CVTE电话一面 3.24

1、简单的自我介绍

您好，我是蔡文钦，目前是广东工业大学网络工程专业的一名大三学生，在校主修了计算机网络，数据结构，操作系统等课程，在大二时接触到了前端，发现对前端很感兴趣，于是我学习了html，css，js以及vue框架以及浏览器的相关知识，并选择以后从事前端方向的工作
目前我写了两个项目，其一是网页版的音乐播放器，使用vue框架仿照网易pc云客户端进行组件化开发，完成了大部分功能，如账号登录，音乐，MV播放，歌词，评论显示等，并通过懒加载的技术进行优化图片加载，其二是基于vue框架的移动端二手交易平台，我负责了首页，商品列表，购物车，商品详情等页面的开发，并使用路由懒加载，分段渲染等技术进行加载优化。
虽然目前经验尚浅水平有限，但我一直在努力尝试提高，为了能够进一步提高技术与增长经验，我想找一个发展环境良好的公司实习来提升自己的能力，也希望字节跳动能给我这个机会，以上就是我的自我介绍，谢谢

2、之前有做过什么项目吗？是否上线

目前我完成了两个项目，第一个是网页版的音乐播放器，我使用了Vue框架仿照网易pc云客户端进行组件化开发，完成了大部分功能，如账号登录，音乐，MV播放，歌词，评论显示等，并通过懒加载的技术进行优化图片加载，第一个是基于Vue框架的移动端二手交易平台，我负责了首页，商品列表，购物车，商品详情等页面的开发，并使用路由懒加载，分段渲染等技术进行加载优化。
目前还没有上线

3、项目的难点是什么？如何解决

音乐播放器：

二手交易平台：

难点：vue详情页返回列表页缓存数据和浏览位置、其他页面进入列表页刷洗数据
使用keeplive缓存组件，然后在列表页的beforeRouteLeave钩子中缓存数据
在列表的activated中判断要使用缓存数据还是重新刷新
但后来发现在列表页切换到详情页时回退是滚动条保留在详情页的位置，而不是列表页的位置，我了解到如果是hastory模式的话可以使用vue-router自带的scrollBehavior实现滚动条记忆，而且需要浏览器支持hastory.putstate,但hash模式下没有自带方法，而我们当时使用的正好是hash模式，所有后来我就自己写了一个插件去兼容history和hash模式
定义了三个方法，save保存滚动条位置，get恢复滚动条，goTop设置滚动条到顶部，
在beforeEach的钩子中调用save保存离开页面的滚动条位置，在mounted中调用get函数恢复滚动条的位置,刚开始也在beforeEach中使用setTimeOut去恢复,但时间很难把握，太早有偏差，太晚有会被看到页面的跳动，所以改成在mounted中调用，这个时候当前页面组件都挂载到DOM树了，而官方文档说“mounted不会承诺所有的子组件也一起被挂载”,于是我在$nextTick中去调用get（这样其实在created中也可以调用，不过放mouted中更合时宜），又因为有写页面不需要记住滚动条位置，所以我定义了一个getTop方法来让滚动条恢复到顶部

4.1、hash模式和history模式的区别，history的原理

1、hash模式

：只能改变 # 后面的 url 片段即 hash 值。hash 值的变化，并**不会导致浏览器向服务器发出请求**，浏览器不发出请求，也就不会刷新页面。每次 hash 值的变化，会**触发** `hashchange` 这个事件，通过这个事件我们就可以知道 hash 值发生了哪些变化。然后我们便可以**监听** `hashchange` **来实现更新页面部分内容的操作**：

hash 模式的工作原理是 hashchange 事件，可以在 window 监听 hash 的变化。

```
<div id="test" style="height: 500px;width: 500px;margin: 0 auto"></div>

<script>
    window.onhashchange = function (event) {
      console.log(event) // HashChangeEvent {..., newURL: "...test.html#red",
oldURL: "...test.html", ...}
      console.log(location) // location {..., hash: "#red", ...}
      let hash = location.hash.slice(1); // red
      document.body.style.color = hash;
      document.getElementById('test').style.backgroundColor = hash
    }
</script>
```

在 url 后面随便添加一个 #xx 会触发 onhashchange 事件。打印 event，里边有两个属 性 newURL 和 oldURL。可以通过模拟改变 hash 的值，动态改变页面数据。

2、history模式：

通过pushState api,可以设置与当URL同源的任意URL，即和正常访问后端的URL一样，但如果进行刷新，而后端没有配置相应的URL路径，则会报404

后端可以配置无论访问什么路径都返回index（主页面）然后由前端进行处理

原理：

通过监听DOMContentLoaded事件获取localtion.pathname('/'后面的内容)

当用户进行切换路由时，利用history的api pushState 来改变url，然后再进行页面渲染

使用window.onpopstate 去监听用户对浏览器的前进后退操作，然后渲染页面

4.2、$nextTick原理

nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 nextTick，则可以在回调中获取更新后的 DOM
功能：延迟回调

源码：判断是否支持Promise,支持则使用promise

不支持则使用MutationObserver 它会在指定的DOM发生变化时被调用

如果不支持 MutationObserver 的话就用 setImmediate(只有最新的IE和node.js 10.0支持），可替代 setTimeout（fn,0）

如果都不支持就使用setTimeout

为何nextTick 要优先使用`Promise`和`MutationObserver`： 因为他俩属于微任务，会在执行栈空闲的时候立即执行，它的响应速度相比setTimeout会更快，因为无需等渲染。

5、css盒子模型

盒模型包含了元素内容（content）、内边距（padding）、边框（border）、外边距（margin）几个要素

有两种盒模型：标准模式和IE模型，他们计算宽高的方式不同

在标准模型中：宽高等于内容的宽高，可以通过设置box-sizing:content-box设置

在IE模型中：宽高等于内容宽高+padding+border

- js如何获得宽高：

dom.style.width/height:获取不到style和link的样式

dom.getBoundingClientRect().width/height:大多数浏览器和IE9以上支持

- 外边距重叠：普通文档流中的块元素的垂直margin会合并为较大的一个

6、什么是BFC

- BFC：块级格式化上下文

  一个独立的块级渲染区域，该区域拥有一套渲染规格来约束块级盒子的布局，且与区域外部无关。

- 特性(原理)：
  1. BFC元素垂直方向的外边距会发生重叠。但如果两个元素不属于同一个BFC，则外边距不会发生重叠（处理垂直外边距重叠问题）
  2. BFC的区域不会与浮动元素的布局重叠。（浏览器将会给BFC创建隐式的外边距来阻止它和浮动元素的外边距的叠加）
  3. 独立的容器，内外元素互不影响
  4. 计算BFC高度的时候，浮动元素也会参与计算
- 如何创建BFC：

  1、overflow不为visible

  2、float不为none

  3、position的值不为static或relative

  4、display不为inline-blocks,table,table-cell,table-caption,flex,inline-flex;

7、怎么清除浮动

浮动：当元素浮动时，元素就会脱离文档流，像一条小船一样浮动在文档之上，没有清除浮动的元素感受不到其存在，（浮动元素会形成BFC）

如何浮动：设置float

特点：

1.浮动的元素，讲向左或者向右浮动，浮动到包围元素的边上，或者上一个浮动元素的边上为止。

2.浮动的元素，不再占用空间，且浮动元素的层级要高于普通元素。

3.浮动的元素，一定是块元素，不管之前是什么元素。

4.如果浮动的元素没有指定宽度的话，浮动后会尽可能变窄，因此浮动元素要指定宽和高。

5.一行的多个元素，要浮动大家一起浮动。

6、子元素浮动，父元素未指定高度，则父元素会产生高度坍塌

清除浮动方法：

1、添加一个空标签，使用clear：both/left/right，以清除后面元素受浮动影响

2、父级div定义 overflow: auto

3、使用伪类选择器：after（作用在浮动元素的父亲）

```
.outer:after {clear:both;content:'.';display:block;width: 0;height:
0;visibility:hidden;}
```

8、什么是闭包，有什么优缺点

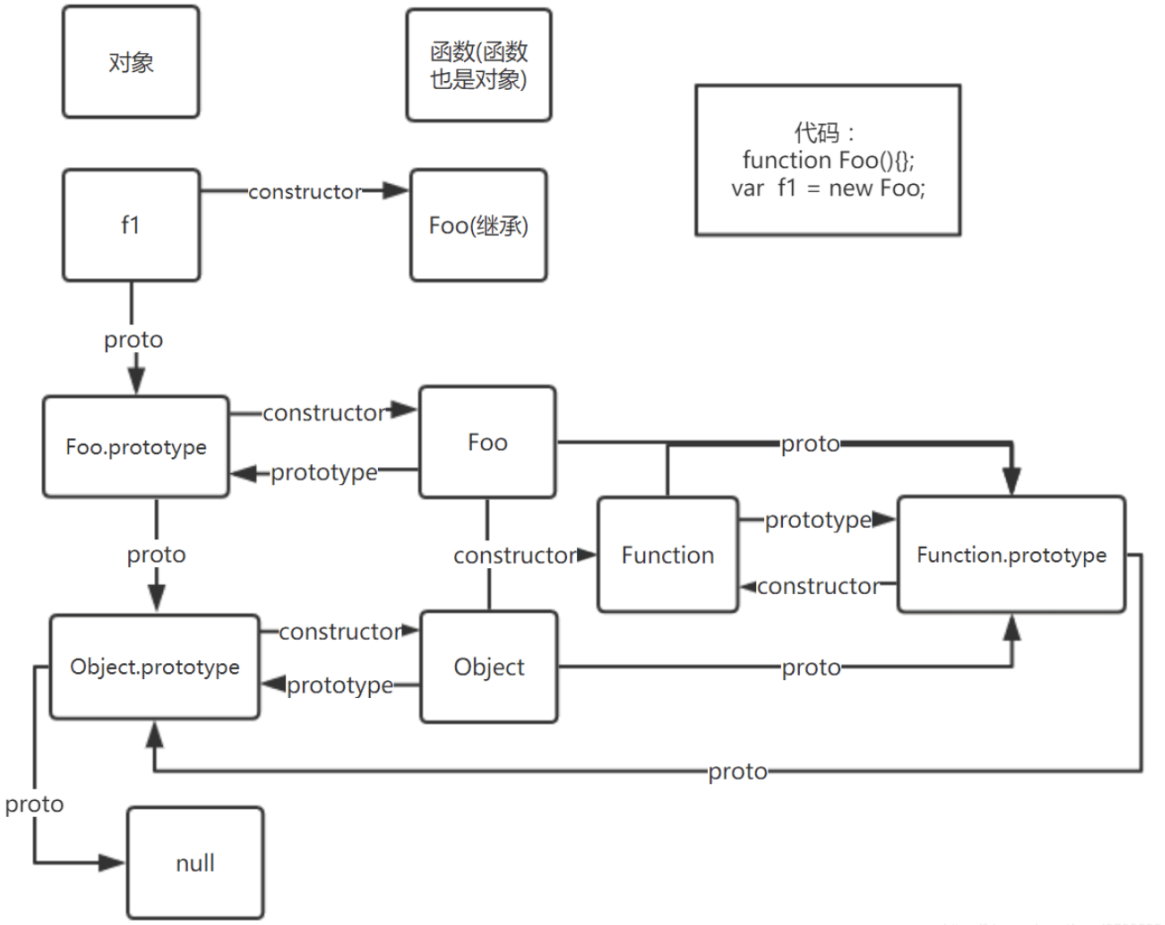
思路：
闭包概念 -> 闭包代码解析 -> 闭包本质 (AO,GO结合执行器上下文)->垃圾回收机制->闭包优缺点等
-闭包：能够读取其他函数内部变量的函数（能访问自由变量的函数）
-优点：
1、可以读取函数内部的变量，
2、让这些变量的值始终保持在内存中，并不会在一个函数调用后被清除
-缺点：
1、闭包会使得函数中的变量都被保存在内存中，消耗很大，退出函数时要将用不到的变量清除
-原理：
首先要了解什么是作用域：
作用域：一个变量的作用范围
1、全局作用域:
(1)、在页面打开时产生，关闭时销毁
(2)、在scrpt标签中编写的变量和函数作用域为全局，任何地方可以访问到
(3)、windows为全局对象，代表一个浏览器窗口
(4)、全局作用域中声明的函数作用域为全局作用域
2、函数作用域：
(1)、调用函数时产生，执行结束销毁
(2)、每调用一次会产生一个作用域
(3)、函数作用域可以访问全局作用域，函数外部无法访问函数作用域内的变量
(4)、在访问变量时，在自身作用中查找,找不到则到上一级作用域中寻找
作用域的深层次理解：
执行器的上下文：
-当函数执行前期会创建一个执行期上下文的内部对象AO（函数作用域）
-这个内部的对象是预编译时创建的，函数被调用时会先进行预编译
-在全局代码的执行前期会创建一个执行期的上下文对象GO
再理解什么是作用域链：
作用域链：保存在隐式的属性scope中,用于给js引擎访问，里面存储着作用域链，是AO和GO的集合
当一个函数内部的变量被其他函数访问是会产生一个作用域链，连接着函数与变量，只有当变量没有被任何函数访问是才会被回收

9、原型链

思路：原型，构造函数，实例的关系->原型链

概念：我们都知道构造函数都有一个原型对象（prototype），而每个原型对象都有一个指向构造函数的指针(constructor)，而每个实例都有一个指向构造函数原型对象的指针（*__proto__*）,*而如果原型对象也是另一个原型对象的实例，那么同样也会有指向原型对象的原型的指针，这样层层递进就构成了实例和原型的链条，也就是原型链了，*

原型链最终会指向对象的原型（Object.prototype）,而对象的原型则指向null，但Object本身也都是函数的，所以他上面都会有一个指针（*__proto__*）指向函数的原型（Function()）,而函数的原型最终也会指向对象的原型，一般情况下，构造函数的隐式原型和显式原型并不相等。但有一个例外，Function 的隐式原型和显式原型相等。`Function.prototype = Function.__proto__`



作用：是实现继承的主要方法

继承的基本思想：利用原型让一个引用类型继承另一个引用类型的属性和方法（使一个引用类型的原型指针指向另一个引用类型）

10、apply(),call()和bind()的区别，如何实现一个bind()

apply和call共同点：改变函数的执行时的上下文，能劫持另外一个对象的方法，继承另外一个对象的属性

区别：apply：，第二个参数是一个数组或类数组 Function.apply(obj,[param1,param2])（什么是类数组：**类数组无法使用 forEach、splice、push 等数组原型链上的方法**，毕竟它不是真正的数组。）

call：与apply一样，但参数是多 个 Function.call(obj,param1,param2)

call的用法：

```
//对象的继承
function superClass () {
    this.a = 1;
    this.print = function () {
        console.log(this.a);
    }
}
function subClass () {
    superClass.call(this);
    this.print();
}
subClass()
//借用方法
let domNodes = Array.prototype.slice.call(document.getElementsByTagName("*"));
```

call的实现：

```
// 第三版
//1、将函数设为对象的属性
//2 、执行该函数
//3、删除该函数
Function.prototype.call2 = function (context) {
    //判断是否传入第一个参数
    var context = context || window;
    //获取调用call的函数，即this，将其设置为传入对象的属性
    context.fn = this;
    //处理arguments，即第二个到最后一个参数
    var args = [];
    for(var i = 1, len = arguments.length; i < len; i++) {
        args.push('arguments[' + i + ']');
    }
    //ES3用eval执行执行函数，并获取返回的结果
    //这里 args 会自动调用 Array.toString() 这个方法
    var result = eval('context.fn(' + args +')');
    //或者直接ES6方法 ： var result =context(...args) ;

    delete context.fn  // 删除对象上的属性
    return result; // 返回结果
}
```

apply的用法：

```
//获取数组最大值
var max = Math.max.apply(null, array);
//合并两个数组
Array.prototype.push.apply(arr1, arr2);
```

apply的实现：

```
Function.prototype.apply = function (context, arr) {
    var context = Object(context) || window;
    context.fn = this;

    var result;
    if (!arr) {
        result = context.fn();
    }
    else {
        var args = [];
        for (var i = 0, len = arr.length; i < len; i++) {
            args.push('arr[' + i + ']');
        }
        result = eval('context.fn(' + args + ')')
    }

    delete context.fn
    return result;
```

```
}
```

bind():创建一个新函数，在调用时设置this为提供的值

与call(),apply()的区别：bind返回的是函数，需要稍后调用才会执行

（如果第一个参数为null则this指向window）

实现一个bind（）：

1、使用call或apply改变this的指向

在bind中返回一个函数，在该函数中用call或者apply使传入的目标函数的this指向当前环境，当调用该函数时，再返回目标函数

```
// 第一版
Function.prototype.bind2 = function (context) {
    var self = this;
    return function () {
        return self.apply(context);
    }

}
```

2、传参的模拟：使用arguments处理实现传入多个参数

```
// 第二版
Function.prototype.bind2 = function (context) {
    var self = this;
    // 获取bind2函数从第二个参数到最后一个参数
    var args = Array.prototype.slice.call(arguments, 1);
    return function () {
        // 这个时候的arguments是指bind返回的函数传入的参数
        var bindArgs = Array.prototype.slice.call(arguments);
        return self.apply(context, args.concat(bindArgs));
    }

}
```

3、构造函数效果的实现：修改返回函数的原型

bind有一个特点：一个绑定函数也能使用new操作符创建对象：这种行为就像把原函数当成构造器。提供的 this 值被忽略，同时调用时的参数被提供给模拟函数。

```
// 第三版
Function.prototype.bind2 = function (context) {
    var self = this;
    var args = Array.prototype.slice.call(arguments, 1);
    var fBound = function () {
        var bindArgs = Array.prototype.slice.call(arguments);
        // 当作为构造函数时，this 指向实例，此时结果为 true，将绑定函数的 this 指向该实例，
可以让实例获得来自绑定函数的值
        // 以上面的是 demo 为例，如果改成 `this instanceof fBound ? null : context`，
实例只是一个空对象，将 null 改成 this ，实例会具有 habit 属性
        // 当作为普通函数时，this 指向 window，此时结果为 false，将绑定函数的 this 指向
context
```

```
        return self.apply(this instanceof fBound ? this : context,
args.concat(bindArgs));
    }
    // 修改返回函数的 prototype 为绑定函数的 prototype，实例就可以继承绑定函数的原型中的值
    fBound.prototype = this.prototype;
    return fBound;
}
```

最终版：

```
Function.prototype.bind2 = function (context) {
    //调用的不是函数报错
    if (typeof this !== "function") {
      throw new Error("Function.prototype.bind - what is trying to be bound is
not callable");
    }
    //保存当前环境的this
    var self = this;
     // 获取bind2函数从第二个参数到最后一个参数
    var args = Array.prototype.slice.call(arguments, 1);
    //创建一个空函数作为中转
    var fNOP = function () {};

    var fBound = function () {
         // 这个时候的arguments是指bind返回的函数传入的参数
        var bindArgs = Array.prototype.slice.call(arguments);
        //判断this是否指向实例，指向实例则为构造函数,将this绑定为实例对象，否则将绑定函数的
this 指向 context
        return self.apply(this instanceof fNOP ? this : context,
args.concat(bindArgs));
    }
     // 修改返回函数的 prototype 为绑定函数的 prototype，实例就可以继承绑定函数的原型中的值
    fNOP.prototype = this.prototype;
    fBound.prototype = new fNOP();
    return fBound;
}
```

10、浏览器的事件循环

（js为什么是单线程的） 如何保证代码的执行,宏任务和微任务

1、js为什么是单线程的

js是脚本语言，主要用来与用户互动，操作dom，如果是多线程的，有两个线程对同一个DOM进行添加和删除，这时候浏览器就不知道以哪个线程为准了

2、如何保证代码的执行

执行栈：js第一次执行时，js引擎会将其中的同步代码放到执行栈中从头顺序执行，通过执行的是一个方法则在执行栈中添加该方法的执行环境，在该环境执行代码，当该方法代码执行完了就会销毁该执行环境，回到上一级的执行环境

3、事件循环

js事件分为同步事件和异步事件

js引擎遇到一个异步事件后并不会一直等待其返回结果，而是会将这个事件挂起，继续执行执行栈中的其他任务。当一个异步事件返回结果后，js会将这个事件加入与当前执行栈不同的另一个队列，我们称之为事件队列。被放入事件队列不会立刻执行其回调，而是等待当前执行栈中的所有任务都执行完毕，主线程处于闲置状态时，主线程会去查找事件队列是否有任务。如果有，那么主线程会从中取出排在第一位的事件，并把这个事件对应的回调放入执行栈中，然后执行其中的同步代码...，如此反复，这样就形成了一个无限的循环。这就是这个过程被称为“事件循环（Event Loop）”的原因。

4、js任务：

js任务分为同步和异步任务，同步是阻塞的，而异步是非阻塞的，

同步任务在主线程上执行，异步任务进入任务队列，当异步操作被触发时才放到主线程里面执行

异步任务又分为宏任务和微任务：

宏任务：setTimeout、setInterval、setImmediate、I/O、UI rendering

微任务：promise.then、process.nextTick、MutationObserver、queneMicrotask(开启一个微任务)

一个宏任务执行后会查看是否有微任务，有的话会先执行完所有微任务（期间有新微任务加入会一并执行），执行完后再继续执行下一个宏任务

4、使用微任务的原因：

- 减少操作中用户可感知到的延迟
- 确保任务顺序的一致性，即便当结果或数据是同步可用的
- 批量操作的优化

11、事件代理(事件委托和冒泡)

一个事件发生后会在子元素和父元素之间传播，分为3个阶段

1、从window对象传导到节点，称为捕获阶段

2、在目标节点上触发，称为“目标阶段”

3、从目标节点传导会window对象，称为冒泡阶段

事件代理：由于事件会在冒泡阶段向上传播到父节点，因此可以把子节点的监听函数定义在父节点上，由父节点的监听函数统一处理多个子元素的事件。这种方法叫做事件的代理（delegation）。

如果希望事件到某个节点为止，不再传播，可以使用事件对象的`stopPropagation`方法，但只会阻止当前事件的传播，而不能阻止其他事件的传播（即后面有同类事件仍会传播），要彻底阻止所有同类事件的监听函数不再触发，要使用stopImmediatePropagation

12、vue了解得怎么样

会用，没读过源码（别骂了别骂了）。面试官就没再问了

（下次要淡定地说还行）

13、http协议，tcp和udp的区别，https

http

1、http:即是超文本传输协议（HyperText Transfer Protocol），是一个专门在两点之间传输文字，图片，音频，视频等超文本数据的约定和规范,基于TCP/UDP的应用层协议

HTTP 协议是基于 TCP 协议出现的。TCP 协议是一条双向的通讯通道。HTTP 在 TCP 的基础上，规定了 Request-Response 的模式。这个模式决定了通讯必定是由浏览器首先发起的。

2、HTTP特性（优点）：

- 简单：格式为header+body ，容易理解
- 灵活易扩展：各个部分的组成没有被固定死，可以自由扩充，而且工作在应用层，下层可以随意变化（https就是在Http和TCP中间加了SSL/TLS 安全传输层，http/3甚至把TCP层换成了基于udp的QUIC）
- 应用广泛和跨平台
- 无状态：服务器不会去记忆http状态，可减少负担

3、缺点：

- 无状态：没有记忆能力，完成关联性操作比较麻烦（可使用缓存机制弥补（可能会问到缓存相关问题））
- 明文传输：方便了阅读，却暴露了信息，容易被窃取
- 不安全：
    - 明文可能导致消息泄露，
    - 没有验证身份，通信身份可能遭遇伪装
    - 无法证明报文完整性，容易被篡改
    - 队头堵塞问题

4、状态码：

- 1xx：临时回应，表示客户端应继续。
- 2xx：请求成功。
    - 200：请求成功。
- 3xx：请求的目标有变化，需要客户端进一步处理。
    - 301&302：永久性与临时性跳转。
    - 304：跟客户端缓存没有更新。
- 4xx：客户端请求错误。
    - 400：请求语法有误
    - 401：无权限访问
    - 403：服务端拒绝服务。
    - 404：请求的页面不存在。
    - 408：等待超时
- 5xx：服务端请求错误。
    - 500：服务器端错误。
    - 501：功能服务端不支持
    - 502：服务端正常，但错误未知
    - 503：服务端暂时性错误，可以一会儿再试。（服务端满）

（还有http1.0，hhtp1.1，http2的坑，以后再补）

http0.9：只支持html，只有get请求，无转态码和错误代码

http1.0：支持http头，协议的版本信息要随请求一起发送，支持了head好

http1.1：长连接，支持分块/断点续传，管道传输，host头域（URL要带上主机名），队头堵塞问题

http2.0：二进制分帧，多路复用，header压缩，数据流优先级

tcp和udp的区别

1、tcp即传输控制协议：是基于连接的协议，在收发数据前，双方必须建立可靠的连接，是面向字节流的传输

一个tcp建立必须进行三次握手

三次握手：

（1） 主机A向主机B发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；

（2）主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；

（3）. 主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，然后双方就建立了连接这是第三次对话。

为什么是三次不是两次?

三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据

两次即服务端同意连接后即马上准备连接，如果第一次发的请求延时到达，而请求端重发了数据包建立了连接后延时的请求到了服务端，服务端有开启了连接，而请求端并没有开启，则服务端会一直开着一个无用的连接，而三次请求端只要不理重发的响应包，连接就不会建立

2、udp： 用户数据报协议，在正式通信前不必与对方先建立连接，不管对方状态就直接发送。这与现在风行的手机短信非常相似：你在发短信的时候，只需要输入对方手机号就OK了。

	TCP	UDP
是否连接	面向连接	面向非连接
传输可靠性	可靠	不可靠
应用场合	传输大量的数据，对可靠性要求较高的场合	传送少量数据、对可靠性要求不高的场景
速度	慢	快

https

1、http和https的区别

- http明文传输，https加密传输
- http三次握手后即建立连接，https在三次握手后还要进行SSL/TLS的握手再加密传输
- http的默认端口为80，http是的端口为443
- https需要向CA申请数字证书

2、通信过程

- https使用混合加密（对称加密+非对称加密）方式保证信息的机密性：
  - 通信前使用非对称加密交换[会话密钥]（客户端与服务端先通过通信协商好加密算法，客户端使用数字证书中的服务器公钥加密生成会话密钥的信息发送给服务器，服务器使用私钥解密，并使用协商算法算出会话密钥）
  - 通信过程使用对称加密加密数据
- 使用摘要算法实现数据完整性
- 使用数字证书保证服务端身份

14、cookies和session，怎么防止cookies被劫持

- Session:就是客户端请求服务端时，服务端为这次请求开辟的一块`内存对象`。服务器会利用session **存储客户端在同一个会话期间的一些操作记录。**会话结束就失效，Session是基于cookie的机制完成的（一个会话结束就过期的cookie）

  缺点：比如 A 服务器存储了 Session，就是做了负载均衡后，假如一段时间内 A 的访问量激增，会转发到 B 进行访问，但是 B 服务器并没有存储 A 的 Session，会导致 Session 的失效。

- cookies：它是服务器发送到 Web 浏览器的一小块数据。服务器发送到浏览器的 Cookie，浏览器会进行存储，并与下一个请求一起发送到服务器。通常，它用于判断两个请求是否来自于同一个浏览器，例如用户保持登录状态。
  - cookie主要用于会话管理（登录，购物车等），个性化（用户习惯），追踪（记录分析用户行为）
  - cookie有两种：
    - Session Cookies，没有到期日期，存在内存中，浏览器关闭就丢失（但浏览器可能会还原）
    - Persistent Cookies：有有效期，存在硬盘中，到期就删除

15、打地鼠游戏要怎么设计,数据结构

不会！！！，我又不是做游戏的，我tmd怎么知道怎么设计

16、设计模式

- 工厂模式（知道）
- 单例模式（知道）
- 建造者模式
- 适配器模式
- 装饰器模式
- 代理模式
- 原型模式
- 备忘录模式
- 观察者模式（知道）
- 策略模式

https://blog.csdn.net/qq_33706840/article/details/81631762?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&dist_request_id=1328740.2577.16167733611188871&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control

17、有什么问题要问吗?

18、建议

- 不要紧张，回答时整理好思路再回答
- 基础概念要再复习，陈述问题时不能用我觉得，我认为等口语化的词，表述要正确，面试前要好好复习
- 介绍时要突出自己的亮点，因为面试官有可能没细看你的简历，这时候强调一下有利于引导面试官问问题
- 要多写代码，项目难点要体现出个人解决问题的实力