发言总结

0.开场白

我是来自郑州大学计算机科学与技术专业的一名大四学生，目前从事前端方向相关工作。接下来我想把自我介绍分为三个部分：第一个部分是我的实习工作经历；第二个部分是我的技术栈；最后我将简略概括一下我之前写的三个项目。首先来介绍我的实习工作经历：

1.实习经历

目前，我担任的岗位是腾讯-业务安全部的前端开发实习生，工作的内容是：负责业务安全部端游对抗中心的一个内部外挂对抗监控平台-tguard 平台的维护以及新功能的开发工作，使用的技术栈为：Angular1+Layui+Echarts。

我认为在实习工作中解决的比较值得说地方有以下几点：

- 对平台的优化：

  使用 localStorage 结合 cookie 实现对部分更新频率低且高度复用的数据进行存储，其中使用 localStorage 存储数据字典，使用 cookie 存储 game_id

- 对用户体验的提升：

  使用 sessionStorage 存储当前页面的标识，在连续跳转其他页面后，返回当前页面，加载后将页面还原，其中主要标记列表当前数据页，以及哪一列进行了排序，我认为其中的难点就是准确标记数据列表哪一列进行了排序，进行了几次排序，这里的存储数据中哪一列进行了排序，逻辑是这样的：先获取 sessionStorage 中对应的标记，这里我把列名的标记起名为 showKey，排序标记起名为 showFlag，排序的话，点击该列只有按照字符串排序进行正排和反排，然后我将正排标记记为 0，反排位-1，通过位运算转换 showFlag 的值，因为 0 按位取反是-1，-1 按位取反是 0

```
let showFlag = sessionStorage.getItem('showFlag') || 0;
let showKey = sessionStorage.getItem('showKey');
if (showKey == key) {
   showFlag = ~showFlag;
sessionStorage.setItem('showFlag', showFlag);
} else {
sessionStorage.setItem('showFlag', 0);
}
//注：此处 key 值为排序函数中获取的
sessionStorage.setItem('showKey', key);
```

- 避免框架间的冲突：

  Angular 与 JQuery 的冲突，angular 操作 DOM 是在真实 DOM 加载之后，故 jquery 操作 dom 要先于 angular，对此，我的策略是统一操作，某个页面全部使用 angular 中的指令替代 jquery 操作，或者全部使用 jquery 替代

- Layui 框架缺陷：

  Layui 渲染表单元素后，经常会出现如果数据更新，样式渲染不及时的问题，比如复选框的 value 与文本框绑定时出现的问题，使用 layui 的表单监听出现复选框的 value 可以传达到文本框，但是文本框的值改变时，多选框的样式却没有对应改变，查看 element，

值已经传给多选框，但是多选框使用的 layui 渲染样式，样式没有及时渲染到的 bug，针对这种情况，我这边用 angular 监听文本框，当其值改变时，使用 js 或 jquery 操作样式，因为 layui 渲染多选框是将原☐隐藏，然后在它的位置重新添加一个 div，并生成对应样式，于是我使用 jquery 操纵该 div 的属性，当文本框出现与多选框匹配的数据时，添加 layui-form-checked 属性，即多选框被选中的属性，没有时则移除。

- 增加新功能，中英文切换：

  我在论坛上了解到实现这个功能有三种方案：

  （1） 引入百度或者 google 翻译 api

  （2） 每个页面写一个两个，一个中文版，一个英文版

  （3） 将页面每个含有文字的元素加上 key，并写一个双语的 json 语言包，再点击按钮切换语言时转换

  最终我选择了第三种，因为第三种相对于第一种，可以做到不同语言页面私有化的部署，自定义修改词汇，机器翻译的话，很多词语词不达意；相对于第二种，成本要低很多，如果使用第二种，此平台那么多页面过于复杂。

  实现步骤是：在主页面添加按钮，对每个需要转换的元素添加相同 class（类）名，以及独有的 key，并添加对应的 JS，JS 的主要功能是浏览器刚启动时检测当前浏览器语言环境，并调用语言包中对应的语言，(使用 navigator.language 获取语言 //"en-US")当使用者点击页面上中文或者英文按钮时，对应的将中文或者英文语言包通过 key 遍历修改元素的 html 文本

```
$(document).ready(function () {
   $(".lang").each(function (index, element) {
      $(this).text(arrLang[lang][$(this).attr("key")]);
   });
});

// get/set the selected language
$(".translate").click(function () {

   var lang = $(this).attr("id");

   // update localStorage key
   if ('localStorage' in window) {
      localStorage.setItem('lang', lang);
      console.log(localStorage.getItem('lang'));
   }

   $(".lang").each(function (index, element) {
      $(this).text(arrLang[lang][$(this).attr("key")]);
   });
});
```

2.个人技术栈

- HTML、CSS、JS（Bom、Dom、Ajax、Json）、Jquery、以及 ES6

（1）HTML：

块级元素：总是从新的一行开始，高度宽度可控，宽度默认值为 100%，块级元素可以包含块级元素和行内元素，例：address、center、h1-h6、hr（水平分割线）、p（段落）、pre（预格式化）、ul（无序）、ol（有序）、dl（定义列表）、table（表格）、form（表单）、div。

行内元素：和其他元素都在一行，宽高和 padding 不可控，且宽高是内容的高度，不可改变，行内元素内只能包含行内元素，不能包含块级元素，例：span、a（链接）、br（换行）、b/strong（加粗）、img（图片）、input（文本框）、textarea（多行文本）、select（下拉列表）

（2）CSS：

CSS 盒模型：标准盒模型 height 和 width 等于 content，IE 盒模型 height 和 width 等于 content+padding+border（故标准盒模型的宽为左右 border+左右 padding+width，IE 盒模型的宽为 width），CSS3 中 box-sizing：content-box（标准盒模型）、box-sizing：border-box（IE 盒模型）；

CSS 权重：important（无限高），行内样式（权重 1000）、id 选择器（权重 100）、类伪类属性选择器（权重 10）、元素选择器（权重 1），比较规则：1.先从高等级进行比较，高等级相同时，再比较低等级的，以此类推；2.完全相同的话，就采用 后者优先原则（也就是样式覆盖）；3.css 属性后面加 !important 时，无条件绝对优先（比内联样式还要优先）；

em：1em 与父级字体大小相同

定位（position）：relative（相对定位）相对于其正常位置定位、absolute（绝对定位）相对于父元素定位、fixed（固定定位）相对于浏览器窗口定位、inherit（继承定位）继承父元素 position 的值

水平垂直居中实现方式：父级元素设置 text-align:center 然后高度设置为 line-height（行高）；父节点设置 display:grid，子节点设置 align-self:center;justify-self:center；父节点设置 display:flex;justify-content:center;align-items:center;

（3）BOM：

window 对象：全局作用域等

location 对象：主要用来跳转页面（改变浏览器位置），常见方法是设置 location.href 属性，例：URL 为：“http://www.wrox.com:8080/finndu/#section1?q=javascript”，location.hash=”#section1”，location.search=”?q=javascript”，location.hostname=”www.wrox.com”，location.pathname=”finndu”，location.port=”8080”

navigator 对象：language（判断浏览器语言类型），cookieEnabled（判断 cookie 是否启用），useAgent（判断浏览器类型，搭配 toLowerCase 转换为小写使用）

history 对象：history.go()正数前进，负数后退，具体参数跳转具体位置，history.back()后退，history.forward()前进

（4）DOM：

Document 表示整个文档，

（5）Ajax：

创建 Ajax 对象，向服务器发送请求

open(method,url,async)：规定请求的类型，URL 以及是否异步，method：请求的类型（GET/POST），url：文件位置，async：true（异步）false（同步）；

send（string）：将请求发送到服务器，string 仅用于 post 请求。

0：请求未初始化；1：服务器连接已建立；2：请求已接受；3：请求处理中；4：请求已完成，且相应已就绪。

（6）JS 基础：

JS 数据类型：Undefined、Null、Boolean、Number、String（五个基本数据类型），Object、

Array、Function（三个引用类型），基础类型存储的是值，引用类型存的是指针；

数组方法：toString() 把数组转为字符串；join()将数组结合为字符串：[1,2,3]。Join(“*”)=”1*2*3”；pop()删除数组最后一个元素，返回值为被删除的值；push()在数组结尾处添加一个新元素，返回值为新数组长度；shift()删除首个数组元素，返回值为被弹出的元素；unshift()在数组开头添加新元素，返回值为新数组长度；splice(2,0,”XXX”)拼接数组，2 为位置索引，0 为删除的元素个数，”XXX”为填充的新元素；slice(1,3)裁剪数组，从位置 1 到 3 裁剪，返回值为裁剪后的数组，不影响原数组；sort()数组排序，按字符排序，如果改成数字排序，需要加比值函数，array.sort(function(a,b){return a-b}) 升序排序；reverse()反转数组元素，array.reverse()；array.forEach(“回调函数”)，forEach()会改变原数组；array.map(“回调函数”)，map 不会改变原数组。

字符串方法：str.length，返回字符串长度；indexOf()查找字符串中的字符串，返回首次出现的索引；lastIndexOf()返回最后一次出现的索引；slice(start,end)裁剪字符串，参数为负数时从字符串尾部开始计数。

闭包：函数嵌套函数，函数内部引用函数外部的参数和变量，参数和变量不会被垃圾回收机制回收；

原型和原型链：js 的对象中都包含一个 protype 的内部属性，这个属性对应的就是该对象的原型；每个对象都有原型，对象的原型指向原型对象，然后其父的原型指向他的原型对象，这种原型层层连接起来就构成了原型链

对 this 的理解：this 指的是它所属的对象，在方法中谁调用它它就是谁，单独使用则为全局对象。

call()、apply()、bind()：都是用来重定义 this 对象的，示例代码：

```
var name = 'finndu';
var age = 17;
var obj = {
   name:'杜屹峰',
   objage:this.age,
   myFun:function(go,come){
      console.log(this.name +'年龄'+this.age,'来自'+go+'去往'+come);
   }
};
var db = {
   name:'德玛',
   age:99
}
obj.myFun.call(db,'成都','深圳');//德玛年龄 99 来自成都去往深圳
obj.myFun.apply(db,['成都','深圳']);//德玛年龄 99 来自成都去往深圳
obj.myFun.bind(db,'成都','深圳')();//德玛年龄 99 来自成都去往深圳
obj.myFun.bind(db,['成都','深圳'])();//德玛年龄 99 来自成都，深圳去往 undefined
```

故，三者参数允许是各种类型，apply 多个参数需要放到数组里传进去，bind 必须调用了才会执行，参数可以 call 一样

实现 call

第一个参数为 undefined 或 null 的时候，那么会转变为 window

改变了 this 执行，让新的对象可以执行该函数。

```
Function.prototype.myCall = function(context) {
    if (typeof context === "undefined" || context === null) {
        context = window
    }
   //context=context||window   和上面的代码一样
    context.fn = this
    const args = [...arguments].slice(1)
    const result = context.fn(...args)
    delete context.fn
    return result
}
```

实现 apply

apply 和 call 实现类似，不同的就是参数的处理

```
Function.prototype.myApply = function(context) {
    if (typeof this !== 'function') {
        throw new TypeError('Error')
    }
    context = context || window
    context.fn = this
    let result
    if (arguments[1]) {
        result = context.fn(...arguments[1])
    } else {
        result = context.fn()
    }
    delete context.fn
    return result
}
```

实现 bind

因为 bind 转换后的函数可以作为构造函数使用，此时 this 应该指向构造出的实例，而不是 bind 绑定的第一个参数

```
Function.prototype.myBind = function(context) {
    if (typeof this !== 'function') {
        throw new TypeError('Error')
    }
    //返回一个绑定 this 的函数，这里我们需要保存 this
    const _this = this
    const args = [...arguments].slice(1)
        //返回一个函数
    return function F() {
        //因为返回一个函数，我们可以 new F()需要判断能当做构造函数吗
        if (this instanceof F) {
            return new _this(...args, ...arguments)
        }
```

```
            return _this.apply(context, args.concat(...arguments))
        }
}
```

垃圾回收机制：大都采用的方案为标记清除和引用计数。

标记清除：垃圾回收机制在运行的时候会给存储再内存中的所有变量都加上标记（可以是任何标记方式），然后，它会去掉处在环境中的变量及被环境中的变量引用的变量标记（闭包）。而在此之后剩下的带有标记的变量被视为准备删除的变量，原因是环境中的变量已经无法访问到这些变量了。

引用计数：语言引擎有一张"引用表"，保存了内存里面所有资源（通常是各种值）的引用次数。如果一个值的引用次数是 0，就表示这个值不再用到了，因此可以将这块内存释放。

防抖：就是指触发事件后，在 n 秒内函数只能执行一次，如果触发事件后在 n 秒内又触发了事件，则会重新计算函数延执行时间（类似于玩游戏人物的技能前摇）

```
例：function debounce(fn, wait) {
    var timer;
    return function () {
            var args = Array.prototype.slice.apply(arguments);
            if (timer) {
                clearTimeout(timer);
            }
            timer = setTimeout(function () {
                fn.apply(this, args);
            }, wait);
    };
}
```

节流：在持续触发事件时，保证一定的时间段内只调用一次事件处理函数（类似于游戏中人物技能的 CD）

```
例：function throttle(fn, wait) {
        var lastTime = new Date().getTime();
        return function () {
            var args = Array.prototype.slice.apply(arguments);
            var curTime = new Date().getTime();
            if ((curTime - lastTime) < wait) {
                return;
            };
            lastTime = curTime;
            fn.apply(this, args);
        };
}
```

去重：indexOf()去重，新建一个数组，遍历要去重的数组，当值不在新数组的时候（indexOf 为 -1）就加入该新数组中；set 去重，return ([···new Set(array)])；sort()去重，将数组排序，比较前后位元素，有相同的删除当前位置的值。

大量 dom 优化的办法：https://blog.csdn.net/m0_45070460/article/details/107604658

（7）ES6:

var let const：let 声明的变量只在其所在的代码块内有效，const 声明的是一个只读的常量，let 和 const 声明时都必须赋初始值，并且都不能重复声明，不存在变量提升。

Symbol：表示独一无二的值，用法是用来定义对象的唯一属性名。

Set：Set 对象允许你存储任何类型的唯一值，无论是原始值或者是对象引用。

应用：交集 var a = new Set([1, 2, 3]); var b = new Set([4, 3, 2]); var intersect = new Set([...a].filter(x => b.has(x))); // {2, 3}、并集 var a = new Set([1, 2, 3]); var b = new Set([4, 3, 2]); var union = new Set([...a, ...b]); // {1, 2, 3, 4}、差集 var a = new Set([1, 2, 3]); var b = new Set([4, 3, 2]); var difference = new Set([...a].filter(x => !b.has(x))); // {1}

箭头函数：

```
// 两个参数:
(x, y) => x * x + y * y
// 无参数:
() => 3.14
// 可变参数:
(x, y, ...rest) => {
    var i, sum = x + y;
    for (i=0; i<rest.length; i++) {
        sum += rest[i];
    }
    return sum;
}
```

Promise：Promise 异步操作有三种状态：pending（进行中）、fulfilled（已成功）和 rejected（已失败）。除了异步操作的结果，任何其他操作都无法改变这个状态；可链式调用。

```
例：const p = new Promise(function(resolve,reject){
    resolve(1);
}).then(function(value){ // 第一个 then // 1
    console.log(value); return value * 2;
}).then(function(value){ // 第二个 then // 2
    console.log(value);
});
```

export 和 import：export 导出，import 导入

```
/*-----export [test.js]-----*/
let myAge = 20;
let myfn = function(){
    return "My name is" + myName + "! I'm '" + myAge + "years old."
}
let myClass = class myClass { static a = "yeah!"; }
export { myAge, myfn, myClass }
/*-----import [xxx.js]-----*/
import { myAge, myfn, myClass } from "./test.js";
console.log(myfn());// My name is Tom! I'm 20 years old.
console.log(myAge);// 20
console.log(myClass.a );// yeah!
```

解构赋值：针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。

- H5、CSS3

（1） H5 新特性：语义化标签，Canvas，SVG，sessionStorage,locationStorage

（2） CANVAS 与 SVG 区别：CANVAS 通过 JS 绘制 2D 图形，SVG 使用 XML 描述的 2D 图形，CANVAS 就像动画，每次显示一帧，想修改就只能下一帧重新显示，SVG 图形则是以 DOM 节点的形式插入页面，可以用 JS 直接操作。

（3） CSS3 新特性

- Bootstrap、Layui
- AngularJS（依赖注入、路由）：依赖注入，使用 factory 传入参数以及返回值，控制器 controller，路由：通过#!+标记跳转。
- ECharts 中 Canvas 与 SVG
- Http 协议工作原理、http 常用状态码

（1） 跨域：

首先了解跨域是什么，当一个请求 url 的协议、域名、端口有任意一个与当前页面 url 不同的就是跨域。

跨域的几种方式：

【1】 主域相同，子域不同：浏览器是通过 document.domain 属性来检查两个页面是否同源，所以设置相同的 document.domain，就可以共享 cookie。

【2】 在同一个浏览器上，父子窗口互相发信息，使用 window.postMessage();

【3】 JSONP：兼容性好，但是只支持 get 请求，核心思想：网页通过添加一个<script>元素，向服务器请求 JSON 数据，服务器收到请求后，将数据放在一个指定名字的回调函数的参数位置传回来。

【4】 CORS（腾讯开心鼠官网也在用）：服务器端设置 header，Access-Control-Allow-Origin: *是全部，或者添加 URL

Nginx 设置：

```
location / {

        if ($request_method = 'OPTIONS') {
            add_header Access-Control-Allow-Origin *;
            add_header Access-Control-Allow-Methods
GET,POST,PUT,DELETE,OPTIONS;
            return 204;
        }
}
```

（2） http 与 https 的区别：

（3）http 状态码：2 开头为成功，3 开头为重定向，4 开头客户端错误，5 开头服务器错误，详细：

200 OK：表示从客户端发送给服务器的请求被正常处理并返回；

204 No Content：表示客户端发送给客户端的请求得到了成功处理，但在返回的响应报文中不含实体的主体部分（没有资源可以返回）；

206 Patial Content：表示客户端进行了范围请求，并且服务器成功执行了这部分的 GET 请求，响应报文中包含由 Content-Range 指定范围的实体内容。

301 Moved Permanently：永久性重定向，表示请求的资源被分配了新的 URL，之后应使用更改的 URL；

302 Found：临时性重定向，表示请求的资源被分配了新的 URL，希望本次访问使用新的 URL；

301 与 302 的区别：前者是永久移动，后者是临时移动（之后可能还会更改 URL）

303 See Other：表示请求的资源被分配了新的 URL，应使用 GET 方法定向获取请求的资源；

302 与 303 的区别：后者明确表示客户端应当采用 GET 方式获取资源

304 Not Modified：表示客户端发送附带条件（是指采用 GET 方法的请求报文中包含 if-Match、If-Modified-Since、If-None-Match、If-Range、If-Unmodified-Since 中任一首部）的请求时，服务器端允许访问资源，但是请求为满足条件的情况下返回改状态码；

307 Temporary Redirect：临时重定向，与 303 有着相同的含义，307 会遵照浏览器标准不会从 POST 变成 GET；（不同浏览器可能会出现不同的情况）；

400 Bad Request：表示请求报文中存在语法错误；

401 Unauthorized：未经许可，需要通过 HTTP 认证；

403 Forbidden：服务器拒绝该次访问（访问权限出现问题）

404 Not Found：表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用；

500 Inter Server Error：表示服务器在执行请求时发生了错误，也有可能是 web 应用存在的 bug 或某些临时的错误时；

503 Server Unavailable：表示服务器暂时处于超负载或正在进行停机维护，无法处理请求；

（3） 重定向

当 client 向 server 发送一个请求，要求获取一个资源时，在 server 接收到这个请求后发现请求的这个资源实际存放在另一个位置，于是 server 在返回的 response 中写入那个请求资源的正确的URL，并设置reponse的状态码为301（表示这是一个要求浏览器重定向的response），当 client 接受到这个 response 后就会根据新的 URL 重新发起请求。

- 计算机网络

（1） 网页 HTTP 请求的整个过程

1.打开浏览器，地址栏输入 blog.csdn.net。

2.开始进行域名解析

3.浏览器自身搜 dns 缓存 搜 blog.csdn.net 有没有缓存 看看有没有过期，如果过期就这个结束；

【1】 搜索操作系统 自身的 dns 缓存；

【2】 读取本地的 host 文件；

【3】 浏览器发起一个 dns 的一个系统调用

3. 浏览器获得域名对应的 ip 地址后 发起 http 三次握手

4. tcp/ip 链接建立起来后，浏览器就可以向服务器发送 http 请求 。

5. 服务器端接受到请求，根据路径参数，经过后端的一些处理之后，把处理后的一个结果数据返回给浏览器，如果是一个完整的网页，就是把完整的 html 页面代码返回给浏览器。

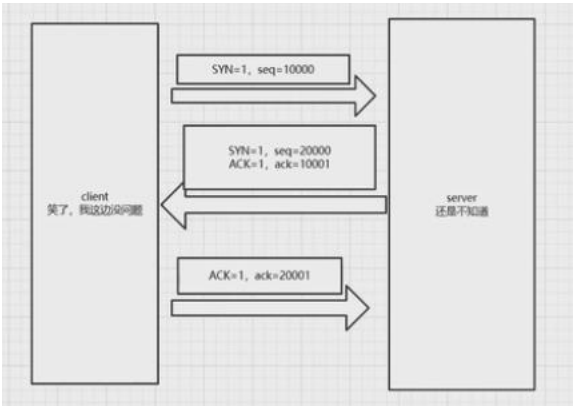
6. 浏览器拿到 html 页面代码，解析和渲染页面，里面的 js、css 图片资源都需要经过上面的步骤。

7. 浏览器拿到资源对页面进行渲染，最终把一个完整的页面呈现给用户。

（2）TCP 三次握手

TCP/IP 是传输层面向连接的安全可靠的协议，三次握手的机制是为了保证能建立安全可靠的连接。

TCP 建立的连接是安全可靠的连接：客户端知道自己能连上服务端，服务端也能连上上自己；服务端也要知道自己能连上客户端，客户端能连上自己。



TCP 报文中有：32 位序号 seq(initial sequence number) + 32 位确认序号 ack(acknowledge number) 还有标记位：ACK

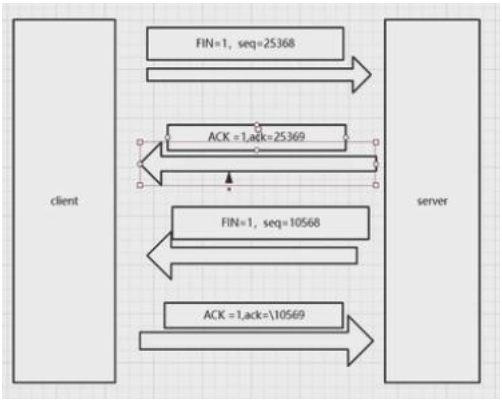
【1】客户端给服务端发起一个建立连接的请求(报文)：标志位 SYN=1，同时会随机生成一个 32 位的 initial sequence number 初始化序列号 X。

【2】服务端在收到了请求之后，就会发出一个确认消息报：ACK=1，同时有一个 32 位的 acknowledge number 确认序号 X+1。除此以外，它还会发送一个建立连接的请求：SYN=1，seq=Y。它会把上面这些打包发送给客户端。在经过上面两次握手之后，客户端这边其实已经可以满足：自己连上服务端，服务端也能连上自己的条件。现在还不能满足的就是：服务端这边能确定客户端能连上自己，但是不确定自己能否连上服务端。在这种情况下，便产生了第三次握手

【3】客户端在收到服务端的消息之后，立马给服务端一个消息确认包：ACK=1，acknowledge number 确认序号 Y+1

通过这三次握手，我们就能够保证客户端和服务器双方都知道我既能给对方发消息，也能收到对方的消息，从而建立了安全可靠的传输通道，也就是建立了全双工信道。

（2） TCP 四次挥手



【1】客户端请求断开连接：FIN 位=1，seq = X

【2】服务端回应应答数据包：ACK=1，ack = X+1

【3】服务端还会发送一个断开连接的数据包：FIN = 1，seq = Y

【4】客户端在收到消息后，会发送一个消息确认包：ACK=1，ack = Y+1

对比三次握手，我们来思考一下第二次和第三次挥手是否能像握手一样，合并成一次？实际情况是不可以的，原因是：tcp 连接是安全可靠的，客户端发送 FIN 之后，已经准备好了断开连接，不会在发送数据给服务端；但是服务端还没有做好准备，可能还有消息需要发送给客户端，因此这个逻辑就需要拆开：所以服务端只能先回复一个 ACK：相当于告诉客户端“好的客户端，你的邀请我已经收到了，但是我还没有准备好，你先等我的消息，我这边准备好

断开连接之后会通知你的"当服务端也准备好了，才会进行第三次挥手操作。

（3） TCP 和 UDP 的区别

1.TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的 UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）

2.每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信

3. TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

TCP 和 UDP 的区别【都是传输层】

TCP：面向连接（如打电话要先拨号建立连接）

1)优点：可靠,无差错、不丢失、不重复，且按序到达。TCP 数据传输之前，会有三次握手；而且传输数据时，有滑动窗口。超时重传。拥塞控制等机制；数据传输完后，会断开连接来节约系统资源 2)缺点：慢，效率低，占用系统资源高。而且 TCP 有确认机制、三次握手机制，导致 TCP 容易被人利用。

UDP：无连接的，即发送数据之前不需要建立连接

1)优点：快，尽最大努力交付，不保证可靠传输。比 TCP 稍安全，UDP 没有 TCP 的握手、确认、窗口、重传、拥塞控制等机制，UDP 是一个无状态的传输协议，所以它在传递数据时非常快。没有 TCP 的这些机制，UDP 较 TCP 被攻击者利用的漏洞就要少一些。2)缺点：不可靠，不稳定 因为 UDP 没有 TCP 那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

（4）HTTP 报文结构

HTTP 请求报文由请求行、请求头、空行和请求内容 4 个部分构成。

请求行：由请求方法字段、URL 字段、协议版本字段三部分构成，它们之间由空格隔开。常用的请求方法有：GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

请求头：请求头由 key/value 对组成，每行为一对，key 和 value 之间通过冒号(:)分割。请求头的作用主要用于通知服务端有关于客户端的请求信息

（4） GET 和 POST 的区别

GET 和 POST 是 HTTP 协议中的两种发送请求时最常用的方法。

1、get 请求：可被缓存；参数保留在浏览器历史记录中；可被收藏为书签；有长度限制

2、Post 请求：不会被缓存；参数不会保留在浏览器历史记录中；不能被收藏为书签；对数据长度无要求

3、GET 把参数包含在 URL 中，POST 通过 request body 传递参数。

（5）HTTP 与 HTTPS 的区别

1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。

2、http 信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全

- 数据结构链表、栈、队列，冒泡排序、选择排序、插入排序

（1）链表：（2）栈（3）队列

（4）冒泡排序：时间复杂度 O(n2)，空间复杂度 O(1)

实现原理：数组中有 n 个数，比较每相邻两个数，如果前者大于后者，就把两个数交换位置；这样一来，第一轮就可以选出一个最大的数放在最后面；那么经过 n-1（数组的 length - 1） 轮，就完成了所有数的排序。

```
function bSort(arr) {
  var len = arr.length;
  for (var i = 0; i < len-1; i++) {
    for (var j = 0; j < len - 1 - i; j++) {
          // 相邻元素两两对比，元素交换，大的元素交换到后面
        if (arr[j] > arr[j + 1]) {
            var temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
  }
  return arr;
}
```

（5）选择排序：时间复杂度 O(n^2)，空间复杂度 O(1)

选择排序的工作原理就是首先从原始数组中找到最小的元素，并把该元素放在数组的最前面，然后再从剩下的元素中寻找最小的元素，放在之前最小元素的后面，直到排序完毕。

```
function selectSort(arr){
    var len=arr.length;
    var minIndex,temp;
    for(i=0;i<len-1;i++){
        minIndex=i;
        for(j=i+1;j<len;j++){
            if(arr[j]<arr[minIndex]){
                minIndex=j;
            }
        }
    temp=arr[i];
    arr[i]=arr[minIndex];
    arr[minIndex]=temp;
    }
    return arr;
```

```
}
```

（6）插入排序：时间复杂度 O(n^2)，空间复杂度 O(1)

插入排序的工作原理就是将未排序数据，对已排序数据序列从后向前扫描，找到对应的位置并插入。

```
function insertSort(arr) {
    var len =arr.length;
    for (var i=1;i<len; i++) {
        var temp=arr[i];
        var j=i-1;//默认已排序的元素
        while (j>=0 && arr[j]>temp) {  //在已排序好的队列中从后向前扫描
            arr[j+1]=arr[j]; //已排序的元素大于新元素，将该元素移到一下个位置
            j--;
        }
        arr[j+1]=temp;
    }
    return arr
}
```

（7）快速排序：时间复杂度 O(n log n)，空间复杂度 O(log n).

通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

- Python 基础
- MySQL

（1） 索引

3.简历项目

- Jsp：session 登录注册，JDBC 连接数据库
- Angular+ECharts：通过 js 调整页面响应式，数据本地 json，通过指令添加到页面
- Python+Django：Django 做路由，拿来练手

4.笔试题反思

- 大数相加

```
function addBigNum(a,b){
    var res = '',
        loc = 0;
    a = a.split('');
    b = b.split('');
    while(a.length || b.length || loc){
        //~~把字符串转换为数字，用~~而不用 parseInt，是因为~~可以将 undefined 转换为 0，当 a 或 b 数组超限，不用再判断 undefined
        //注意这里的+=，每次都加了 loc 本身，loc 为 true，相当于加 1，loc 为 false，相当于加 0
        loc += ~~a.pop() + ~~b.pop();
        //字符串连接，将个位加到 res 头部
```

```
        res = (loc % 10) + res;
        //当个位数和大于 9，产生进位，需要往 res 头部继续加 1，此时 loc 变为 true，true +  任何数字，true 会被转换为 1
        loc = loc > 9;
    }
    return res.replace(/^0+/,'');
}
```

- 跨域、Postmessage
- 浏览器渲染过程：1.构建 DOM 树，2.构建 CSSOM 规则树，3.构建渲染树，4.渲染树布局，5.渲染树绘制
- 正则表达式

5.补充

- 腾讯开心鼠官网一些问题：二维码特效用的 hover，有跨域，开发环境调试。
- 快 UI：通过添加等待动画趣味性的 UI 设计
- Reactjs：基于 JSX，
- nodejs

6.问题

- 请问部门的人员配置
- 请问组内日常开发流程
- 这场面试觉得对我比较满意的地方，以及觉得我还有哪些不足
- 公司内代码开源率