​

手动整理的前端技术(主要是react)，保证干货～最近大家都在分分火火的各种面试，酸甜苦辣咸都看到了，希望这些能给大家带来一点帮助，同时希望能有人指出其中的不足和理解错误的地方。

好的技术永远不要藏藏捏捏，不要等自己过时了才恍然大悟。最佩服的就是Qcon的那些布道者，走在技术的前沿，这才是推动人类进步的力量好伐。

感谢jiljiang，你简直是我人生的导师❤

感谢要花蔡，祝阿里转正顺利

感谢Alex不计前嫌拉我入伙，么么哒（\*＾3＾）

第一部分：纲要

校招面试会涉及到知识的深度

1. jQuery源码读一下，并理解。如jq的对象拷贝，深拷贝浅拷贝等等。

2. 有能力读一下React/Vue的源码，理解他们的实现机制

3. vue2.0和react都使用了虚拟dom这一概念，了解怎么实现的。阅读snabbdom这个虚拟dom引擎

4. 前端工程化，webpack、gulp这样的工具

5. es6

6. 前端性能优化

7. http协议，建议阅读http权威指南

8. webApp的实现方案、优化方案

第二部分：关键词和关键句(面试说了展开讲)

1.关键词：同构（isomorphic、nodejs）、服务端渲染、爬虫及爬虫友好、redux、react和VUE

首屏渲染快、共享Store、维护方便、5秒后用户流逝率

2.Ajax和react首屏渲染：

一个是HTML完成ajax拿数据，一个是页面自带数据。

第三部分：细节

分

-----------------------------------------------

A.Flux和Redux

biubiubiu写在最前面

觉得看完我say的还是觉得有点云里雾里，或者觉得还是看网页舒服的可以移步：

阮一峰大大关于redux的讲解：

http://www.ruanyifeng.com/blog/2016/01/flux.html

淘宝前端国际化团队的redux心得：

http://www.aliued.com/?p=3128

要讲redux，不得不了解一个概念flux。首先这是一个基于单向数据流的概念。

Vue2已经完全推崇单向数据流不在支持双向数据绑定啦！童鞋们多多注意！

flux将页面分为四个部分：Action，Dispatch，Store，View。

先上流程图(取自第二个链接

￼

View部分就是大家熟悉的MVC的view，视图部分，这部分主要包含UI的绝大部分展示与实现。

react.js就是一个纯粹的View层实现框架，render渲染，数据变现，就在这部分实现。

这部分是与用户直接交互的重要接口，用户的输入输出全部在这里被记录，存储、以及一小部分处理。

-----------------------------------------------

Action部分顾名思义，动作动作，自然是处理数据的函数集合。这部分往往由View来启动。react的写法

就是dispactch({actionType:LOAD,bugets:result})，当view层发出一个dispatch动作以后，先统统收集到dispatch容器，然后

再由dispatch调用制定的action。这就是传统模型的control操作，只不过被移除了直接操作的权限。

action 的参数用来筛掉脏数据，调用 action 函数的人，有义务只传入它需要的数据。

action 返回的 plain object 中包含属性为 type 的常量值表明这个对象里携带的其他数据应该被如何「再处理」

或者不带其他数据，仅仅启示已有数据需要如何调整，也可以提出需要获取哪些数据。

-----------------------------------------------

dispatch，可以从前面看到，他是连接view和action的重要部分，不过也别忘记，action最终要操作的实体store也是由这部分分发出去的。dispatch是全局的，在我看来就像一个上帝一样，拿着一根赶

-----------------------------------------------

Store大家已经很熟悉，这部分是包含了若干个Reducer和一个state的容器，另外，大部分store不直接接受action来操作自己的reducer，而是借助一个老鸨，middleWare，也就是传说中的中间件来进行相关操作。他的作用有点像传统MVC模型的modal部分。

-----------------------------------------------

下面是整个redux的实现过程：

页面加载，用户打开screen，screen从view获取渲染完的页面，用户动动mouse，view层抓取到相关数据的改变，通过dispatch方法传递给用户改变的数据和需要执行的操作action，action经过middleWare操作store的reducer，reducer一旦发生改变，就会被react的dom diff给检测到，从而进行setState，告诉view层我的数据改变了，你应该重新渲染了，然后view重新渲染，新页面通过screen映入用户眼帘。

redux和flux的区别就在于，redux可以将state和reducer彻底分离开来，作为两份对象来分别对待与处理，而且redux只有一个state，就是store.getState()。并且他是只读的，不存在修改state，state每次改变redux都是采用的生成新版本的state，而不是修改原来的state，就像深浅拷贝的区别一样。

加上这种类似于git的HEAD流方式存储state，我们可以方便的进行调试与测试，测试一个case大可不必从头开始,加上server render以后，用户可以方便的从服务器接过来的state开始渲染，因此这是实现同构的最重要基础，没有之一。

最后，从性质上来讲，flux是一种技术，redux是一种具体的实现。

分

-----------------------------------------------

B.同构(Isomorphic)

先上干货链接：

淘宝前端国际化团队的redux同构开发心得：

http://www.aliued.com/?p=3077

引言与问题：

1.为什么要首屏服务器端渲染(同构)？

2.服务端渲染通过ReactDOMServer.renderToString 和 ReactDOMServer.renderToStaticMarkup将虚拟DOM渲染出来后传输的是什么数据给用户？

答案：

爬虫是指通过网络的遍历获取网页HTML实体DOM内容的程序，是搜索引擎的最重要的一部分。

众所周知，绝大部分爬虫是不支持js语句识别运行的，代价太大。当然谷歌爸爸在09年开始就支持js读取识别，然鹅，即便是google这样的巨无霸搜索引擎，也仅仅能做到识别特定限制范围的js语句，例如js window.location重定向。

在现在这个风起云涌、数据引领的大前端时代，已经完全难以跟上。

刚才讲到爬虫，现在就来讲讲同构的第一个好处，最商业话的好处：SEO友好。

对于一个中大型的react项目，采取的往往是redux的数据存储，render对DOM虚拟树进行识别渲染，生成HTML页面，所以在没有采用isomorphic的情况下：

往往是项目代码先到达，数据随后到达，然后代码调用数据进行客户端渲染，然鹅可怜的爬虫不具备客户端浏览器的这些功能，往往爬虫能爬取到的只是服务器端传给他的虚拟Store数据流和一堆爬虫根本读不懂的JSX代码(JSX是react发明的，遇到<当html解析，遇到{当js解析)。不懂的同学可以想一想英语六级考试阅读理解要你找出题目主线的时候，啊是迷茫爆表

那么，你也许已经知道我要说什么了，服务器端在第一次和客户机（包括爬虫）通讯传递数据的时候，他帮助可以爬虫先进行一下网页的渲染呀！

和 ReactDOMServer.renderToStaticMarkup ，生成一堆代表完整HTML页面的字符串，这样爬虫就能识别了！就像帮你把阅读理解给翻译成中文一样，感觉是不是很酸爽～

如果您觉得生成的字符串量比较大可能会影响，可以用webpack-isomorphic-tools(以下简称同构一代工具

顺带提一下这个库的作者最近写了一个新的打包管理服务器端代码的工具universal-webpack(同构二代工具

balabalabala

同构一代二代工具

有人会说npm不是打包工具吗？和gulp有个杰宝关系？npm是打包工具不错，但是很久之前，npm的package.json里面有个scripts属性可以定义当前模块的一些构建功能，

比如大家熟悉的源码uglify：

（有没有人觉得丑化ugly非常生动形象的！握爪

build：uglifyjs Underscore.js|bootstrap.js

以及大家测试经常要用的：

test: phantomjs a/index.html

gulp和npm相比其实大致是一个从属关系，npm包含gulp的大部分功能，但使用起来较为繁琐。

gulp是工具链、构建工具，可以配合各种插件做js压缩，css压缩，less编译，替代手工实现自动化工作。

如果非要列举出他们的区别：

gulp：

1.构建工具 2.自动化 3.提高效率用

npm：

1.打包工具 2.模块化识别 3.编译模块代码方案用

回到我们的ReactDOMServer.renderToString

和 ReactDOMServer.renderToStaticMarkup ，

业界目前普遍采用的上述方式进行服务器端渲染，可是如果只是纯粹的，服务端渲染完就把数据传输出去不做任何处理，这种方式在我看来还是比较危险的，会存在很多bug和问题，这也是同构这个潮流向前迈进要解决的。如果大家有所想法，不妨试一下递归render，这部分涉及公司内部的代码，就不和大家细致描述了。

好了，继续刚才讲的，我们讲完了第一个好处，爬虫友好，现在讲讲第二个，我称之为减轻5秒症状，不知道你们有没有听过前端有一个专业词：叫用户流逝率，根据腾讯的16年11月份的大数据报告，加载超过5秒就有74%的用户离开，8秒后达到99%

好了，讲了这么多引子，引入这个关键词，同构的好处之二：首屏渲染快。

我们知道，一个网站基本都有多个模块，比如首页是数据展示，下属部分比如图表展示、用户操作等等。

以首页为例子，当服务器第一次收到了用户请求的时候，只挑选用户需要展示的这个部分模块进行渲染，

其余全部在客户端通过react的dom diff机制来动态挂载。这样用户的浏览器拿到的数据不仅仅是直接拿来可用的html数据，而且数据传输量大大降低，再加上前面提到的同构一代工具

这样，进入用户眼帘的第一页可以非常快。

快到什么地步呢：往往用户还来不及叉掉你的

网站，网站就已经在用户面前活蹦乱跳了。

ok，讲到这里，基本上就将同构的第二个好处首屏渲染快给讲清楚了。

以上都是站在用户的角度来讲的，现在站在开发者的角度来思索这个问题。

对于一个小型项目，就四五个页面，四五个api的那种，完全不需要也不建议用同构技术，甚至对于redux的store使用我都强烈建议绝不要使用多个store，往往几个api对应几个reducer，数据处理完以后用combinereducers进行数据打入即可。(看来在这一点上我已经和redux的作者们完全达成一致了呢

然鹅，sorry我又要讲到然鹅了，对于大型项目，复杂型项目，一个页面涉及多个api交互，甚至一个页面要进行大量的用户输入数据处理的那种项目，就需要大量的reducer来存放数据，甚至有些极端的项目需要根据需求使用两个甚至多个store来解决了。

好了，这样问题就来了，暂时撇开同构生成的页面对会让老板更舒爽、更赏心悦目不提，你那么多数据容器，啧啧啧，维护起来堪忧

如果现在需要你做前端的优化，除了

哥哥哥，再顺带讲一下前端优化吧！面试必问！纯干货！

1.减少HHTTP请求合并重复API

2.采用HTTP2新技术方案

3.减少重定向和DNS查找

4.合并图片

5.减少DOM操作（ps:我个人认为使用react-redux、Vue-Vuex还用超过两个getelmentById的是挺犯傻的行为）,

6.采用json来进行数据交换：json是原生的js数据格式，不需要其他的包协助处理，这个基本不用说了，我们做服务器端的cache和storage存储的时候，已经开始大量使用json，没有哪个傻瓜会不用json而去自建数据结构，自建数据处理方法。json的可读性和可操作性（js操作）太美。

7.然后是采用less、sass来替代传统的css解除代码冗余

8.还有CDN内容分发网络，但是对于公司内部项目，往往不被批准，就算是蚂蚁这种能提供自家CDN的服务的公司，你把自己部门的内容全部放在别的部门的CDN服务器上，甚至包含很多包含密码和TOKEN的config文件夹，这肯定不行。退一万步讲，就算你只放css和js代码，那CDN的实时性也太差，源头早就改成alert(新春快乐！)了，CDN服务器上还在返回（除夕快乐！）的请求，这特么就有点尴尬了

9.最后就是延迟加载js和压缩技术：图片精灵，css、js压缩处理、GZIP，这些技术非常靠谱，但用多了反而会拖累服务器运行速度，毕竟服务器只是用来和客户机交互传递数据的呀！

好了，顺手讲了一下前端优化，回头我们看看，这些处理究竟有几个比得上首屏渲染。又有几个可以达到首屏渲染那种飞一样的感觉

有大牛可能会说，我一定要用同构吗？人家就是超级超级不想采用react+nodejs的首屏渲染技术，非要采用Angular2或者Pulsar开源框架；

更加厉害的大牛会说：我就不想世界和平安定，不想要同一个代码能同一个世界运行，我就是这样的python、ruby大神不在乎使用多份代码处理前后端，并且后台生成前端页面能达到与render方式完全一致。

那么，有个问题你必须要考虑，那就是数据的维护，同构可以让你使用一个数据模型，不管你是几个数据池几个store几个reducer，只要代码前后端一致，那么完全不必在意特地去维护数据，修改需求也比较方便、没有那么繁琐。

献上爱心

ps：

Aurelio最近讨论了同构JavaScript应用。实质上，它是将渐进增强（progressive enhancement）推到了极致：应用可以在客户端或服务器上快乐地运行了。使用现代浏览器的用户可以使用单页应用。老式浏览器和搜索引擎爬虫将会看到服务端渲染的应用。理论上，应用可以根据设备的速度和带宽能力来实现不同等级

的客户端/服务器处理。

分

-----------------------------------------------

C.跨域请求几种主流在用的方式

1.我们都知道，凡是拥有Src属性的标签都有跨域能力，如：<script><img><iframe>，因此我们可以在web页面

调用js文件，而json是一种支持js的字符格式，因此可以通过读取json（类似于读取js）来获得想要使用的数据

这种方式就是最著名的jsonp方式

2.最新的也最不稳定的方式：websocket，注意，websocket就是为非同源策略而生的，是为了两个远端的

个体之间全双工的双向通信而服务的，因此当js调用了一个ws的请求之后，先通过一个http请求连接A,B,

紧接着AB之间建立websocket协议连接。一旦连接建立，随你传递什么数据都可以了。

3.另外一个比较新的方式是H5的postMessage方式，这个方式目前还没有被广泛采用，原理同jsonp

4.document.domain + iframe,主域相同的时候直接可以使用这个方法，最简单。

5.利用js可以跨域读取，动态创建js，这个方法理论上可以，但是实际操作肯定远比不上jsonp

6.location.hash + iframe

通过隐藏的iframe来传递、操作hash值

7.同主域下，利用一个元素的window.name属性可以存储2M的数据量,来进行数据

分

-----------------------------------------------

D.排序算法js实现

http://www.cnblogs.com/dushao/p/6004883.html

分

-----------------------------------------------

E.闭包以及闭包存在的历史

啥？都拥抱es6了，你还闭包？

哈哈哈，开个玩笑，对于闭包的历史，就是js的血泪史啊！同志们还是需要去研究一下的！

百度直接搜索闭包阮一峰即可。有兴趣的同学也可以去看一下阮老师的es6，真的很厉害。