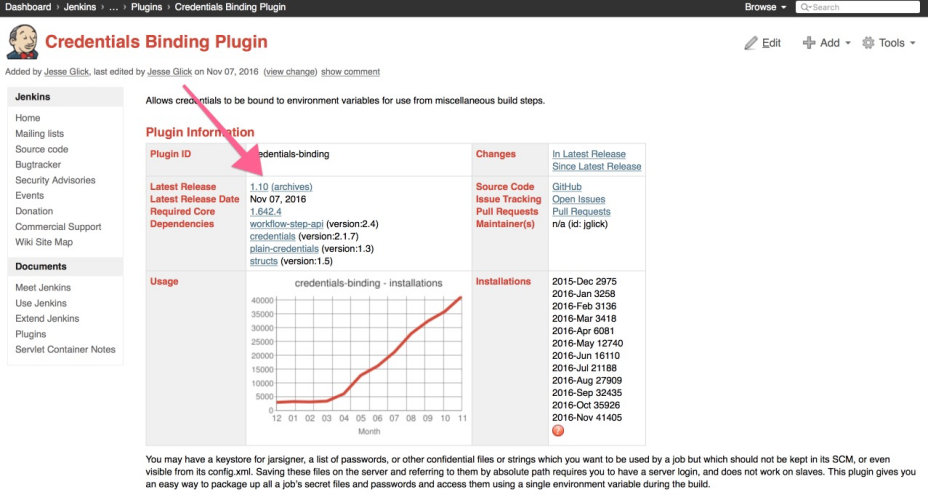
插件错误信息

在“系统管理”中往下拉,找到“管理插件”点击进去就可以查看和管理所有的插件，点击“可选插件”显示所有jenkins支持的插件，在右上角的“过滤”输入框中，输入需要安装的插件名就可以筛选查找到想要的插件



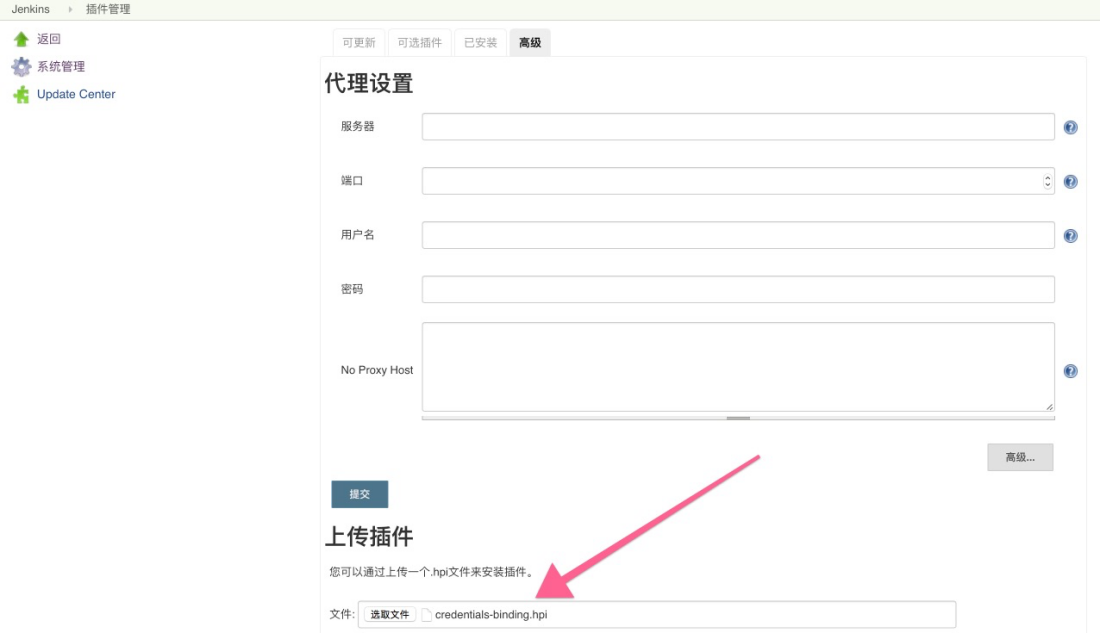
查找插件

选中要安装的插件输入框，点击安装就可以在线安装需要的插件，当然由于网络的原因也可能再次安装错误，或者安装的比较慢。我们可以点击插件名进入插件的主页，里面有该插件的详细信息并能下载hpi文件进行手动安装



插件详情

在“管理插件”的页面中点击高级选项，我们可以在下面找到“上传插件”，上传下载好的插件，点击“上传”，系统就会自动上传安装该插件。



上传插件

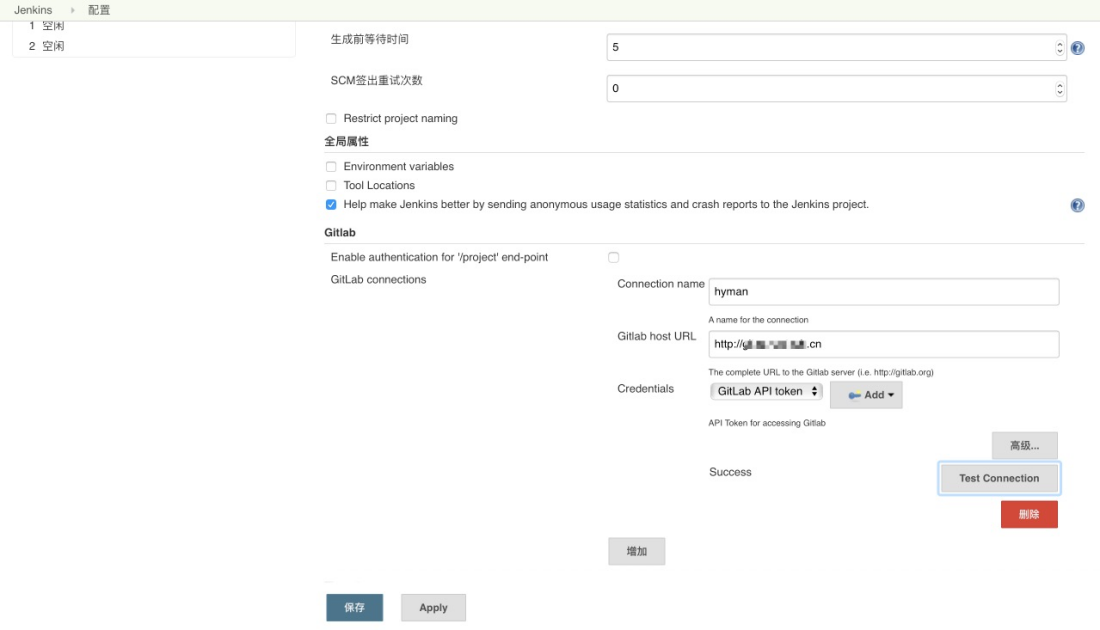
**gitlab的配置**

集成gitlab，让jenkins能够直接读取修改gitlab中的代码，方便项目的构建

安装gitlab-plugin

在“系统管理” -> “系统设置“ -> “Gitlab” 中配置对应的gitlab信息

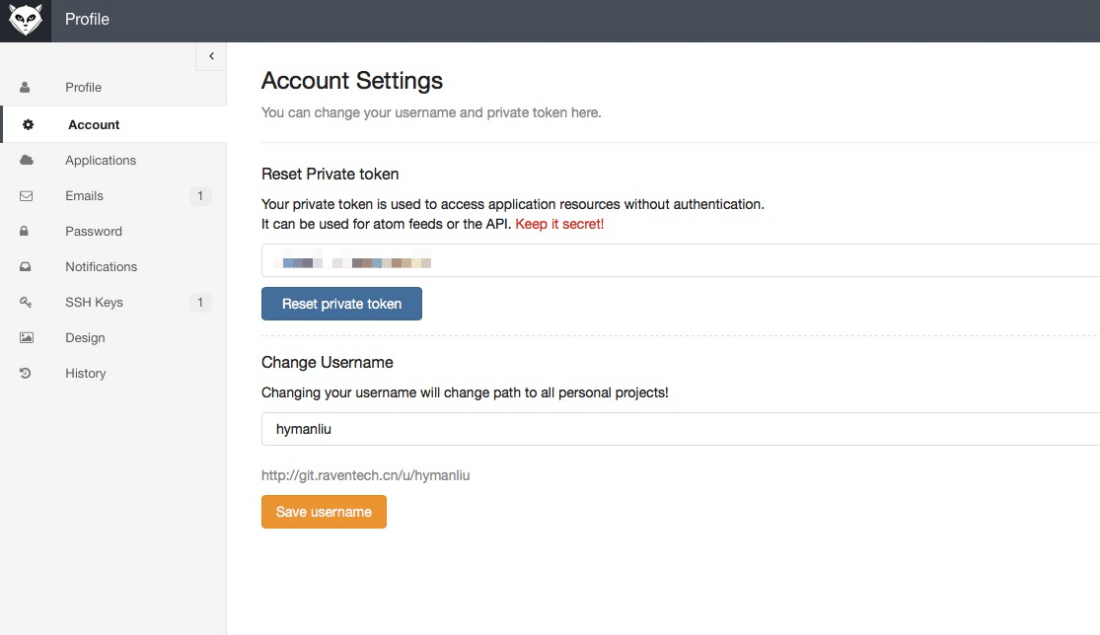
点击“Test Connection”测试下配置是否成功



Gitlab配置



Add Gitlab Credentials



Gitlab API token

**Publish Over SSH**

通过ssh连接远程服务器，并能执行脚本部署项目

安装publish-over-ssh

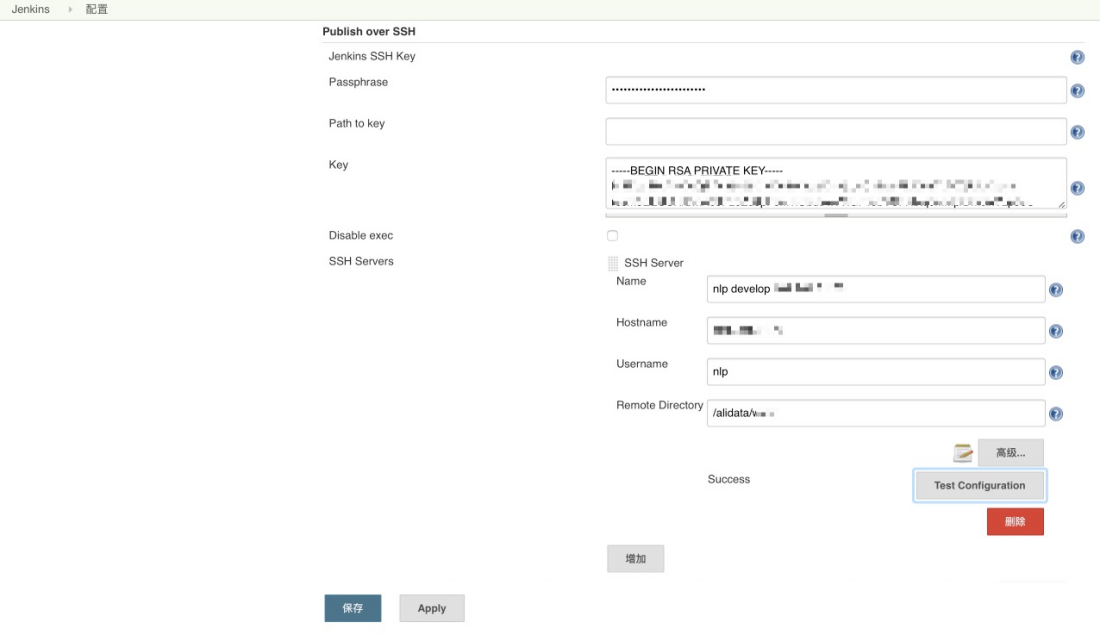
在“系统管理” -> “系统设置“ -> “Publish over SSH” 中配置对应的ssh信息

Key中填登录远程服务器的密码([ssh免密码登录](https://link.jianshu.com/?t=https://www.google.com.hk/search?client=safari&rls=en&q=ssh%E5%85%8D%E5%AF%86%E7%A0%81%E7%99%BB%E5%BD%95&ie=UTF-8&oe=UTF-8&gws_rd=cr,ssl))

点击"SSH Servers"后的“增加”按钮，新增一个远程服务器

点开“高级...”按钮，能进一步的配置端口等信息。

配置为Server信息后，点击"Test Configuration"按钮测试是否能够连接成功。



ssh配置

监测代码变动自动部署

点击左侧边栏的“新建”按钮，新建一个任务。

填写项目的名称，并选择一种构建的方式，此时我们选择第一个，构建一个自由风格的软件项目，然后点击“OK”按钮创建任务，并进行详细的配置



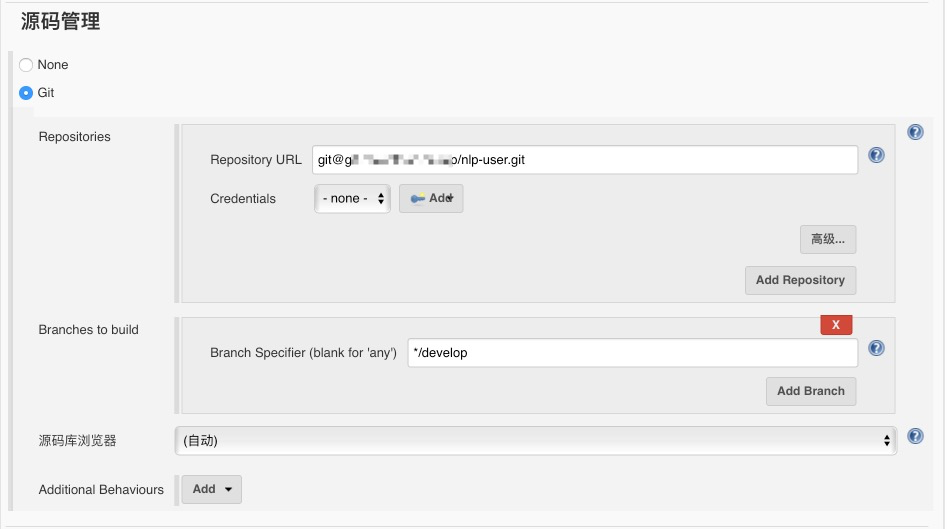
新建任务

默认设置里填写项目名和描述，并选择之前配置好的要连接的gitlab



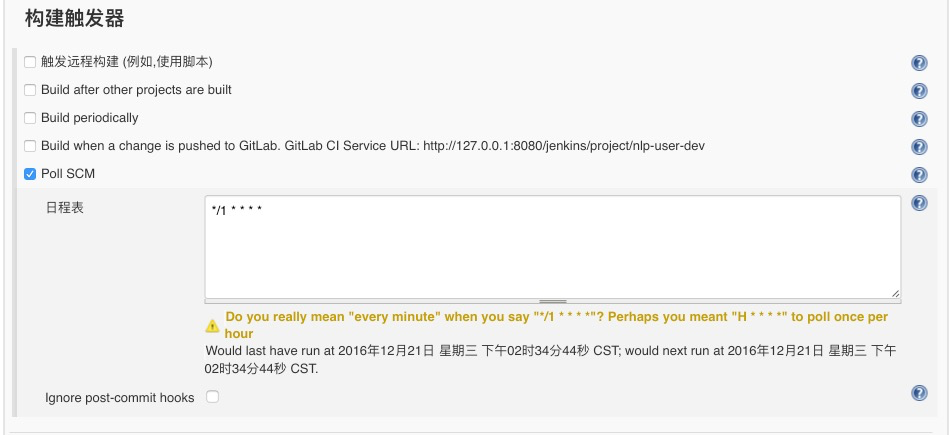
默认配置

配置源码，填写要构建项目的源码仓库地址，并指定要构建的分支



配置源码

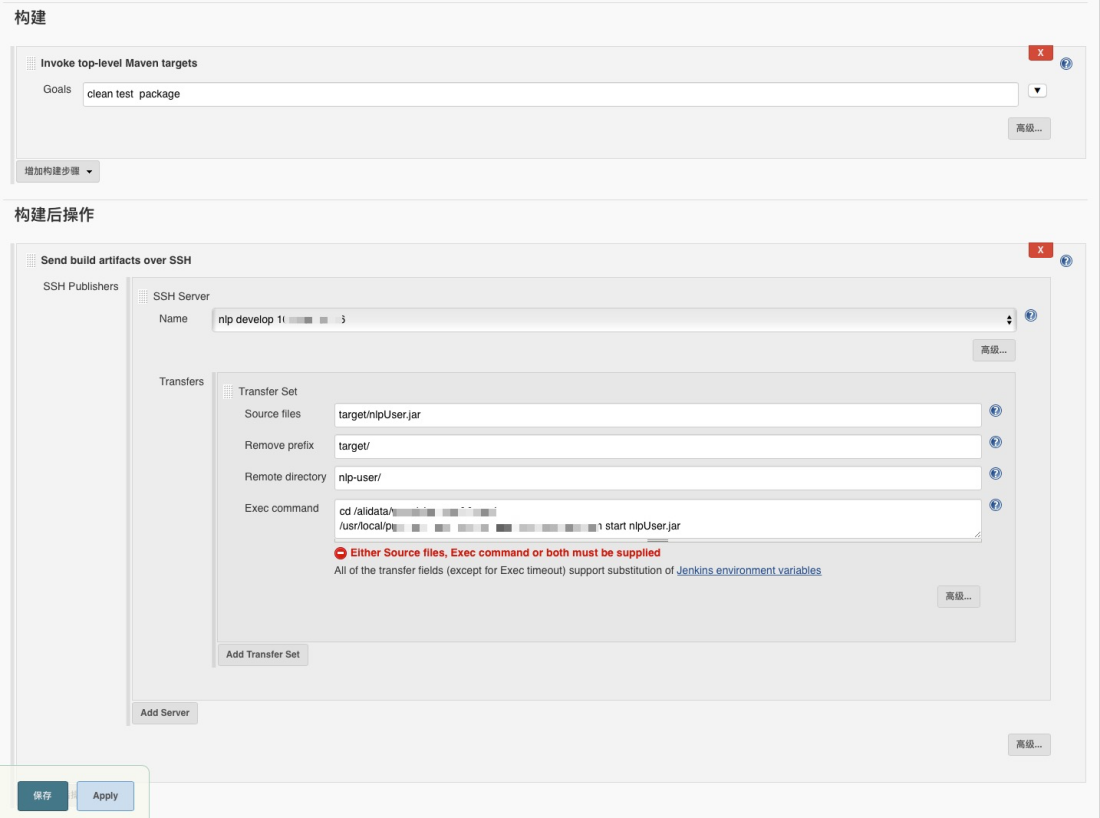
配置触发器，选择触发构建的方式，可以通过hook，根据jenkins提供的地址，放到gitlab中的hook配置中，就会自动触发构建。此时我们选择的是定时检测项目变动，如果检测到分支有新的变动就触发构建，如果感觉一分钟时间太频繁的话，可以自己设置时间频率。



构建触发器

配置构建，构建选用的是“Invoke top-level Maven target”,填写对应的maven命令，就会自动执行maven命令进行侯建

配置构建后操作， 该行为会在构建完成后执行，我们选用的是“Send build artifacts over SSH”的方式，把构建完成的jar包发送到远程服务器上用ssh命令执行启动，此时jenkins所有机器的默认路径是任务所在的目录，远程机器的默认路径是之前publish-over-ssh中指定的文件地址。Source files指定要传送到远程服务器上的文件，remote directory指定的是传送到远程服务器上的文件地址，Remove prefix是值要去除的文件目录，不然传送到远程服务器也会带有该目录层级结构的。exec Command里输入的是在远程服务器上要执行的指令。



构建部署配置

项目构建后会有构建历史，点击进去，选择“Console Output”就可以查看构建过程中的执行记录

**项目回滚**

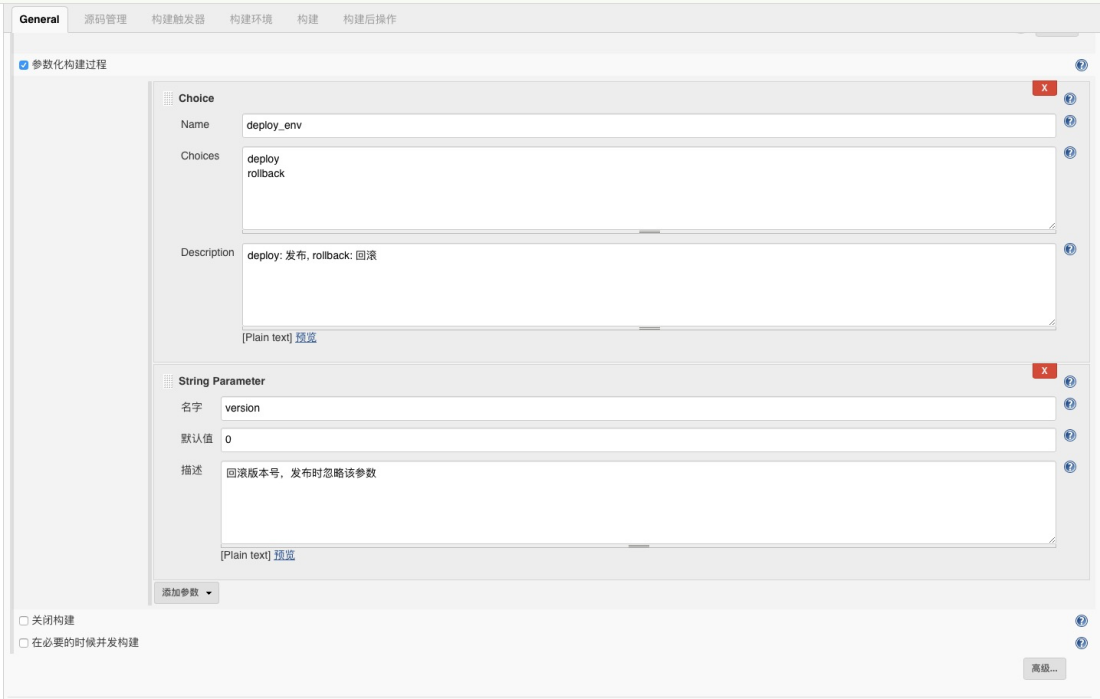
上面虽然实现了项目的自动部署，但是有时部署失败的时候我们需要回滚到指定版本的构建，这样才能更灵活的进行项目的构建部署。我们可以选择“参数化的构建过程”进行传递不同的参数来选择是进行新的构建还是回滚

如果要在实现回滚，一定要在构建后将，构建完成的文件进行存档，方便以后回滚的时候使用



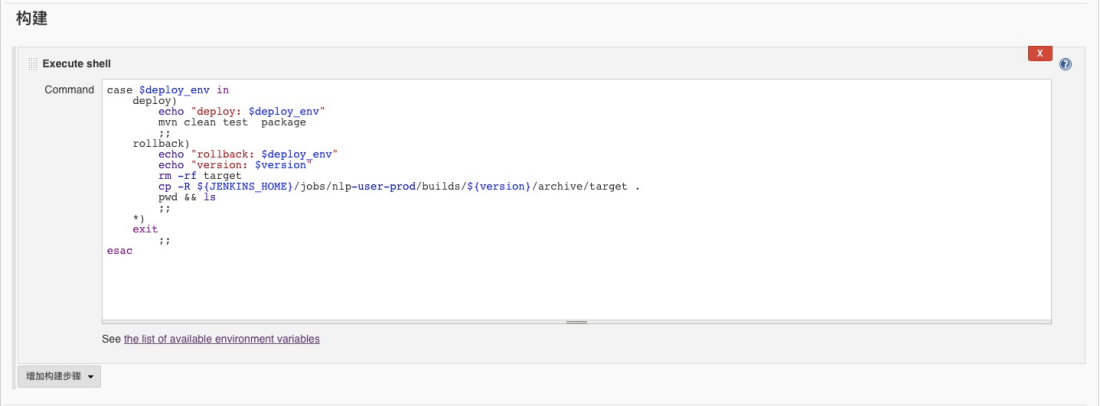
构建后存档

使用参数化构建过程，让后面的脚步可以根据不同的变量执行不同的操作。添加“Choice”参数配置不同的选项，让选择发布还是回滚，添加“String Parameter”参数来传递要回滚的版本号。



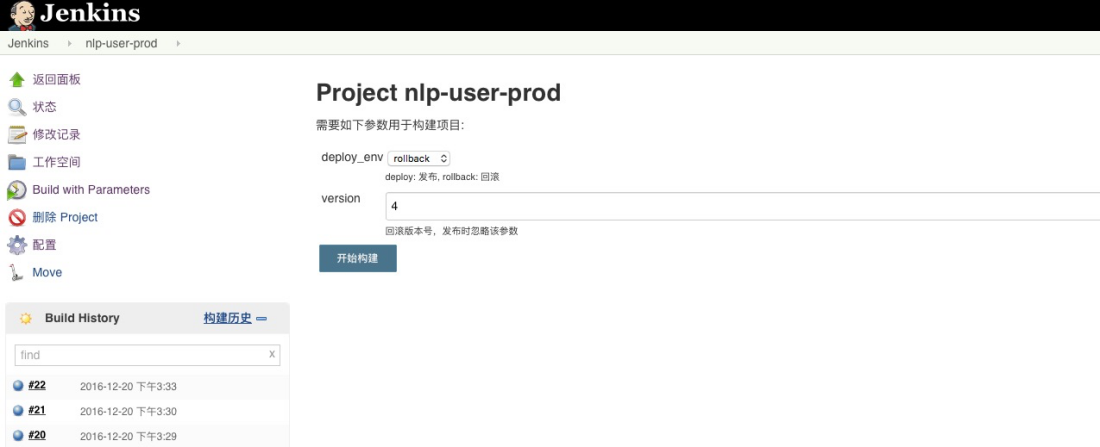
参数和构建过程

构建选择“Execute Shell”的方式，自己根据变量，自定义构建的脚本，此时如果是发布安装maven的构建过程进行新的构建，如果是回滚，知道历史构建后的文件，复制到当前构建结果目录。



构建脚本

点击构建，根据不同的参数选择发布还是回滚，回滚的时候填写要回滚到的历史版本号



构建页面

1. **setTimeOut 传参数**

setTimeout传递的第一个参数为：

1.无参的方法：如function test(){},则可直接通过setTimeout(test,1000)调用，还可以setTimeout("test()",1000)调用。

2.如果方法接收参数：

a) 如果都是字符串形式的参数，如：function test(param){} ，可使用字符串拼接传递参数:

setTimeout("test('" + myParam + "')",1000)。

b) 如果非字符串参数，这时可以通过使用function返回一个无参的函数进行调用，如：

function  test(obj)

{

    return function()

    {

      obj.style.color = "red";

    }

}

调用： setTimeout（test（this),1000);

如果你的方法test为：

function test(obj)

{

   obj.style.color="red";

}

此时是无法通过setTimeout（test（this),1000);调用的，会提示脚本出错。

1. **jq $()能传什么参数 ?html代码怎么解析?传function呢?**

根据jQuery的文档来看,$()能传入选择器,DOM元素,HTML和回调函数

1).传入选择器或DOM元素:

**描述:**接受一个包含一个CSS选择器的字符串，用于匹配的一组元素。

**jQuery( selector [, context ] )**

* **selector**  
  类型: [selector](https://www.jquery123.com/Types/" \l "selector)  
  一个包含CSS选择器的字符串
* **context**  
  类型: [Element, jQuery](https://www.jquery123.com/Types/" \l "Element, jQuery)  
  一个DOM 元素、文档或 作为上下文（context）的jQuery 对象。

**jQuery( element )**

* **element**  
  类型: [Element](https://www.jquery123.com/Types/" \l "Element)  
  一个用于封装成jQuery对象的DOM元素。

**jQuery( elementArray )**

* **elementArray**  
  类型: [Array](https://www.jquery123.com/Types/" \l "Array)  
  一个用于封装成jQuery对象的DOM元素数组。

**jQuery( object )**

* **object**  
  类型: [PlainObject](https://www.jquery123.com/Types/" \l "PlainObject)  
  一个普通的对象包装在一个jQuery对象。

**jQuery( jQuery object )**

* **jQuery object**  
  类型: [PlainObject](https://www.jquery123.com/Types/" \l "PlainObject)  
  一个用于克隆的jQuery对象。

**jQuery()**

* 这个方法不接受任何参数。

2).传入HTML

**描述:**根据提供的原始 HTML 标记字符串，动态创建由 jQuery 对象包装的 DOM 元素。

**jQuery( html [, ownerDocument ] )**

* **html**  
  类型: [String](https://www.jquery123.com/Types/" \l "String)  
  用于动态创建DOM元素的HTML标记字符串，**不是**XML。
* **ownerDocument**  
  类型: [document](https://www.jquery123.com/Types/" \l "document)  
  一个文档的新元素将被创建。

**jQuery( html, attributes )**

* **html**  
  类型: [String](https://www.jquery123.com/Types/" \l "String)  
  一个单标记的HTML 元素字符串(例如 <div/> or <div></div>).
* **attributes**  
  类型: [PlainObject](https://www.jquery123.com/Types/" \l "PlainObject)  
  访问新创建元素的属性，事件和方法。

3).传入方法

**描述:**当DOM完成加载的时候绑定一个要执行的函数。

**jQuery( callback )**

* **callback**  
  类型: [Function](https://www.jquery123.com/Types/" \l "Function)()  
  当DOM ready的时候执行的函数。

1. **AMD怎么加载文件的?**

原文地址:

<https://www.cnblogs.com/axl234/p/5295307.html>

AMD全称(Asynchronous Module Definition)，异步模块加载。

AMD的模块依赖策略是提前执行,代码风格是依赖前置.

提前执行是指假设依赖列表里面存在依赖['a', 'b']，谁先加载完毕，谁先实例化，实例化的顺序因为异步的关系所以是乱序，当然，执行的时候，还是会保证顺序的关系。

而依赖前置则是指提前告诉加载器模块依赖的情况.

define(['dep1','dep2'],function(dep1,dep2){

//内部只能使用制定的模块

    return function(){};

});

而基于AMD规范实现的requirejs则是使用创建script元素,通过指定script元素的src属性来实现加载模块的.

requireJS的load方法的主要代码如下:

req.load = function (context, moduleName, url) {

var config = (context && context.config) || {},

node;

if (isBrowser) {

//create a async script element

node = req.createNode(config, moduleName, url);

//add Events [onreadystatechange,load,error]

.....

//set url for loading

node.src = url;

//insert script element to head and start load

currentlyAddingScript = node;

if (baseElement) {

head.insertBefore(node, baseElement);

} else {

head.appendChild(node);

}

currentlyAddingScript = null;

return node;

} else if (isWebWorker) {

.........

}

};

req.createNode = function (config, moduleName, url) {

var node = config.xhtml ?

document.createElementNS('<http://www.w3.org/1999/xhtml>', 'html:script') :

document.createElement('script');

node.type = config.scriptType || 'text/javascript';

node.charset = 'utf-8';

node.async = true;

return node;

};

rquire使用的是script标签去拿js，node上设定了 async 属性（异步加载script标签），并且在标签上绑定了load等事件，当文件loading完成后，则要做的主要工作是执行 completeLoad 事件函数，但是要注意的是这时候把script加载完成后，立即执行的是script标签内部的内容，执行完后才触发的 completeLoad事件.

1. **webpack如何加载资源的?**

webpack的核心概念:

* **entry** 一个可执行模块或库的入口文件。
* **chunk** 多个文件组成的一个代码块，例如把一个可执行模块和它所有依赖的模块组合和一个 chunk 这体现了webpack的打包机制。
* **loader** 文件转换器，例如把es6转换为es5，scss转换为css。
* **plugin** 插件，用于扩展webpack的功能，在webpack构建生命周期的节点上加入扩展hook为webpack加入功能。

从启动webpack构建到输出结果经历了一系列过程，它们是：

1. 解析webpack配置参数，合并从shell传入和**webpack.config.js**文件里配置的参数，生产最后的配置结果。
2. 注册所有配置的插件，好让插件监听webpack构建生命周期的事件节点，以做出对应的反应。
3. 从配置的**entry**入口文件开始解析文件构建AST语法树，找出每个文件所依赖的文件，递归下去。
4. 在解析文件递归的过程中根据文件类型和loader配置找出合适的loader用来对文件进行转换。
5. 递归完后得到每个文件的最终结果，根据**entry**配置生成代码块**chunk**。
6. 输出所有**chunk**到文件系统。

需要注意的是，在构建生命周期中有一系列插件在合适的时机做了合适的事情，比如UglifyJsPlugin会在loader转换递归完后对结果再使用UglifyJs压缩覆盖之前的结果。

1. **一像素线的方案**

移动端window对象有个devicePixelRatio属性, 它表示设备物理像素和css像素的比例, 在retina屏的iphone手机上, 这个值为2或3, css里写的1px长度映射到物理像素上就有2px或3px那么长.

1px解决方案:

1).用小数来写px值

IOS8下已经支持带小数的px值, media query对应devicePixelRatio有个查询值-webkit-min-device-pixel-ratio, css可以写成这样:

.border { border: 1px solid #999 }

@media screen and (-webkit-min-device-pixel-ratio: 2) {

    .border { border: 0.5px solid #999 }

}

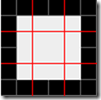
@media screen and (-webkit-min-device-pixel-ratio: 3) {

    .border { border: 0.333333px solid #999 }

}

缺点: 安卓与低版本IOS不适用, 这个或许是未来的标准写法, 现在不做指望

**2).border-image**



这样的1张6X6的图片, 9宫格等分填充border-image, 这样元素的4个边框宽度都只有1px:

@media screen and (-webkit-min-device-pixel-ratio: 2){

    .border{

        border: 1px solid transparent;

**border-image:** **url(border.gif) 2 repeat;**

    }

}

缺点: 对于圆角样式, 将图片放大修改成圆角也能满足需求, 但这样无形中增加了border的宽度

存在多种边框颜色或者更改的时候麻烦

**3).background渐变**

背景渐变, 渐变在透明色和边框色中间分割, frozenUI用的就是这种方法, 借用它的上边框写法:

@media screen and (-webkit-min-device-pixel-ratio: 2){

    .ui-border-t {

        background-position: left top;

**background-image: -webkit-gradient(linear,left bottom,left top,color-stop(0.5,transparent),color-stop(0.5,#e0e0e0),to(#e0e0e0));**

    }

}

缺点: 代码量大, 而且需要针对不同边框结构, frozenUI就定义9种基本样式.

而且这只是背景, 这样做出来的边框实际是在原本的border空间内部的, 如果元素背景色有变化的样式, 边框线也会消失.

最后不能适应圆角样式

**4).::before,::after与transform**

构建1个伪元素, 将它的长宽放大到2倍, 边框宽度设置为1px, 再以transform缩放到50%.

.radius-border{

    position: relative;

}

@media screen and (-webkit-min-device-pixel-ratio: 2){

    .radius-border:before{

        content: "";

        pointer-events: none; /\* 防止点击触发 \*/

        box-sizing: border-box;

        position: absolute;

        width: 200%;

        height: 200%;

        left: 0;

        top: 0;

        border-radius: 8px;

        border:1px solid #999;

        -webkit-transform(scale(0.5));

        -webkit-transform-origin: 0 0;

        transform(scale(0.5));

        transform-origin: 0 0;

    }

}

需要注意**<input type="button">是没有:before, :after伪元素的**

优点: 其实不止是圆角, 其他的边框也可以这样做出来

缺点: 代码量也很大, 占据了伪元素, 容易引起冲突

1. **webpack中的ESlint怎么配置**

1).安装

需要这几个npm包:

* eslint
* eslint-loader
* eslint-plugin-html (用以lint一些在html文件里面通过script包裹的js代码,它默认的匹配规则是不带type属性，或者是`/^(application|text)/(x-)?(javascript|babel|ecmascript-6)$/i`,具体的内容请查阅相关文档,通过cli启动lint的时候定义文件后缀名时eslint --ext .html,.js)
* eslint-config-standard (和javascript-style-standard风格指南需要的包)
* eslint-plugin-promise
* eslint-plugin-standard
* eslint-friendly-formatter (生成的报告格式)

2).配置ESlint

关于eslint的配置方式。比较多元化：

* js注释
* .eslintrc.\*文件
* package.json里面配置eslintConfig字段

可以使用任何一个配置来配置ESlint,对于ESlint如何配置,这个里就不赘述了.

3).配置webpack

在webpack中的js模块处进行eslint-loader的配置,示例代码如下所示:

module: {

rules: [

{

test: /\.js$/,

exclude: /node\_modules/, // exclude: /node\_modules/ 可以不用定义这个字段的属性值，eslint会自动忽略node\_modules

enforce: 'pre', // 在babel-loader对源码进行编译前进行lint的检查

include: /src/, // src文件夹下的文件需要被lint

use: [{

loader:'babel-loader'

},{

loader: 'eslint-loader',

options: {

formatter: require('eslint-friendly-formatter') // 编译后错误报告格式

}

}],

query: {

cacheDirectory: true

}

}

]

}

4).配置package.json

在package.json中配置npm命令:

{

        ...

        "lint": "eslint --ext .js src"

        ...

}

然后执行npm run lint 来进行代码的静态检查

1. **postmessage**

介绍:

window.postMessage() 方法可以安全地实现跨源通信。通常，对于两个不同页面的脚本，只有当执行它们的页面位于具有相同的协议（通常为https），端口号（443为https的默认值），以及主机  (两个页面的模数 Document.domain设置为相同的值) 时，这两个脚本才能相互通信。window.postMessage() 方法提供了一种受控机制来规避此限制，只要正确的使用，这种方法就很安全。

window.postMessage() 方法被调用时，会在所有页面脚本执行完毕之后（e.g., 在该方法之后设置的事件、之前设置的timeout 事件,etc.）向目标窗口派发一个  MessageEvent 消息。 该MessageEvent消息有四个属性需要注意： message 属性表示该message 的类型； data 属性为 window.postMessage 的第一个参数；origin 属性表示调用window.postMessage() 方法时调用页面的当前状态； source 属性记录调用 window.postMessage() 方法的窗口信息。

语法:

**otherWindow.postMessage(message, targetOrigin, [transfer]);**

参数介绍:

**otherWindow**

其他窗口的一个引用，比如iframe的contentWindow属性、执行window.open返回的窗口对象、或者是命名过或数值索引的window.frames。

**message**

将要发送到其他 window的数据。它将会被结构化克隆算法序列化。这意味着你可以不受什么限制的将数据对象安全的传送给目标窗口而**无需自己序列化**。

**targetOrigin**

通过窗口的origin属性来指定哪些窗口能接收到消息事件，其值可以是字符串"\*"（表示无限制）或者一个URI。在发送消息的时候，如果目标窗口的协议、主机地址或端口这三者的任意一项不匹配targetOrigin提供的值，那么消息就不会被发送；只有三者完全匹配，消息才会被发送。这个机制用来控制消息可以发送到哪些窗口；例如，当用postMessage传送密码时，这个参数就显得尤为重要，必须保证它的值与这条包含密码的信息的预期接受者的orign属性完全一致，来防止密码被恶意的第三方截获。**如果你明确的知道消息应该发送到哪个窗口，那么请始终提供一个有确切值的targetOrigin，而不是\*。不提供确切的目标将导致数据泄露到任何对数据感兴趣的恶意站点。**

**transfer (可选)**

是一串和message 同时传递的 Transferable 对象. 这些对象的所有权将被转移给消息的接收方，而发送一方将不再保有所有权。

事件监听:

接受窗口可以通过监听message事件来接收传入的参数.

window.addEventListener("message", receiveMessage, false);

function receiveMessage(event)

{

// For Chrome, the origin property is in the event.originalEvent

// object.

var origin = event.origin || event.originalEvent.origin;

if (origin !== "[http://example.org:8080](http://example.org:8080/)")

return;

// ...

}

可以通过event.data来获取从其他window中传过来的对象

而event.origin则是指调用postMessage的发送方窗口的origin值

event.source则是指对发送消息的窗口对象的引用; 您可以使用此来在具有不同origin的两个窗口之间建立双向通信。

1. **JSBridage 原理****x3 android ios 什么区别**

JSBridge是Hybrid应用开发过程中原生提供给js可调用系统功能模块（相册、照相机等）的接口

参考链接：http://coolnuanfeng.github.io/jsbridge

1. **react 按需加载**

我的理解：

1. react调用类似ant design等UI库时按需加载组件

以ant design为例：

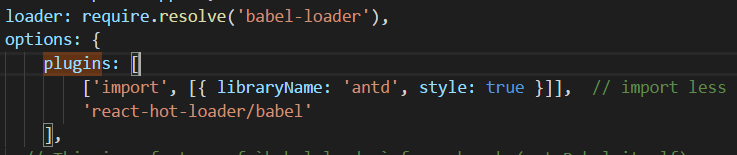
官方推荐安装 babel-plugin-import

配置 .babelrc文件



或

在webapck中配置babel-loader时在options字段中配置



具体可参考ant design官方文档

1. react路由懒加载，结合webpack代码分割&&组件异步加载

当webpack打包后文件过大除了考虑通过commonsChunkPlugin抽离公共部分，我们可以利用webpack的代码分割功能，将各个组件打包进单独的js文件，只有当需要该组件时才去下载对应的js文件加载该组件，来提高首屏加载速度

解决方案：

1. 配置webpack

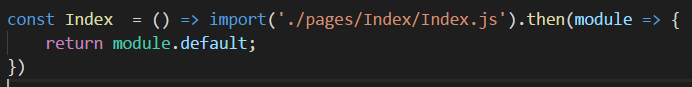
在output中添加 chunkFilename: '[name].chunk.js'

该字段决定非入口 chunk 的名称

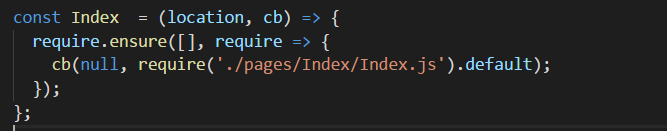
1. 组件异步加载&&路由懒加载

导入异步组件有两种方案

使用符合 ECMAScript 提案 的 import() 语法 （推荐）



使用 webpack 特定的 require.ensure



当模块通过export default导出时，需要加上 .default



将component 改为 getComponent实现路由懒加载

1. **jquery怎么找到 事件源元素**

e.target

1. **模板引擎原理**

模板引擎本质上为一处理字符串的函数，通过我们预定的规则书写类html结构代码

如： <ul>

<% for(var i=0; i<list.length; i++){ %>

<li><%=list[i].value%></li>

<% } %>

</ul>

通过正则处理该段字符串为符合JavaScript代码规范的字符串

log(‘<ul>’);

for(var i=0; i<list.length; i++){

log(‘<li>’);

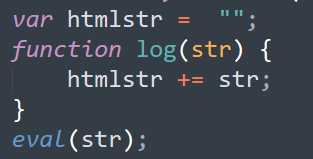
log(list[i].value);

log(‘</li>’);

}

Log(‘</ul>’)

事先定义log方法，然后通过eval方法执行该字符串，拼接得到我们最终需要的完全符合html语法规范的字符串 htmlstr，并插入到页面中



1. **灰度测试时什么**

灰度测试又称灰度发布，及当进行软件升级更新或添加新功能时，要先进行一个小范围的尝试工作，然后再慢慢放量，直到这个全新的功能覆盖到所有的用户，可以帮助研究团队快速试验并发现问题并在大规模推向用户之前及时把问题修正过来，很大成度上减少了不少风险的产生

1. **x5浏览器哪些 特点 哪些坑**

参考链接：

<https://www.qianduan.net/qqliu-lan-qi-x5nei-he-wen-ti-hui-zong/>

http://www.wxdroid.com/index.php/2787.html

1. **border-box 取值 各个值用法**

我暂理解为 box-sizing的取值及各个值的用法

box-sizing共三个属性

border-box: content-box (默认值)

border-box

padding-box(已废弃)

区别

content-box盒子模型width = content

border-box盒子模型 width = content + padding + border

**Promise .then .then 怎么终止**

我们通过new promise创建的promise对象允许我们传入一个参数(函数),包含两个形参(resolve,reject)，我们可以在这个函数里写自己的异步操作通过resolve()和reject()来决定向下执行的方向then方法接受两个参数(resolve之后的回调,reject之后的回调),回调接受一个参数也就是从上一个promise传递过来的结果(如primise执行resolve(data)中的data)，然后我们根据结果进行处理，如果要进行链式调用则在resolve回调函数中返回一个promise对象。如果中间遇到reject则不会执行后续的then。也可以通过new promise().then().catch()的方式处理resolve和reject结果道理和上述相同。

https://developer.mozilla.org/zh-

CN/docs/Web/JavaScript/Reference/Global\_Objects/Promise/then

**chorme 浏览器 sort 排序原理**

sort排序允许接受一个参数(函数),默认情况下根据字符编码进行排序，这个函数允许接受两个形参(a,b当然你写item1,item2也没问题)，a,b分别代表这一次行为的两个数组项，返回值表示这两个数的相对顺序=>(a<b则return 小于0,a==b则return 等于0,a>b则return 大于0)

**\_\_proto\_\_能直接绑定属性吗**

可以，但是有个问题如果a是b的实例通过a.\_\_proto\_\_去绑定属性那么同时也会在a.prototype上绑定这个属性，也就是说以后的每个实例都可以访问这个绑定的属性

例如

var a = function(name,age){  
 this.name = name

this.age = age;

}

var b = new a(“aaa”,123)

console.log(b) //{name : “aaa”,age : 123}

b.\_\_proto\_\_.class = ‘1111’;

console.log(b.class) //1111

console.log(b.\_\_proto\_\_.class) //1111

console.log(a.prototype.class) //1111

**chorme浏览器 IE浏览器 事件兼容和DOM0 DOM2 事件写法区别**

DOM0级方式：  
添加事件：button.onclick = function(){};  
删除事件：button.onclick = null;  
DOM2级方式：  
添加事件：  
非IE下：button.addEventListener(‘onclick ’,function(){},false)//false代表不冒泡  
IE下：button.attachEvent(‘onclick ’,function(){});//没有第三个参数，是因为IE只支持冒泡  
删除事件：  
非IE下：button.removeEventListener(‘onclick ’,function(){},false);  
IE下：button.detachEvent(‘onclick ’,function(){});

**手写 实现bind 实现new 实现函数节流 使用setTimeout实现setInterval**

实现bind：

Function.prototype.testBind = function(that){

var \_this = this；

slice = Array.prototype.slice,

args = slice.apply(arguments,[1]);

return function(){

return \_this.apply(that,args)

}

}

更多查看：https://segmentfault.com/a/1190000007342882

实现new:

function New(f) {

return function () {

var o = {"\_\_proto\_\_": f.prototype};

f.apply(o, arguments);//继承父类的属性

return o; //返回一个Object

}

}

更多查看：https://www.cnblogs.com/zhangxiaoshu/p/6515971.html

函数节流：

function throttle(method,delay){

var timer=null;

return function(){

var context=this, args=arguments;

clearTimeout(timer);

timer=setTimeout(function(){

method.apply(context,args);

},delay);

}

}

更多查看：<https://www.cnblogs.com/dolphinX/p/3403821.html>

setTimeout实现setInterval

setTimeout(function(){

//do something

setTimeout(arguments.callee,interval);

},interval)

更多查看: http://blog.csdn.net/baidu\_24024601/article/details/51862488

1. **检测数组改变 并计算平均值**

<http://www.51xuediannao.com/javascript/javascriptjtszbh_1258.html>

for(var i=0;i<3;i++){

setTimeout(function(){console.log(i--)},0)

}

//打印输出什么

setTimeout执行时插入任务队列，主线执行完后依次执行任务队列里的任务

所以结果是3,2,1

1. **深入理解原型链 一直追溯到root 才是完整的原型 js最核心的知识点**

https://segmentfault.com/a/1190000008959943

1. **尝试评估自己的2年工作经验的 代码量（JS代码）**

看情况，维护期一天一般有效代码一般就20-40行，忙的时候一天200，300行

1. **1W行代码的项目能维护吗？5W ？ 10W? 50W? 100W? （要认识到自己的能力上限）**

项目文档质量   
项目代码质量

如果项目文档质量好，项目结构健全，代码注释完善

可以快速定位bug位置，都是可以维护的

1. **react 实现目录树 （组件自身调用自身）**

自身调用自身，直接在render里面使用自己就可以了，跟递归一样

加一个结束递归的条件

import React from 'react'

class Tree extends React.Component {

render() {

if(false)

return (<Tree></Tree>)

else

return null

}

}

export default Tree

1. **手写一个简单的组件**

**使用React做一个简单的to-do-list <https://www.cnblogs.com/wangfupeng1988/p/5302738.html>**

1. **手写代码 Jquery 实现动画原理**

<http://blog.csdn.net/vbdfforever/article/details/51121012>

1. **浏览器地址栏里面的“ # ”如何消除？**

通过location.href，用正则把#替换掉

1. **Vue中mode 共有几个参数？各参数有什么区别？**

Vue中mode有三个参数

● 类型: string

● 默认值: "hash" (浏览器环境) | "abstract" (Node.js 环境)

● 可选值: "hash" | "history" | "abstract"

配置路由模式:

○ hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History Api 的浏览器。

○ history: 依赖 HTML5 History API 和服务器配置。

○ abstract: 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现没有浏览器的 API，路由会自动强制进入这个模式。

1. **Vue中父组件如何向子组件传值**

主要通过props属性实现 另外：vuex、bus总线

1. **列举 Vue 中的事件修饰符**

Vue.js 为 v-on 提供了事件修饰符。修饰符是由点开头的指令后缀来表示的。

.stop 阻止事件继续传播

.prevent 阻止默认事件

.capture 使用捕获模式

.self 只当事件在该元素本身（而不是子元素）触发时触发回调

.once 事件只会触发一次

1. **如何合并两个对象**

传统实现：

var obj1 = {

'a': 1,

'b': 2

};

var obj2 = {

'c': 3

};

var extend = function (obj1, obj2) {

for (var p in obj2) {

if (obj2.hasOwnProperty(p) && (!obj1.hasOwnProperty(p)))

obj1[p] = obj2[p];

}

return obj1;

};

ES6：

Object.assign()

JQuery：

$.extend( [deep ], target, object1 [, objectN ] )

1. **请分别解释 filter() , map() , every() , some() , join() 的作用以及区别**

filter():创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素

map():返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

every():用于检测数组所有元素是否都符合指定条件

● 如果数组中检测到有一个元素不满足，则整个表达式返回 false ，且剩余的元素不会再进行检测。

● 如果所有元素都满足条件，则返回 true。

some():用于检测数组中的元素是否满足指定条件。

join():用于把数组中的所有元素转换一个字符串。

1. **JQuery 中，$.data() 和 $("#abc").data() 各自是什么作用，有什么区别？**

两者的作用都是向被选元素附加数据，或者从被选元素获取数据

$.data(ele,[key],[val])此方法为底层方法，在jQuery 1.8中删除,不推荐

$("#abc").data([key],[value])更方便，推荐

1. **请编写一个动画，实现一个正方形向左移动100px，并且放大一倍，旋转45度**

<body>

<div id="box"></div>

</body>

<style>

#box {

position: relative;

left: 500px;

top:100px;

height: 100px;

width: 100px;

background: lightgreen;

transition: transform;

animation: tests 3s infinite;

}

@keyframes tests {

from {

left: 500px;

}

to {

left: 400px;

transform: scale(2)rotate(45deg)

}

}

</style>

1. **p>a , p~a , p+a 有什么区别？**

p>a : 子选择器，表示 p元素下的第一级子元素a

p~a: 兄弟选择器，p之后出现的所有兄弟a， a不必紧跟p的后边

p+a： 相邻兄弟选择器，紧跟其后的第一个兄弟元素

1. **列举你所知道的尺寸单位。**

相对单位长度 px em rem

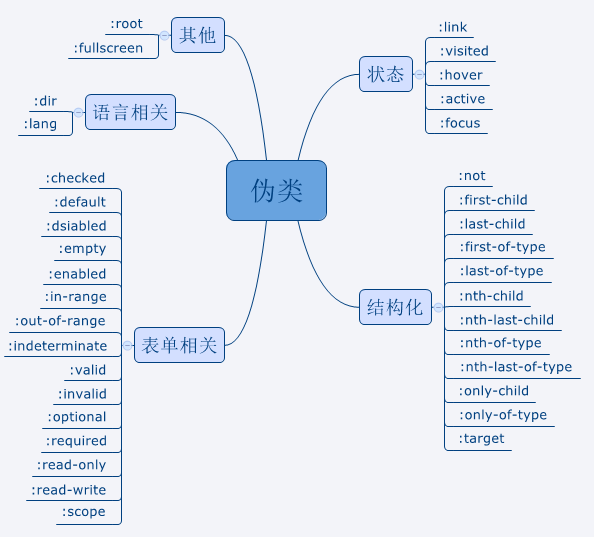
px: 相对长度单位，是相对于显示器屏幕分辨率而言；

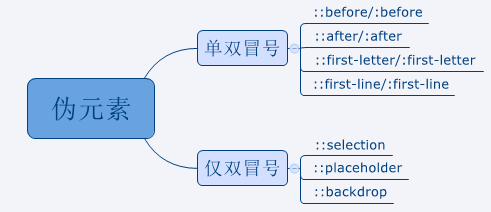
em: 相对长度单位，相对于父元素，1/父元素字体大小\*需要转换的像素值

rem: css3新增，相对html根元素，如果以i5为标准，31.25，1px就是0.01rem

绝对单位长度 in（inch英寸） cm厘米 mm毫米 pt磅 pc（皮卡）

1. **列举你所知道的伪类和伪元素。伪类和伪元素有什么区别？**

常见的伪类: 

常见的伪元素: 

区别：

写法上的区别： CSS3规范中早已明确规定，伪类使用单冒号(:)来表示，伪元素用双冒号(::)来表示。（例如，::before, ::after等伪元素。:hover, :link 等伪类） 虽然CSS3标准要求伪元素用(::)，但由于兼容性的问题，所以现在大部分还是统一的单冒号。

其实两个的根本区别在于：它们是否创建了新的元素。伪类：是为了通过选择器找到那些不存在与DOM树中的信息以及不能被常规CSS选择器获取到的信息伪元素：是在DOM树中创建了一些抽象元素，这些抽象元素是不存在于文档流中的

1. **node.js有哪些模块？**

util是node 里面一个工具模块，node里面几乎所有的模块都会用到这个模块

功能：1实现继承这是主要功能

      2：实现对象的完整输出

      3：实现判断数据类型

path模块：

功能：格式规范化路径

fs模块：

功能：1：操作文件

2：操作目录

http模块：用于搭建HTTP服务端和客户端

url模块：用户解析和处理URL字符串，有三个方法

url.parse(将url字符串解析并返回一个url的对象)，

url.format(将url对象编程一个url字符串并返回)

url.resolve(将url中的参数用/进行拼接)

zlib模块：提供了用Gzip和Deflate/Inflate实现的压缩功能

http://blog.csdn.net/chy555chy/article/details/52663156

socket.io: 实现客服端与服务端之间的实时通信方式，

uglify-js: 用来压缩合并js文件

1. **推荐使用的框架和包：express,**

express是一个基于Node.js平台的极简灵活的web应用开发框架，和丰富的HTTP工具，能够快速的搭建一个完整功能的网站。

Express框架的核心特性

1. 可以设置中间件来响应http请求
2. 定义了路由表用于执行不同的HTTP请求动作
3. 可以通过向模板传递参数来动态渲染html页面
4. **webpack配置流程？**

module.exports = {

entry: //入口文件

output: {},// 出口

devtool: 'source-map', //开发工具

resolve: {},// 别名配置

devServer: {},// 服务配置

module: { //配置各种loader

rules: [{}]

}

},

plugins: [], //插件配置

externals: {} //从外部引入框架

};

参考：<http://www.jianshu.com/p/42e11515c10f>

1. **配置服务，配置公共模块抽离**

配置服务：

devServer: {

contentBase: "./public", //本地服务器所加载的页面所在的目录

historyApiFallback: true, //不跳转

inline: true,

hot: true,

host: 'localhost',

port: 4000,

noInfo: true,

proxy: {

'/api': {

target: 'https://api.douban.com/',

changeOrigin: true,

pathRewrite: {

'^/api': ''

}

}

}

},

公共模块抽离：

plugins:[

// 提取公共代码

//方式一

new webpack.optimize.CommonsChunkPlugin('common.js'), // 默认会把所有入口节点的公共代码提取出来,生成一个common.js

//方式二，有选择的提取公共代码

// 只提取main节点和index节点

new webpack.optimize.CommonsChunkPlugin('common.js',['main','index']),

//方式三，有选择性的提取（对象方式传参）,推荐

new webpack.optimize.CommonsChunkPlugin({

name:'common', // 注意不要.js后缀

chunks:['main','user','index']

}),

]

1. **Vue中两个不相干模块之间如何进行信息传递？不使用vuex**

可以使用一个空的Vue实例作为中央事件总线。

var bus = new Vue()

// 组件A

bus.$emit('id-selected', 1)

// 组件B

bus.$on('id-selected', function (id) {

console.log(id)

})

参考：http://blog.csdn.net/u013034014/article/details/54574989?locationNum=2&fps=1

例子：https://segmentfault.com/q/1010000007491994

1. **怎么把框架单独打包？**

  //添加了此项，则表明从外部引入，内部不会打包合并进去

    externals: {

        jquery: 'window.jQuery',

        react: 'window.React',

    }

1. **自己怎么实现require.js?**

参考：https://www.cnblogs.com/yexiaochai/p/3961291.html

1. **vue组件信息传递**

使用props传递数据---组件内部

//html

<div id="app1">

<i>注意命名规定：仅在html内使用my-message</i>

<child my-message="组件内部数据传递"></child>

</div>

//js

<script>

Vue.component('child', {

props: ['myMessage'],

template: '<mark>{{ myMessage }}<mark/>'

});

new Vue({

el: '#app1'

})

</script>

动态props通信---组件与根节点(父子之间)

<div id="app2">

<input v-model="parentMsg">

<br>

<child :parent-msg="parentMsg"></child>

</div>

<script>

Vue.component('child', {

props: ['parentMsg'],

template: '<mark>{{ parentMsg }}<mark/>'

});

new Vue({

el: '#app2',

data: {

parentMsg: 'msg from parent!'

}

})

</script>

参考：https://www.cnblogs.com/frank26/p/6156378.html

**66.http协议**定义:  
HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。  
HTTP之状态码 :   
状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:  
1xx：指示信息--表示请求已接收，继续处理  
2xx：成功--表示请求已被成功接收、理解、接受  
3xx：重定向--要完成请求必须进行更进一步的操作  
4xx：客户端错误--请求有语法错误或请求无法实现  
5xx：服务器端错误--服务器未能实现合法的请求  
**67.vue指令**v-on  监听DOM事件,比如v-on:click=”handleFunction”,可简写为 @click  
v-bind  绑定属性,比如 v-bind:href=”url”,可简写为 :href  
v-for  循环列表  
v-if  根据表达式seen的真假来插入/删除对应标签,比如 v-if=”seen”  
v-else  必须跟在v-if后  
v-show  根据表达式的真假值来切换元素的display CSS属性  
v-model  表单元素的数据双向绑定  
v-text  定义元素文本, 比如 v-text=”message”  
v-html  更新元素的innerHTML  
v-once  只渲染元素和组件一次    
**68.webpack打包流程**流程图: C:\Users\fan\AppData\Local\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttps://img.alicdn.com/tps/TB1GVGFNXXXXXaTapXXXXXXXXXX-4436-4244.jpg   
**69.一像素线问题  C:\Users\fan\AppData\Local\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttps://www.cnblogs.com/lunarorbitx/p/5287309.html**  
方案一:  使用图片   border-image  兼容IE11+  
@media screen and (-webkit-min-device-pixel-ratio: 2){   
    .border{   
        border: 1px solid transparent;  
        border-image: url(border.gif) 2 repeat;  
    }  
}  
方案二:  使用伪类:before, :after 和transform:scale(…)  
@media screen and (-webkit-min-device-pixel-ratio: 2){  
    .radius-border:before{  
        content: "";  
        pointer-events: none; /\* 防止点击触发 \*/  
        box-sizing: border-box;  
        position: absolute;  
        width: 200%;      height: 200%;  
        left: 0;        top: 0;  
        border-radius: 8px;  
        border:1px solid #999;  
        -webkit-transform(scale(0.5));   transform(scale(0.5));  
        -webkit-transform-origin: 0 0;    transform-origin: 0 0;  
    }  
}  
方案三:  设置viewport  (淘宝移动端采用这种方案)  
var metaEl = doc.createElement('meta');  
metaEl.setAttribute('name', 'viewport');  
metaEl.setAttribute('content', 'initial-scale=' + scale + ', maximum-scale=' + scale + ', minimum-scale=' + scale + ', user-scalable=no');  
如果 dpr是2 ,生成的标签如下:  
<meta name="viewport" content="initial-scale=0.5, maximum-scale=0.5, minimum-scale=0.5, user-scalable=no">  
**70.手写flex相关的属性，闭包，作用域，定时器，vue，手写hellworld，代码逻辑思路，**flex相关的属性: flex-direction  flex-wrap  flex-flow  display:flex  flex-shrink    
闭包 :  闭包就是能够读取其他函数内部变量的函数  
作用域 : 在作用域中定义的所有变量在代码块之外都是不可见的  
定时器 : setTimeout  setInterval  
**71.对象，数组，Es6新语法，**对象语法: Object.assign(obj1,obj2)  
Object.keys(obj)  
数组方法: map, reduce, filter