## node.js的优缺点，针对缺点的解决方案

**优点:**

1. 设计思想方面:node.js采用事件驱动，异步编程的思想来设计，利用了javascript的匿名函数和闭包特性，使得node.js非常适合做网络服务应用。

2. 学习成本方面:node.js是以javascript为基础的服务器平台，使得node.js像javascript一样简单易学，容易上手，特别是从事前端开发的人员可以很快上手服务端的开发。

3. 运行效率方面:node.js非阻塞的I/O处理使得自身可以以相对较低的系统资源占有产生高性能和出众的负载能力，这也使得node.js非常适合依赖于其他I/O资源的中间层服务。

4. 请求处理方面:node.js简单高效，适合最为数据密集型分布式的部署环境下的实时应用系统的解决方案，适合情景，在响应客户端之前，预计会有很大的流量，但服务器端的处理逻辑不一定很多。

**缺点:**

1. 可靠性低

2. node.js是单进程，单线程，运行在单核CPU上，不能充分利用多核CPU，一旦这个进程失去响应，那么整个web服务都将崩溃

**node.js的网络服务器支持多进程的方案**

1. 开启多个进程，并绑定到不同的端口监听，使用反向代理服务器(Nginx)做负载均衡。

使用Nginx的优点:

1. 能很好的完成请求的负载均衡任务

2. 能在Nginx分配请求的时候做检查操作

缺点:

需要在请求的处理中加入中间层.

2. 多个进程绑定到一个端口监听，通过node.js可以在不同进程中发送文件句柄来进行通信。

3. 利用一个进程负责监听，接受请求，之后将请求均匀的分配给其他子进程去处理。(child\_process)node.js中有两个模块可以解决多进程的问题:

multi-node, cluster, 还有一个PM2(第三方模块)

缺点:更新不及时，体系复杂庞大，绑定了实际不需要的其他功能，一旦出现问题比较难以解决。

## 2. 同步和异步什么，阻塞和非阻塞

**同步:** 当一个调用发出后，调用不会立即返回直到得到调用结果。

**异步:** 当一个调用发出后，调用者线程不需要等待调用结果的返回就可以继续执行后续的操作，实际处理调用的操作会以状态或者消息来通知调用者，或者通过已定的回调函数来处理调用结果。

## 3. node.js的事件循环机制

node.js程序是由事件循环开始，到事件循环结束，所有的逻辑都是事件的回调函数。程序的入口是事件循环的第一个事件的回调函数。事件触发执行回调，再回到事件循环中，如果回调中调用了异步I/O操作，如文件的读写，网络通信，数据库操作等，操作的处理结果会通过回调处理触发新的事件，这些事件会进入事件循环的任务队列中等待进入下次事件循环执行队列并执行，之后重复上述过程知道程序结束。

## 4. node.js常用的全局变量

console, process, Buffer, global, exports, \_\_filename(当前绝对路径下文件名), \_\_dirname(当前文件绝对路径目录)

## 5. node.js的核心模块

http, fs(文件系统), events(事件), Stream(流), URL, Query String, Path, child\_process

## 6. props和state的区别

props是属性，state是状态，两者都是React组件对象，都是组件对象存储和传递数据的载体。

(定义) props 大多数组件在定义时就可以通过各种自定义参数来实现组件的定制。对外提供的这些参数就成为props

(定义) state 作为用户交互设计产生的反馈通常都是通过state来进行数据的改变，界面的刷新。

(作用) props 一般用于父组件向子组件通信，在组件之间通信使用。

(作用) state 一般用于组件内部的状态维护，更新组件内部的数据，状态，更新子组件的props等。

**区别:**

props用于父组件向子组件传递数据，在组件之间进行通信。

state用于组件内的状态维护，更新组件内的数据，状态以及子组件的props属性等。

## 7. node.js的整体架构

node.js在整体上分为三部分:应用app >> V8和nodejs内置架构 >> 系统

在上述的中间环节又分为三层:

1. javascript实现的核心模块

2. c++ 绑定(socket, http, fs ,..)

3. V8(node的引擎), async I/O (libuv), Event Loop (libuv)

## 8. React组件的生命周期

React组件的生命周期在大的方面分为三部分:mount, update, unmount

1. mount部分:组件第一次渲染过程: constructor (组件的构造函数,), getInitialState (设置初始状态属性), componentWillMount, mounting, componentDidMount

2. update部分:改变state后组件更新过程: componentReciveProps, shouldComponentUpdate(是否执行本次更新true/false), componentWillUpdate, updating, componentDidUpdate

3. unmount部分:移除组件过程: componentWillUnmount,unmounting

## 9. React的实现原理

React是以组件的形式构建页面上的DOM，并将组件构建为一种状态机，组件的形式是由状态属性决定，状态的属性是组件形式的数据表现。

React组件在加载时，会根据初始的状态属性通过render方法先构建组件的虚拟DOM，再将虚拟DOM渲染到页面中，所以首次加载页面是会慢一些；当行为改变组件状态属性state时，React会生成当前状态对应下的虚拟DOM，并和前一次的虚拟DOM进行diff算法，得到需要改变的DOM，最后将这部分改变的DOM更新到实际的DOM中。

## 10. Redux的实现原理

Redux中主要由三部分组成：Action, Reducer, Store.

Action用来传递操作state的信息到store，是store的唯一来源。

Reducer用来处理action,通过传入旧的state和action来指明如何更新state。

action用来描述发发生了什么，reducer根据action更新state, Store就是把这几样东西连接到一起的对象。

首先用户发出Action.然后Store自动调用Reducer,并且传入两个参数:当前State和收到的Action。Reducer会返回新的State. State一旦有变化，就会重新渲染组件。

