3

【译】JS 工程师应该知道的 10 个面试问题

lizheming 2015-10-30 4645 访问 JAVASCRIPT

原文：https://medium.com/javascript-scene/10-interview-questions-every-javascript-developer-should-know-6fa6bdf5ad95
译者：@lizheming

---

1. 你能列举出两个 JS 应用开发的编程范式吗？

JS 是一种多范式的语言，通过 **OOP** （面向对象编程）和 **函数式编程** 其支持 **命令式/过程式** 编程。JS 通过 **原型链继承** 方式支持 OOP。

加分点：

- 原型链继承（或：属性，对象链接对象即 OLOO）。
- 函数式编程（或：闭包，头等函数，lambdas）。

扣分点：

- 不知道编程范式是什么，回答中没有提及到原型链面向对象对象或函数式编程。

补充学习：

- The Two Pillars of JavaScript Part 1—原型链面向对象。
- The Two Pillars of JavaScript Part 2—函数式编程。

2. 什么是函数式编程？

函数式编程通过构造各种运算函数来组织程序，尽量避免在代码间共享状态和变量数据。Lisp（1958年指定）深受 lambda 运算表达式的启发，是最早支持函数式编程的语言。直到如今 Lisp 和它的衍生语言仍在普遍使用。

函数式编程是JavaScript的一个基本概念（JavaScript的两大支柱之一）。ES5 中增加了一些常用的函数功能组件。

加分点：

- 纯函数。
- 避免除了输出之外的副作用。
- 简单的函数组合。
- 函数式编程语言列举：Lisp, ML, Haskell, Erlang, Clojure, Elm, F#, OCaml 等。
- 函数式编程的特征：头等函数，高阶函数，函数可做为参数和变量值。

扣分点：

- 没有提及纯函数或避免函数副作用。
- 无法列举出函数式编程的语言。
- 无法列举出函数式编程的特点。

补充学习：

- The Two Pillars of JavaScript Part 2.
- The Dao of Immutability.
- Professor Frisby's Mostly Adequate Guide to Functional Programming.
- The Haskell School of Music.

3. 类继承和原型链继承的区别是什么？

类继承：通过类进行实例化继承，类的实例化一般是通过对构造函数使用 `new` 关键字来生成的，在 ES6 中也可以使用 `Class` 关键词来声明类。

原型链继承：通过其他对象直接进行实例化继承，实例化一般通过工厂函数或者是 `Object.create()` 函数。实例可以由多个不同的对象组成，也可以非常方便的选择继承对象。

> 在 JavaScript 中，原型链的继承要比类继承更简单易用。

加分点：

- Classes: create tight coupling or hierarchies/taxonomies.
- Prototypes: mentions of concatenative inheritance, prototype delegation, functional inheritance, object composition.

扣分点：

- 没有提及原型链相对于类继承和示例化的好处。

补充学习：

- The Two Pillars of JavaScript Part 1—Prototypal OO.
- Common Misconceptions About Inheritance in JavaScript.

4. 函数式编程和面对对象编程之间的优缺点是什么？

OOP 优点：非常容易理解对象的基本概念，也非常容易解释各成员函数，OOP相比非常容易阅读的声明式编程而言，更倾向于命令式编程。

OOP 缺点：#### 1. 你能列举出两个 JS 应用开发的编程范式吗？

JS 是一种多范式的语言，通过 **OOP** （面向对象编程）和 **函数式编程** 其支持 **命令式/过程式** 编程。JS 通过 **原型链继承** 方式支持 OOP。

加分点：

- 原型链继承（或：属性，对象链接对象即 OLOO）。
- 函数式编程（或：闭包，头等函数，lambdas）。

扣分点：

- 不知道编程范式是什么，回答中没有提及到原型链面向对象对象或函数式编程。

补充学习：

- The Two Pillars of JavaScript Part 1—原型链面向对象。

- The Two Pillars of JavaScript Part 2—函数式编程。

5. 什么合适的时机去使用类继承？

这是一个欺骗性的问题，答案是永远不要用。我花了很多年去得到这个答案，但是我听到的答案总是一些常见的误解，更多的是没有答案。

> “ 如果一个功能有时有用，有时危险，如果有更好的选项，那就一直*用这个更好的选项。
> —— Douglas Crockford

加分点：

- 很少，几乎不用，从来不用
- 优先使用对象组合而不是类继承

扣分点：

- 其他回答
- “React 组件” ——不，类继承的陷阱不会因为一个新框架而改变，哪怕有 `class` 的关键字，和普通的看法不同，你不需要在 React 中使用 `class` ，这个回答表示了对 `class` 和 React 的误解

补充学习：

- The Two Pillars of JavaScript Part 1——Prototypal OO.
- [JS Objects——Inherited a Mess] (http://davidwalsh.name/javascript-objects)

6. 什么合适的时机去使用原型基础

有好几种原型继承：

- 委托（比如原型链）
- 拼接（比如 mixing、`Object.assign()` ）
- 函数化（不要和函数式混淆，一个使用闭包的函数可以保护私有状态或者封装）

每一种原型继承都有自己的一套用例，但是都能够使用各自的能力去实现诸如 has-a 、uses-a 或者 can-do 关系，相对类继承的 is-a 关系。

加分点：

- 模块和函数式编程无法提供一个明显的解决方案的情况
- 当你需要从多个资源组合对象
- 任何你需要继承的时机

扣分点：

- 对合适使用原型继承没有认知
- 对 mixins 或者 `Object.assign()` 有任何了解

补充学习：

- “Programming JavaScript Applications”: Prototypes section.

7. “优先使用对象组合而不是类继承” 是什么意思？

这是从 “Design Patterns: Elements of Reusable Object-Oriented Software” 的引用，这意味着应该使用更小的函数型单元来复用新的对象，而不是从类里面继承来创建新的分类。 也就是说，用 can-do, has-a, uses-a 关系来代替 is-a 关系。

加分点：

- 避免类层次结构
- 避免脆弱的基类的问题
- 避免紧密解耦
- 避免严格分类（强制 is-a 关系最终会在新用里中报错）
- 避免大猩猩香蕉问题（你只想要一只香蕉，但是你得到一个拿着香蕉的大猩猩和整片丛林
- 让代码更加灵活

扣分点：

- 没有提到上面的任何问题
- 没有明确指出组合和类继承之间的区别，或者组合的优点

补充学习：

- https://youtu.be/wfMtDGfHWpA
- https://medium.com/javascript-scene/introducing-the-stamp-specification-77f8911c2fee

8. 什么是双向绑定和单项数据流，他们之间有什么区别？

双向绑定是说 UI 模型和数据模型绑定，当界面发生变化的时候，数据模型也会跟着变化，反之亦然。 单向数据流是数据模型是唯一资源，改变界面只会触发消息通知把用户意图发送给数据模型（类似 React 中的 “store”），只有数据模型才能去改变应用的状态，这样就保证数据永远朝一个方向流动，能够让代码非常容易理解。

单向数据流是确定的，然而双向数据绑定会有副作用让数据流难以琢磨和跟踪。

加分点：

- React 是一个在单向数据流方向的新权威例子，所以提到 React 也是一个好的信号，Cycle.js 是另一个比较流行的单向数据流的实现。
- Angular 是一个使用双向数据流的流行框架

扣分点：

- 不知道这两者的意思，不能很好解释两者的区别。OOP通常依赖于同享的状态，很多对象和行为通常会争夺同一个类似可能不确定顺序的随机被一些函数存储的资源，竞争可能会造成不可预计的后果。

补充学习：

- https://youtu.be/XxVg_s8xAms

9. 整体架构和微服务架构的优缺点

整体架构是指你的应用里的代码是使用一个精密结合的单元，所有组件使用同一个内存空间和资源来共同工作。微服务架构是指你的应用由很多小而独立的应用块组成，他们跑在各自的内存空

间里，彼此之间能够各自扩展甚至能够跨机器。

整体架构优点：整体架构的主要优势在于大多数的应用需要关注很多点，比如日志，速度限制，甚至一些诸如审查跟踪和 DOS 保护之类的安全功能。

当一切通过同一个应用运行起来的时候，这样能够轻松把所有组件连接起来。

这样同时也有性能优势，因为共享内存要比跨内存通信快。

整体架构缺点：整体架构的代码一般是紧耦合，在版本迭代的时候非常困难，如果想让服务独立扩展或者提高代码可维护性会变得非常困难。

整体架构会变得越来越难理解，当你关注服务细节或者控制器细节的时候，会有一些看不到的依赖、副作用及不可思议的问题。

微服务架构优点：微服务架构通常更容易组织，而且每个部分有各自的功能，不会去关心其他组件的功能。解耦服务也非常容易重构和重新配置来为不同的应用提供服务。（比如同时提供网页客户端和开放 API 的服务）

只要组织的合理，也会有比较好的性能，因为能够支持服务热插拔来支持应用剩余的部分。

微服务架构缺点：当你在构建一个新的微服务架构的时候，可能会在设计阶段发现很多组件之间的通讯需要关心，但是在整体架构中可以一些共享的函数或者中间件非常轻松去处理跨组件通讯。

在微服务架构中，你需要去承担隔离组件之间通讯的开销，或者封装起来到另一个服务层中来让所有的组件能够使用。

最终，虽然整体架构能够通过外层服务层来保证所有组件的通讯，但是整体架构可能会把这个工作的开销拖延到哪怕项目变得越来越成熟。

微服务架构通常会部署在各自的虚拟机或者容器中，能够通过快速管理工具自动化完成任务。

加分点：

- 对微服务抱有倾向态度，尽管相比整体架构刚开始有比较大的开销。认识到微服务架构能够越来越容易扩展。

- 实际关于微服务架构 vs 整体架构的应用，组成的应用中的服务在代码层对彼此之间独立，但是在开始的时候整体架构能够把他们都捆绑起来，微服务架构能够把过高的开销延后到直到项目越发成熟。

扣分点：

- 两者架构没有认知
- 对微服务的开销没有认知或者不切实际的想法
- 对于跨进程通讯和跨网络通讯的开销没有认知
- 对微服务的缺点太过消极，不能明确表达如何如何解耦整体架构来轻松分割进微服务架构的时候。
- 低估了微服务架构的扩展优势。

10. 什么是异步编程？为什么在 JavaScript 中这么重要？

同步编程是指，除了条件语句和函数回调，代码会从上至下依次执行，会被类似网络请求或者磁盘读写这样长时间运行的任务堵塞。

异步编程意味着引擎运行在一个事件循环中，当需要做一些会堵塞的操作，就发一个请求，代码会继续运行而不堵塞，当响应准备好的时候，中断就会被触发，然后调起一个控制流的事件回调去运行，这样一来，一段简单的程序能够并发执行很多操作。

用户界面天生异步，需要等待用户输入来中断事件循环然后触发事件回调。Node 是默认异步的，意味着服务也是同一种方式，循环等待网络请求，当第一个请求在处理的同时还要接受更多请求。

这在 JavaScript 中非常重要，因为他非常适合编写用户界面代码，在服务端也非常高效。

加分点：

- 知道什么是阻塞，以及性能的意义
- 知道事件回调，以及为什么对界面代码的重要性

扣分点：

- 对同步和异步的概念不熟悉

- 不能表达性能影响以及异步代码和界面代码和异步的关系

补充学习：

- https://youtu.be/8aGhZQkoFbQ

结论

这只是面试题，坚持高水平的话题，如果他们能够回答出这些问题，通常意味着他们有足够多的编程经验能够在几周之内解决语言怪癖以及熟悉语法，哪怕他们没有足够多的 JavaScript 经验。

当然也不要忘记考察一下其他比较容易学习的东西（包括经典的算法或一些解谜问题）。

不管怎么说你真正需要知道的是，“这个面试者是否能学以致用？”

> 你也可以问一些 ES6，测试驱动开发，原型链面向对象或者 React 来提升你的面试水平。

共收到3条回复

雨雪天气 2015-10-30 21:18

看到头等函数我查了下原文。。。

话说，虽然有这么多范式。每次写什么还是不由自主强烈的仅仅依靠闭包来封装、共享变量和函数。。。

然后，重构或者调试的时候就呵呵了。。。真难改，得去抄抄别人的代码。。。

回复此楼

lizheming 2015-10-31 21:55

@雨雪天气 原文的术语太多翻译的不好还请见谅 T_T... 范式就和练功一样，秘籍就在那里，关键还是靠自己多学多练。

回复此楼

雨雪天气 2015-11-04 14:12

@lizheming 多谢，翻译得很好啊，我就是不太习惯有些东西的中文表示。。。

回复此楼

只有w3ctech成员才能发表和回复。
请 登录 或者 注册 后发表或回复！