Vue or React面试题汇总

Vue or React面试题汇总

- Vue篇
  - vue2.0组件通信方式有哪些?
  - v-model是如何实现双向绑定的?
  - Vuex和单纯的全局对象有什么区别?
  - Vue 的父组件和子组件生命周期钩子执行顺序是什么?
  - v-show 和 v-if 有哪些区别?
  - computed 和 watch 有什么区别?
  - Vue 中的 computed 是如何实现的
  - Vue 中 v-html 会导致什么问题
  - Vue 的响应式原理
  - Object.defineProperty有哪些缺点?
  - Vue2.0中如何检测数组变化?
  - nextTick是做什么用的，其原理是什么?
  - Vue 的模板编译原理
  - v-for 中 key 的作用是什么?
  - 为什么 v-for 和 v-if 不建议用在一起
  - vue-router hash 模式和 history 模式有什么区别?
  - vue-router hash 模式和 history 模式是如何实现的?
  - vue 中组件 data 为什么是 return 一个对象的函数，而不是直接是个对象?
  - MVVM 的实现原理
  - vue3.0 相对于 vue2.x 有哪些变化?
  - 那你能讲一讲MVVM吗?
  - 你知道Vue3.x响应式数据原理吗?
  - Vue2.x和Vue3.x渲染器的diff算法分别说一下
  - 你都做过哪些Vue的性能优化?
- React篇
  - 你的技术栈主要是react，那你说说你用react有什么坑点?
  - 怎么去设计一个组件封装
  - react 的虚拟dom是怎么实现的
  - react hooks 原理是什么?
  - useState 中的状态是怎么存储的?
  - 如何遍历一个dom树
  - 数据双向绑定单向绑定优缺点
  - React fiber 的理解和原理
  - 解释 React 中 render() 的目的
  - 调用 setState 之后发生了什么?
  - 触发多次setstate，那么render会执行几次?
  - react中如何对state中的数据进行修改? setState为什么是一个异步的?
    - 为什么建议传递给 setState的参数是一个callback而不是一个对象?
    - 为什么setState是一个异步的?
  - 原生事件和React事件的区别?
  - React的合成事件是什么?
  - 什么是高阶组件（HOC）

Vue篇

vue2.0组件通信方式有哪些?

- 父子组件通信:

```
props` 和 `event`、`v-model`、 `.sync`、 `ref`、 `$parent` 和 `$children
```

- 非父子组件通信:

```
$attr` 和 `$listeners`、 `provide` 和 `inject`、`eventbus`、通过根实例`$root`访问、`vuex`、`dispatch` 和 `brodcast
```

v-model是如何实现双向绑定的?

vue 2.0

`v-model` 是用来在表单控件或者组件上创建双向绑定的，他的本质是 `v-bind` 和 `v-on` 的语法糖，在一个组件上使用 `v-model`，默认会为组件绑定名为 `value` 的 `prop` 和名为 `input` 的事件。

Vue3.0

在 3.x 中，自定义组件上的 `v-model` 相当于传递了 `modelValue` prop 并接收抛出的 `update:modelValue` 事件

Vuex和单纯的全局对象有什么区别?

Vuex和全局对象主要有两大区别:

1. Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
2. 不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

Vue 的父组件和子组件生命周期钩子执行顺序是什么?

渲染过程：
父组件挂载完成一定是等子组件都挂载完成后，才算是父组件挂载完，所以父组件的mounted在子组件mouted之后
父beforeCreate -> 父created -> 父beforeMount -> 子beforeCreate -> 子created -> 子beforeMount -> 子mounted -> 父mounted

子组件更新过程：

1. 影响到父组件： 父beforeUpdate -> 子beforeUpdate->子updated -> 父updted
2. 不影响父组件： 子beforeUpdate -> 子updated

父组件更新过程：

1. 影响到子组件： 父beforeUpdate -> 子beforeUpdate->子updated -> 父updted
2. 不影响子组件： 父beforeUpdate -> 父updated

销毁过程：
父beforeDestroy -> 子beforeDestroy -> 子destroyed -> 父destroyed

看起来很多好像很难记忆，其实只要理解了，不管是哪种情况，都一定是父组件等待子组件完成后，才会执行自己对应完成的钩子，就可以很容易记住

v-show 和 v-if 有哪些区别?

`v-if` 会在切换过程中对条件块的事件监听器和子组件进行销毁和重建，如果初始条件是false，则什么都不做，直到条件第一次为true时才开始渲染模块。
`v-show` 只是基于css进行切换，不管初始条件是什么，都会渲染。
所以， `v-if` 切换的开销更大，而 `v-show` 初始化渲染开销更大，在需要频繁切换，或者切换的部分dom很复杂时，使用 `v-show` 更合适。渲染后很少切换的则使用 `v-if` 更合适。

computed 和 watch 有什么区别?

`computed` 计算属性，是依赖其他属性的计算值，并且有缓存，只有当依赖的值变化时才会更新。
`watch` 是在监听的属性发生变化时，在回调中执行一些逻辑。
所以， `computed` 适合在模板渲染中，某个值是依赖了其他的响应式对象甚至是计算属性计算而来，而 `watch` 适合监听某个值的变化去完成一段复杂的业务逻辑。

Vue 中的 computed 是如何实现的

流程总结如下：

1. 当组件初始化的时候， `computed` 和 `data` 会分别建立各自的响应系统， `Observer` 遍历 `data` 中每个属性设置 `get/set` 数据拦截
2. 初始化 `computed` 会调用 `initComputed` 函数
    1. 注册一个 `watcher` 实例，并在内实例化一个 `Dep` 消息订阅器用作后续收集依赖（比如渲染

函数的 `watcher` 或者其他观察该计算属性变化的 `watcher` ）

2. 调用计算属性时会触发其 `Object.defineProperty` 的 `get` 访问器函数
3. 调用 `watcher.depend()` 方法向自身的消息订阅器 `dep` 的 `subs` 中添加其他属性的 `watcher` 依赖收集
4. 调用 `watcher` 的 `evaluate` 方法（进而调用 `watcher` 的 `get` 方法）让自身成为其他 `watcher` 的消息订阅器的订阅者，首先将 `watcher` 赋给 `Dep.target`，然后执行 `getter` 求值函数，当访问求值函数里面的属性（比如来自 `data`、`props` 或其他 `computed`）时，会同样触发它们的 `get` 访问器函数从而将该计算属性的 `watcher` 添加到求值函数中属性的 `watcher` 的消息订阅器 `dep` 中，当这些操作完成，最后关闭 `Dep.target` 赋为 `null` 并返回求值函数结果。

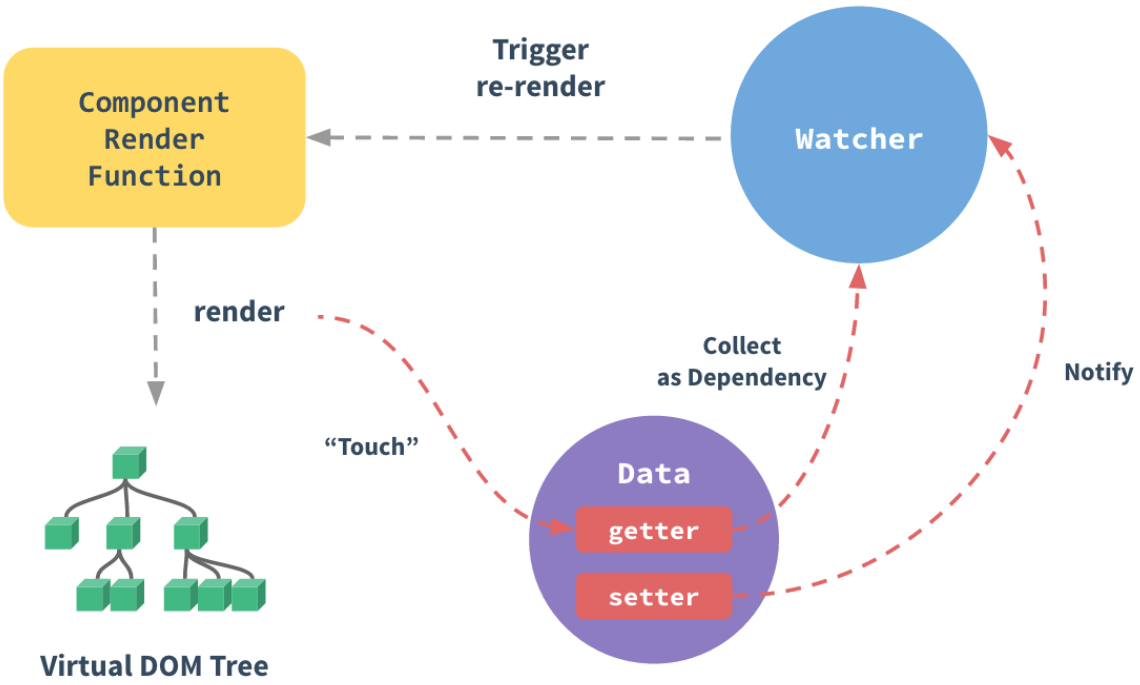
3. 当某个属性发生变化，触发 `set` 拦截函数，然后调用自身消息订阅器 `dep` 的 `notify` 方法，遍历当前 `dep` 中保存着所有订阅者 `wathcer` 的 `subs` 数组，并逐个调用 `watcher` 的 `update` 方法，完成响应更新。

Vue 中 v-html 会导致什么问题

在网站上动态渲染任意 HTML，很容易导致 XSS 攻击。所以只能在可信内容上使用 v-html，且永远不能用于用户提交的内容上。

Vue 的响应式原理

如果面试被问到这个问题，又描述不清楚，可以直接画出 Vue 官方文档的这个图，对着图来解释效果会更好。



Vue 的响应式是通过 `Object.defineProperty` 对数据进行劫持，并结合观察者模式实现。 Vue 利用 `Object.defineProperty` 创建一个 `observe` 来劫持监听所有的属性，把这些属性全部转为 `getter` 和 `setter`。Vue 中每个组件实例都会对应一个 `watcher` 实例，它会在组件渲染的过程中把使用过的数据属性通过 `getter` 收集为依赖。之后当依赖项的 `setter` 触发时，会通知 `watcher`，从而使它关联的组件重新渲染。

Object.defineProperty有哪些缺点?

1. `Object.defineProperty` 只能劫持对象的属性，而 `Proxy` 是直接代理对象
   由于 `Object.defineProperty` 只能对属性进行劫持，需要遍历对象的每个属性。而 `Proxy` 可以直接代理对象。
2. `Object.defineProperty` 对新增属性需要手动进行 `Observe`， 由于 `Object.defineProperty` 劫持的是对象的属性，所以新增属性时，需要重新遍历对象，对其新增属性再使用 `Object.defineProperty` 进行劫持。 也正是因为这个原因，使用 Vue 给 `data` 中的数组或对象新增属性时，需要使用 `vm.$set` 才能保证新增的属性也是响应式的。
3. `Proxy` 支持13种拦截操作，这是 `defineProperty` 所不具有的。
4. 新标准性能红利
   `Proxy` 作为新标准，长远来看，JS引擎会继续优化 `Proxy` ，但 `getter` 和 `setter` 基本不会再有针对性优化。
5. `Proxy` 兼容性差 目前并没有一个完整支持 `Proxy` 所有拦截方法的Polyfill方案

Vue2.0中如何检测数组变化?

Vue 的 `Observer` 对数组做了单独的处理，对数组的方法进行编译，并赋值给数组属性的 `__proto__` 属性上，因为原型链的机制，找到对应的方法就不会继续往上找了。编译方法中会对一些会增加索引的方法（`push`， `unshift`， `splice`）进行手动 observe。

nextTick是做什么用的，其原理是什么?

能回答清楚这道问题的前提，是清楚 EventLoop 过程。
在下次 DOM 更新循环结束后执行延迟回调，在修改数据之后立即使用 `nextTick` 来获取更新后的 DOM。
`nextTick` 对于 micro task 的实现，会先检测是否支持 `Promise`，不支持的话，直接指向 macro task，而 macro task 的实现，优先检测是否支持 `setImmediate`（高版本IE和Etage支持），不支持的再去检测是否支持 MessageChannel，如果仍不支持，最终降级为 `setTimeout` 0；
默认的情况，会先以 `micro task` 方式执行，因为 micro task 可以在一次 tick 中全部执行完毕，在一些有重绘和动画的场景有更好的性能。
但是由于 micro task 优先级较高，在某些情况下，可能会在事件冒泡过程中触发，导致一些问题，所以有些地方会强制使用 macro task （如 `v-on`）。

注意：之所以将 `nextTick` 的回调函数放入到数组中一次性执行，而不是直接在 `nextTick` 中执行回调函数，是为了保证在同一个tick内多次执行了 `nextTcik`，不会开启多个异步任务，而是把这些异步任务都压成一个同步任务，在下一个tick内执行完毕。

Vue 的模板编译原理

vue模板的编译过程分为3个阶段：

第一步：解析
将模板字符串解析生成 AST，生成的AST 元素节点总共有 3 种类型，1 为普通元素， 2 为表达式，3为纯文本。

第二步：优化语法树
Vue 模板中并不是所有数据都是响应式的，有很多数据是首次渲染后就永远不会变化的，那么这部分数据生成的 DOM 也不会变化，我们可以在 patch 的过程跳过对他们的比对。

此阶段会深度遍历生成的 AST 树，检测它的每一颗子树是不是静态节点，如果是静态节点则它们生成 DOM 永远不需要改变，这对运行时对模板的更新起到极大的优化作用。

1. 生成代码

```
const code = generate(ast, options)
```

通过 `generate` 方法，将ast生成 `render` 函数。

v-for 中 key 的作用是什么?

`key` 是给每个 `vnode` 指定的唯一 `id`，在同级的 `vnode` diff 过程中，可以根据 `key` 快速的对比，来判断是否为相同节点，并且利用 `key` 的唯一性可以生成 `map` 来更快的获取相应的节点。
另外指定 `key` 后，就不再采用“就地复用”策略了，可以保证渲染的准确性。

为什么 v-for 和 v-if 不建议用在一起

当 `v-for` 和 `v-if` 处于同一个节点时，`v-for` 的优先级比 `v-if` 更高，这意味着 `v-if` 将分别重复运行于每个 `v-for` 循环中。如果要遍历的数组很大，而真正要展示的数据很少时，这将造成很大的性能浪费。
这种场景建议使用 `computed`，先对数据进行过滤。

vue-router hash 模式和 history 模式有什么区别?

区别：

1. url 展示上，hash 模式有“#”，history 模式没有

2. 刷新页面时，hash 模式可以正常加载到 hash 值对应的页面，而 history 没有处理的话，会返回 404，一般需要后端将所有页面都配置重定向到首页路由。
3. 兼容性。hash 可以支持低版本浏览器和 IE。

vue-router hash 模式和 history 模式是如何实现的?

- hash 模式：
  `#`后面 hash 值的变化，不会导致浏览器向服务器发出请求，浏览器不发出请求，就不会刷新页面。同时通过监听 hashchange 事件可以知道 hash 发生了哪些变化，然后根据 hash 变化来实现更新页面部分内容的操作。
- history 模式：
  history 模式的实现，主要是 HTML5 标准发布的两个 API，`pushState` 和 `replaceState`，这两个 API 可以在改变 url，但是不会发送请求。这样就可以监听 url 变化来实现更新页面部分内容的操作。

vue 中组件 data 为什么是 return 一个对象的函数，而不是直接是个对象?

当data定义为对象后，这就表示所有的组件实例共用了一份data数据，因此，无论在哪个组件实例中修改了data,都会影响到所有的组件实例。组件中的data写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一份新的data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份data，就会造成一个变了全都会变的结果。

MVVM 的实现原理

1. 响应式：vue如何监听data的属性变化
2. 模板解析：vue的模板是如何被解析的
3. 渲染：vue模板是如何被渲染成HTML的

vue3.0 相对于 vue2.x 有哪些变化?

- 监测机制的改变（Object.defineProperty —> Proxy）
- 模板
- 对象式的组件声明方式 （class）
- 使用ts
- 其它方面的更改：支持自定义渲染器、 支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组建内容）组件、基于 treeshaking 优化，提供了更多的内置功能

那你能讲一讲MVVM吗?

MVVM是`Model-View-ViewModel`缩写，也就是把`MVC`中的`Controller`演变成`ViewModel`。Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据。

你知道Vue3.x响应式数据原理吗?

Vue3.x改用`Proxy`替代Object.defineProperty。因为Proxy可以直接监听对象和数组的变化，并且有多达13种拦截方法。并且作为新标准将受到浏览器厂商重点持续的性能优化。

> Proxy只会代理对象的第一层，那么Vue3又是怎样处理这个问题的呢?

判断当前Reflect.get的返回值是否为Object，如果是则再通过`reactive`方法做代理， 这样就实现了深度观测。

> 监测数组的时候可能触发多次get/set，那么如何防止触发多次呢?

我们可以判断key是否为当前被代理对象target自身属性，也可以判断旧值与新值是否相等，只有满足以上两个条件之一时，才有可能执行trigger。

Vue2.x和Vue3.x渲染器的diff算法分别说一下

简单来说，diff算法有以下过程

- 同级比较，再比较子节点
- 先判断一方有子节点一方没有子节点的情况(如果新的children没有子节点，将旧的子节点移除)
- 比较都有子节点的情况(核心diff)
- 递归比较子节点

正常Diff两个树的时间复杂度是`O(n^3)`，但实际情况下我们很少会进行`跨层级的移动DOM`，所以Vue将Diff进行了优化，从`O(n^3) -> O(n)`，只有当新旧children都为多个子节点时才需要用核心的Diff算法进行同层级比较。

Vue2的核心Diff算法采用了`双端比较`的算法，同时从新旧children的两端开始进行比较，借助key值找到可复用的节点，再进行相关操作。相比React的Diff算法，同样情况下可以减少移动节点次数，减少不必要的性能损耗，更加的优雅。

Vue3.x借鉴了 ivi算法和 inferno算法

在创建VNode时就确定其类型，以及在`mount/patch`的过程中采用`位运算`来判断一个VNode的类型，在这个基础之上再配合核心的Diff算法，使得性能上较Vue2.x有了提升。(实际的实现可以结合Vue3.x源码看。)

你都做过哪些Vue的性能优化?

编码阶段

- 尽量减少data中的数据，data中的数据都会增加getter和setter，会收集对应的watcher
- v-if和v-for不能连用
- 如果需要使用v-for给每项元素绑定事件时使用事件代理
- SPA 页面采用keep-alive缓存组件
- 在更多的情况下，使用v-if替代v-show
- key保证唯一
- 使用路由懒加载、异步组件
- 防抖、节流
- 第三方模块按需导入
- 长列表滚动到可视区域动态加载
- 图片懒加载

SEO优化

- 预渲染
- 服务端渲染SSR

打包优化

- 压缩代码
- Tree Shaking/Scope Hoisting
- 使用cdn加载第三方模块
- 多线程打包happypack
- splitChunks抽离公共文件
- sourceMap优化

用户体验

- 骨架屏
- PWA

还可以使用缓存(客户端缓存、服务端缓存)优化、服务端开启gzip压缩等。

React篇

你的技术栈主要是react，那你说说你用react有什么坑点?

1、JSX做表达式判断时候，需要强转为boolean类型，如：

```
render() {
  const b = 0;
  return <div>
    {
      !!b && <div>这是一段文本</div>
    }
  </div>
}
```

如果不使用 !!b 进行强转数据类型，会在页面里面输出 0。

2、尽量不要在 componentWillRevicеProps 里使用 setState，如果一定要使用，那么需要判断结束条件，不然会出现无限重渲染，导致页面崩溃。(实际不是componentWillRevicеProps会无限重渲染，而是componentDidUpdate)

3、给组件添加ref时候，尽量不要使用匿名函数，因为当组件更新的时候，匿名函数会被当做新的prop处理，让ref属性接受到新函数的时候，react内部会先清空ref，也就是会以null为回调参数先执行一次ref这个props，然后在以该组件的实例执行一次ref，所以用匿名函数做ref的时候，有的时候去ref赋值后的属性会取到null

4、遍历子节点的时候，不要用 index 作为组件的 key 进行传入。

怎么去设计一个组件封装

- 组件封装的目的是为了重用，提高开发效率和代码质量
- 低耦合，单一职责，可复用性，可维护性

react 的虚拟dom是怎么实现的

首先说说为什么要使用Virturl DOM，因为操作真实DOM的耗费的性能代价太高，所以react内部使用js实现了一套dom结构，在每次操作在和真实dom之前，使用实现好的diff算法，对虚拟dom进行比较，递归找出有变化的dom节点，然后对其进行更新操作。为了实现虚拟DOM，我们需要把每一种节点类型抽象成对象，每一种节点类型有自己的属性，也就是prop，每次进行diff的时候，react会先比较该节点类型，假如节点类型不一样，那么react会直接删除该节点，然后直接创建新的节点插入到其中，假如节点类型一样，那么会比较prop是否有更新，假如有prop不一样，那么react会判定该节点有更新，那么重渲染该节点，然后在对其子节点进行比较，一层一层往下，直到没有子节点。

react hooks 原理是什么?

hooks 是用闭包实现的，因为纯函数不能记住状态，只能通过闭包来实现。

useState 中的状态是怎么存储的?

通过单向链表，fiber tree 就是一个单向链表的树形结构。

如何遍历一个dom树

```
function traversal(node) {
    //对node的处理
    if (node && node.nodeType === 1) {
        console.log(node.tagName);
    }
    var i = 0,
        childNodes = node.childNodes,
        item;
    for (; i < childNodes.length; i++) {
        item = childNodes[i];
        if (item.nodeType === 1) {
            //递归先序遍历子节点
            traversal(item);
        }
    }
}
```

数据双向绑定单向绑定优缺点

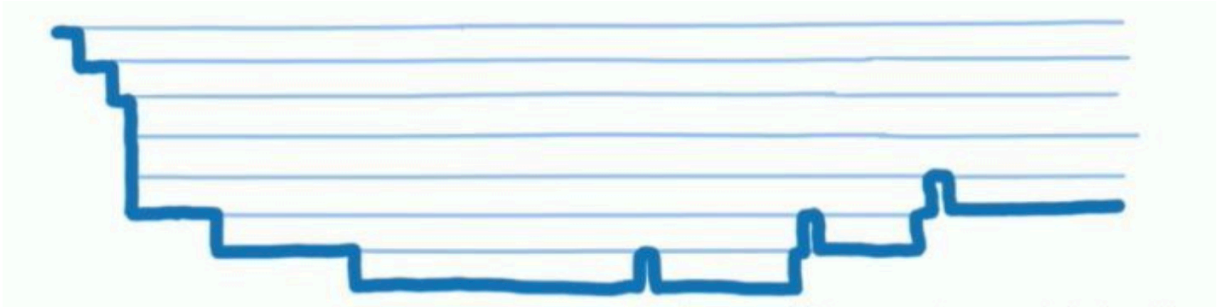
- 双向绑定是自动管理状态的，对处理有用户交互的场景非常合适，代码量少，当项目越来越大的时候，调试也变得越来越复杂，难以跟踪问题
- 单向绑定是无状态的, 程序调试相对容易, 可以避免程序复杂度上升时产生的各种问题, 当然写代码时就没有双向绑定那么爽了

React fiber 的理解和原理

- 理解

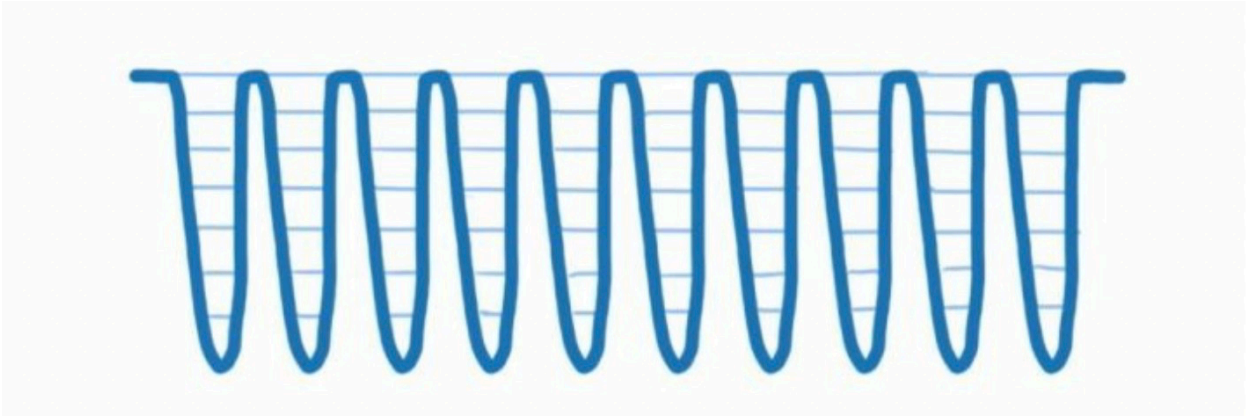
React16 以前

React16 以前，对virtural dom的更新和渲染是同步的。就是当一次更新或者一次加载开始以后，diff virtual dom并且渲染的过程是一口气完成的。如果组件层级比较深，相应的堆栈也会很深，长时间占用浏览器主线程，一些类似用户输入、鼠标滚动等操作得不到响应。借Lin的两张图，视频 A Cartoon Intro to Fiber - React Conf 2017。



React16 Fiber Reconciler

React16 用了分片的方式解决上面的问题。
就是把一个任务分成很多小片，当分配给这个小片的时间用尽的时候，就检查任务列表中有没有新的、优先级更高的任务，有就做这个新任务，没有就继续做原来的任务。这种方式被叫做异步渲染(Async Rendering)。



- 一些原理

Fiber就是通过对象记录组件上需要做或者已经完成的更新，一个组件可以对应多个Fiber。

在render函数中创建的React Element树在第一次渲染的时候会创建一颗结构一模一样的Fiber节点树。不同的React Element类型对应不同的Fiber节点类型。一个React Element的工作就由它对应的Fiber节点来负责。

一个React Element可以对应不止一个Fiber，因为Fiber在update的时候，会从原来的Fiber（我们称为current）clone出一个新的Fiber（我们称为alternate）。两个Fiber diff出的变化（side effect）记录在alternate上。所以一个组件在更新时最多会有两个Fiber与其对应，在更新结束后alternate会取代之前的current的成为新的current节点。

其次，Fiber的基本规则：协调阶段 提交阶段
更新任务分成两个阶段，Reconciliation Phase和Commit Phase。Reconciliation Phase的任务干的事情是，找出要做的更新工作（Diff Fiber Tree），就是一个计算阶段，计算结果可以被缓存，也就可以被打断；Commmit Phase 需要提交所有更新并渲染，为了防止页面抖动，被设置为不能被打断。

PS: componentWillMount componentWillReceiveProps componentWillUpdate 几个生命周期方法，在Reconciliation Phase被调用，有被打断的可能（时间用尽等情况），所以可能被多次调用。其实shouldComponentUpdate 也可能被多次调用，只是它只返回true或者false，没有副作用，可以暂时忽略。

解释 React 中 render() 的目的

每个React组件强制要求必须有一个 **render()**。它返回一个 React 元素，是原生 DOM 组件的表示。如果需要渲染多个 HTML 元素，则必须将它们组合在一个封闭标记内，例如 `、`、``` 等。此函数必须保持纯净，即必须每次调用时都返回相同的结果。

调用 setState 之后发生了什么?

- 在代码中调用 setState 函数之后，React **会将传入的参数对象与组件当前的状态合并**，然后触发所谓的调和过程。
- 经过调和过程，React会以相对高效的方式根据新的状态构建React元素树并且着手重新渲染整个 UI 界面。
- 在 React 得到元素树之后，React **会自动计算出新的树与老树的节点差异**，然后根据差异对界面**进行最小化重渲染**。
- 在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了**按需更新**，而不是全部重新渲染。

触发多次setstate，那么render会执行几次?

- 多次setState会**合并**为一次render，因为setState并不会立即改变state的值，而是将其放到一个**任务队列**里，最终将多个setState合并，一次性更新页面。
- 所以我们可以在代码里多次调用setState，每次只需要关注当前修改的字段即可

react中如何对state中的数据进行修改？setState为什么是一个异步的?

- 修改数据通过this.setState(参数1,参数2)
- this.setState是一个异步函数
  - 参数1 : 是需要修改的数据是一个对象
  - 参数2 : 是一个回调函数，可以用来验证数据是否修改成功，同时可以获取到数据更新后的DOM结构等同于componentDidMount
- this.setState中的第一个参数除了可以写成一个对象以外，还可以写成一个函数 ！，函数中第一个值为prevState 第二个值为prePprops　this.setState((prevState,prop)=>({}))

为什么建议传递给 setState的参数是一个callback而不是一个对象?

- 因为this.props 和this.state的更新可能是异步的，不能依赖它们的值去计算下一个state

为什么setState是一个异步的?

- 当批量执行state的时候可以让DOM渲染的更快,也就是说多个setstate在执行的过程中还需要被合并

原生事件和React事件的区别?

- `React` 事件使用驼峰命名，而不是全部小写。
- 通过 `JSX` , 你传递一个函数作为事件处理程序，而不是一个字符串。
- 在 `React` 中你不能通过返回 `false` 来阻止默认行为。必须明确调用 `preventDefault` 。

React的合成事件是什么?

`React` 根据 `W3C` 规范定义了每个事件处理函数的参数，即合成事件。

事件处理程序将传递 `SyntheticEvent` 的实例，这是一个跨浏览器原生事件包装器。它具有与浏览器原生事件相同的接口，包括 `stopPropagation()` 和 `preventDefault()`，在所有浏览器中他们工作方式都相同。

`React`合成的 `SyntheticEvent`采用了事件池，这样做可以大大节省内存，而不会频繁的创建和销毁事件对象。

另外，不管在什么浏览器环境下，浏览器会将该事件类型统一创建为合成事件，从而达到了浏览器兼容的目的。

什么是高阶组件（HOC）

高阶组件是重用组件逻辑的高级方法，是一种源于 React 的组件模式。 HOC 是自定义组件，在它之内包含另一个组件。它们可以接受子组件提供的任何动态，但不会修改或复制其输入组件中的任何行为。你可以认为 HOC 是“纯（Pure）”组件。

你能用HOC做什么?

HOC可用于许多任务，例如：

- 代码重用，逻辑和引导抽象
- 渲染劫持
- 状态抽象和控制
- Props 控制