

详细参见：<http://blog.csdn.net/fengyinchaoy/article/details/52303118>

30.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/fengyinchaoy/article/details/52303118>

31.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

32.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

33.百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>

34.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

35.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)

36.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>

37.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>

38.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

39. UIWebView和JavaScript之间是怎么交互的

? UIWebView是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 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。

40.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Native 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js组件，对象为plus

apiloud引入各种第三方插件，对象为api

顺便提一下，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

41.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

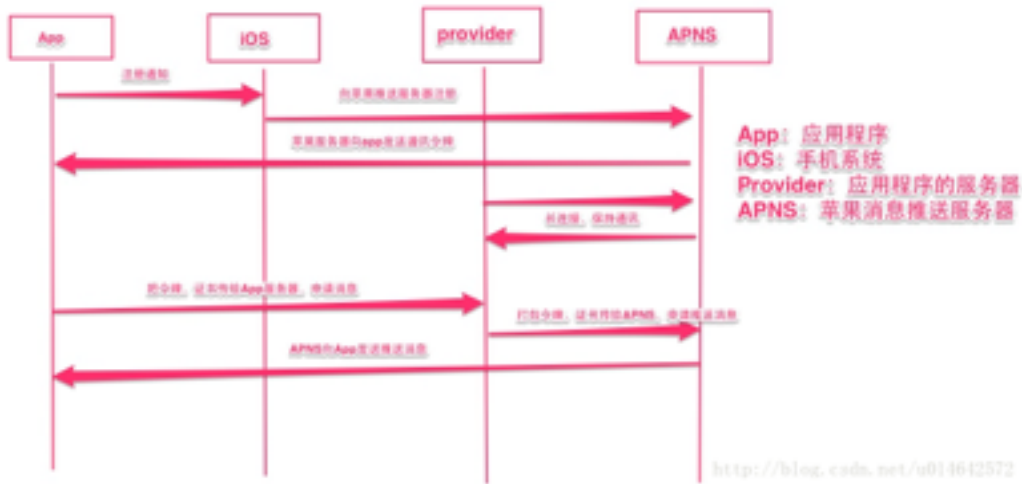
一种是远程消息推送，主要应用联网设备的信息推送，例如：邮件，各种软件的广告或优惠信息的推送。远程推送比较复杂，需要使用开发者账号进行申请证书，获得实

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



42.什么是代理和通知，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 `respondToSelector` 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 iOS 4.0+ 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 语法块代码以闭包的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>

43. `UIViewController` 的生命周期

(1) 通过 `alloc init` 分配内存, 初始化 `controller`。

(2) `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`, 然后赋给 `controller` 的 `view`)

(3) `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。)

(4) `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。)

(5) `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)

(6) `viewWillDisappear` (当视图将从屏幕上移除时候调用)

(7) `viewDidDisappear` (当视图已经从屏幕上移除时候调用)

(8) `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>

44.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>
```

45.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}

46.微信支付怎么做？说说流程

(1)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。

(2)获取商户支付配置信息及支付测试配置：

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。

(3)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）

(4)调用统一下单API，生成预付单

(5)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

(6)返回支付参数prepay_id,paySign参数的html文本给前端。

(7)微信浏览器自动调起支付JSAPI接口支付，提示用户输入密码。

(8)确认支付，输入密码，提交支付。

(9)步通知商户支付结果，商户收到通知返回确认信息。

(10)返回支付结果，并发微信消息提示。

(11)展示支付信息给用户，跳转到支付结果页面。

47.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。

48.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

NodeJS

49.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>

50. 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/>

51. 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 apps
Simple
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

52. 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

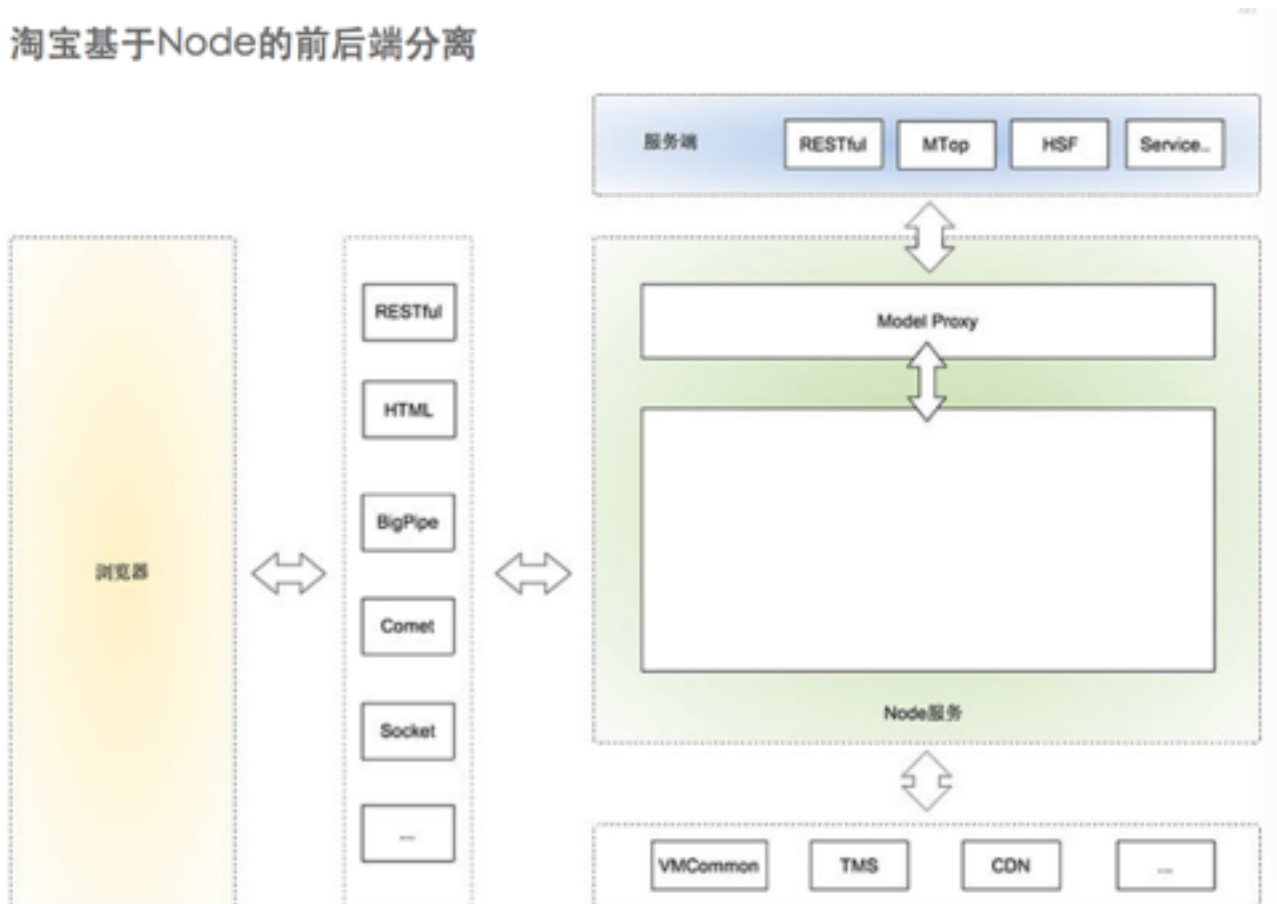
详细参见: https://segmentfault.com/a/1190000006199237?utm_source=tuicool&utm_medium=referral

53. 你们原来公司如何发送的新消息推送?

参见第47第48题。

54. 如何用NodeJS搭建中间层?

淘宝基于Node的前后端分离



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。这一层主要目的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。

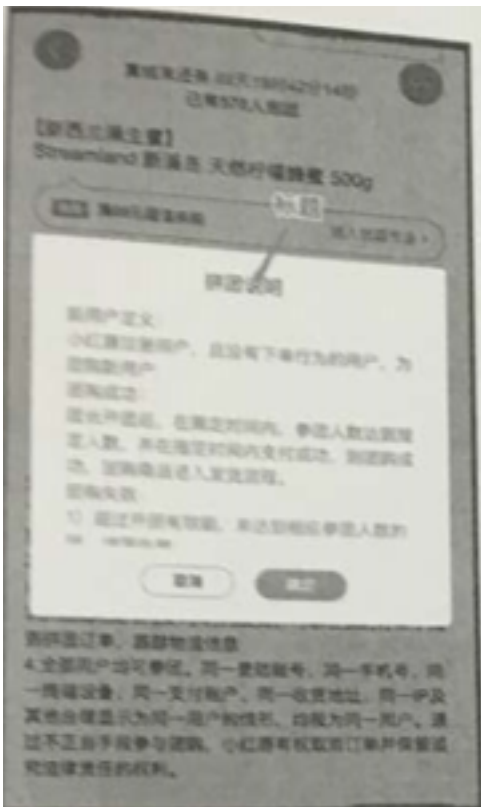
浏览器层在我们这个架构中没有变化，也不希望因为引入Node改变你以前在浏览器中开发的认知。

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

55.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。

56.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)

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

57.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>

58.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

59.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>

60.权限管理如何实现?

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

61.一个大数组,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

62.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

63.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

64. 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。

65. 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

66. 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

67.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>

68.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>

69.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>

70.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>

71.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>

72.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

73.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+iView

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

74.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>

75.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

76.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>

77.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>

78.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>

79.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>

80.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>

81.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

82.jQuery源码中值得借鉴的？

- (1) 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。
- (2) 在设计程序时，要结构清晰，高内聚，低耦合。
- (3) 利用多态的方式，实现方法的重载，提高代码的复用率
- (4) jQuery的链式调用以及回溯
- (5) jQuery.fn.extend 与 jQuery.extend方法来实现扩展静态方法或实例方法

83.\$.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document.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。

```
DOMContentLoaded = function(){
  //取消事件监听，执行ready方法
  if ( document.addEventListener ){
    document.removeEventListener( "DOMContentLoaded",DOMContentLoaded, false );
    jQuery.ready();
  }
  else if( document.readyState === "complete" ){
    document.detachEvent( "onreadystatechange", DOMContentLoaded );
    jQuery.ready();
  }
};
```

84. 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。

85. 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。

86. 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，也就满足了常量的性质，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const a=2; CONST.a是不可以更改的常量;

```
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
```

87.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')
```

88. 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。

89.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
```

90.深拷贝和浅拷贝以及应用场景

1.浅拷贝

//拷贝就是把父对象的属性，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。

91. 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，会先在实例自身上找，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

92.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bootstrap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

}}

93.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

}}

94.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

```
}}
```

95. 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

```
}}
```

96.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  
}}
```

97.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  
}}
```

98.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

```
}}
```

99.两个平级界面块 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，关键是如何解决思路。

```
}}
```

100.一个 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

101.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
}
```

102. 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  
}
```

103. 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  
}
```

104. 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
}}

105.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  
}}
```

106.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。  
}}
```

107.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  
}}
```

108.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

109.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

110.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...})

111.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 代码来说是起不起作用的，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。

112.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/yymm2mf>

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

综合问题

113.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

114.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

115. ES6里面你用过什么？

- (1) 块作用域 - let
- (2) 常量 - 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) super
- (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/>

116. 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>

117.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。

118.你平时写过什么业务逻辑？

根据自己做的项目对答。

119.你负责的具体是什么模块？

根据自己做的项目对答。

120.你掌握的技术栈有哪些？

说自己了解的和擅长的。