

1、我接触过哪些数据类型

(1) 基本数据类型

1、Number: 数字

- `toFixed(num)`: 在数字后面调用, `num`为小数位, 有四舍五入的功能, 得到一个字符串
- `Math.round()` 四舍五入取整
- `parseInt()` 转化成整数
- `parseFloat()`转化成浮点数
- `Math.random()` 生成0-1随机数,不包括1

2、String: 字符串

3、Boolean: 布尔类型

Boolean 类型有两个值: `true`和`false`

(2) 引用数据类型

1、Array: 数组

2、Object: 对象

(3) 特殊数据类型

1、Null Null 类型是一个只有一个值的数据类型, 即特殊的值 `null`。它表示一个空对象引用(指针), 而 `typeof` 操作符检测 `null` 会返回 `object`

2、Undefined 类型只有一个值, 即特殊的 `undefined`, 在使用 `var` 声明变量, 但没有对其初始化时, 这个变量的值就是 `undefined`

3、NaN 是一个特殊的值, 即非数值(Not a Number)。数学运算无法得到数字时, 就会返回NaN

- 不代表任何值, 也不等于任何值, 甚至自己都不等于自己

- 任何数据与它运算都返回NaN
- isNaN(a): 用来判断a到底是不是非数字, 返回布尔值

2、什么时候会返回NaN

数字运算无法得到数字时

3、数据类型判断

(1)、typeof

```
typeof 'html5'; //=>string
typeof 100; //=>number
typeof true //=>boolean
typeof null //=>object
typeof array //=>object
```

(2) Object.prototype.toString.call() 判断 (最靠谱)

```
console.log(Object.prototype.toString.call(1)); // [object Number]
console.log(Object.prototype.toString.call('Hello')); // [object String]
console.log(Object.prototype.toString.call(false)); // [object Boolean]
console.log(Object.prototype.toString.call({})); // [object Object]
console.log(Object.prototype.toString.call([1, 2, 3])); // [object Array]
console.log(Object.prototype.toString.call(new Error('error!'))); // [object Error]
console.log(Object.prototype.toString.call(new Date())); // [object Date]
console.log(Object.prototype.toString.call(new RegExp())); // [object RegExp]
console.log(Object.prototype.toString.call(doSomething)); // [object Function]
console.log(Object.prototype.toString.call(null)); // [object Null]
console.log(Object.prototype.toString.call(undefined)); // [object Undefined]
console.log(Object.prototype.toString.call(JSON.stringify({
  name: 'helloworld'
}))); // [object String]
```

(3)、constructor 判断 (比较常用)

```
arr.constructor === Array; // true
obj.constructor === Object; // true
date.constructor === Date; // true
```

(4)、instanceof 判断

instanceof 判断 (了解)

`instanceof` 用来检测构造函数的 `prototype` 属性是否出现在某个实例对象的原型链上。语法: `object (实例对象) instanceof constructor (构造函数)`。是的话返回 `true`, 否则返回 `false`。所以, `instanceof` 运算符只能用作对象的判断。针对 `typeof` 不能判断的引用型数据, 我们可以使用 `instanceof` 运算符。

```
let arr1 = [1, 2, 3];
let obj1 = {
  name: '小明'
};
function Persion() { }
let persion1 = new Persion();
console.log(arr1 instanceof Array); // true
console.log(arr1 instanceof Object); // true, Array 是Object的子类
console.log(obj1 instanceof Object); // true
console.log(obj1 instanceof Array); // false
console.log(Persion instanceof Function, Persion instanceof Object); // true true
console.log(null instanceof Object); // false
console.log(persion1 instanceof Persion, persion1 instanceof Function, persion1 instanceof Obje
```

4、null和undefined的区别

(1)、定义

undefined: 是所有没有赋值变量的默认值, 自动赋值

null: 主动释放一个变量引用的对象, 表示一个变量不再指向任何对象地址

(2)、何时使用null?

当使用完一个比较大的对象时, 需要对其进行释放内存时, 设置为null

(3)、null与undefined的异同点是什么呢?

共同点: 都是原始类型, 保存在栈中变量本地

不同点:

1、undefined——表示变量声明过但并未赋过值。它是所有未赋值变量默认值。

2、null——表示一个变量将来可能指向一个对象。

5、then、catch 和 finally 序列能否顺序颠倒?

A: 可以, 效果完全一样。但不建议这样做, 最好按 then-catch-finally 的顺序编写程序。

6、除了 then 块以外, 其它两种块能否多次使用?

A: 可以, finally 与 then 一样会按顺序执行, 但是 catch 块只会执行第一个, 除非 catch 块里有异常。所以最好只安排一个 catch 和 finally 块。

7、then 块如何中断?

A: then 块默认会向下顺序执行, return 是不能中断的, 可以通过 throw 来跳转至 catch 实现中断。

8、什么时候适合用 Promise 而不是传统回调函数?

当需要多次顺序执行异步操作的时候, 例如, 如果想通过异步方法先后检测用户名和密码, 需要先异步检测用户名, 然后再异步检测密码的情况下就很适合 Promise。

9、Promise 是一种将异步转换为同步的方法吗?

A: 完全不是。Promise 只不过是一种更良好的编程风格。

10、什么时候我们需要再写一个 then 而不是在当前的 then 接着编程？

A: 当你又需要调用一个异步任务的时候。

11、异步函数和promise的代码

```
async function asyncFunc() {  
  await print(1000, "First");  
  await print(4000, "Second");  
  await print(3000, "Third");  
}  
asyncFunc();
```

哈！这岂不是将异步操作变得像同步操作一样容易了吗！

这次的回答是肯定的，异步函数 `async function` 中可以使用 `await` 指令，`await` 指令后必须跟着一个 `Promise`，异步函数会在这个 `Promise` 运行中暂停，直到其运行结束再继续运行。

异步函数实际上原理与 `Promise` 原生 API 的机制是一模一样的，只不过更便于程序员阅读。

处理异常的机制将用 `try-catch` 块实现：

实例

```
async function asyncFunc() {  
  try {  
    await new Promise(function (resolve, reject) {  
      throw "Some error"; // 或者 reject("Some error")  
    });  
  } catch (err) {  
    console.log(err);  
    // 会输出 Some error  
  }  
}  
asyncFunc();
```

如果 `Promise` 有一个正常的返回值，`await` 语句也会返回它：

实例

```
async function asyncFunc() {  
  let value = await new Promise(  
    function (resolve, reject) {  
      resolve("Return value");  
    }  
  );  
  console.log(value);  
}
```

12、Promise 并行（Promise.all是所有的Promise执行完毕后（reject|resolve）返回一个Promise对象。）

```
function a(resolve, reject) {  
  $.ajax({  
    url: `${api}/rrz/member/showProjectById`,  
    type: 'get',  
    data: { appId: appId },  
    success: function (res) {  
      if (res.result == 'success') {  
        gather['listBy'] = res.data;  
      }  
    }  
  });  
}
```

```

        resolve();
    }
}
});
}

```

```

function b(resolve, reject) {
    $.ajax({
        url: `${api}/rrz/member/showProjectById`,
        type: 'get',
        data: { appId: appId },
        success: function (res) {
            if (res.result == 'success') {
                gather['listBy'] = res.data;
                resolve();
            }
        }
    });
}

```

```

function c(resolve, reject) {
    $.ajax({
        url: `${api}/rrz/member/showProjectById`,
        type: 'get',
        data: { appId: appId },
        success: function (res) {
            if (res.result == 'success') {
                gather['listBy'] = res.data;
                resolve();
            }
        }
    });
}

```

```

var a1 = new Promise(a);
var a2 = new Promise(b);
var a3 = new Promise(c);

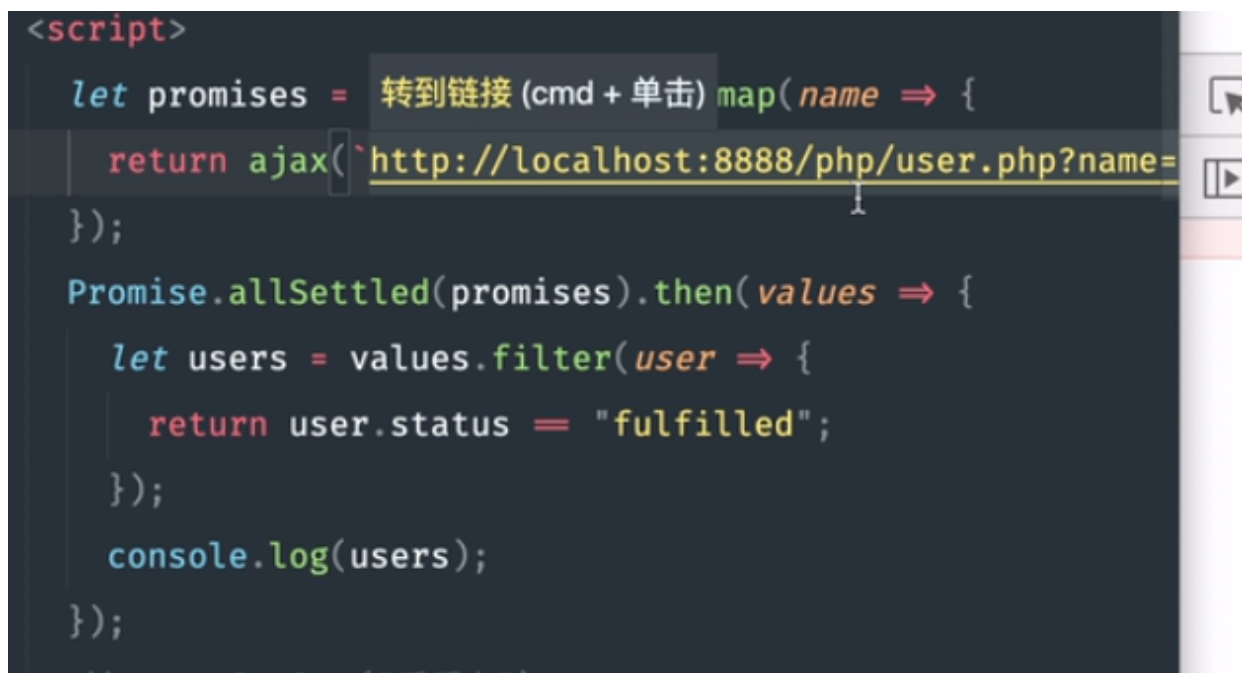
```

```

Promise.all([a1, a2, a3]).then(function (result) {
    info = { data: gather }
    getDetail();
})

```

```
console.log('loading效果图消失');  
})
```



```
<script>  
  let promises = 转到链接 (cmd + 单击) map(name => {  
    return ajax(`http://localhost:8888/php/user.php?name=  
  });  
  Promise.allSettled(promises).then(values => {  
    let users = values.filter(user => {  
      return user.status === "fulfilled";  
    });  
    console.log(users);  
  });  
</script>
```

Promise 接口:

all () 【处理并行，遇到失败就玩完】

allSettled () [用filter过滤自己想要的结果，至于没返回的也不影响]

race()[和all一样用，返回的是请求最快的接口]

13、Promise 串行()

```
function one() {  
  console.log(11111);  
}
```

```
function two() {  
  console.log(22222);  
}
```

```
function three() {
```

```

    console.log(33333);
}

function fiveP(func) {
    return new Promise(function(resolve, reject) {
        func();
        resolve();
    });
}

p.then(fiveP(one))
  .then(fiveP(three))
  .then(fiveP(two))
  .then(function(result) {
    console.log('最后执行' + result);
  });

```

14、js中var、let、const的区别

(1)、var：声明的是全局变量、列入：在for循环里边声明一个变量，在for外面依然可以获取到

```

for(var i=0;i<=1000;i++){
    var num=0;
    num+=i;
}
console.log(num);在这里打印num不会报错，依然能打印出结果；

```

(2)、let声明块级变量，即局部变量。【必须声明'use strict'；后才能使用let声明变量否则浏览并不能显示结果，】

在上面的例子中，跳出for循环，再使用sum变量就会报错

有着严格的作用域，变量只作用域当前隶属的代码块，不可重复定义同一个变量，不可在声明之前调用，必须先定义再使用，会报错，循环体中可以用let

(3)、const：用于声明常量，也具有块级作用域，也可声明块级。

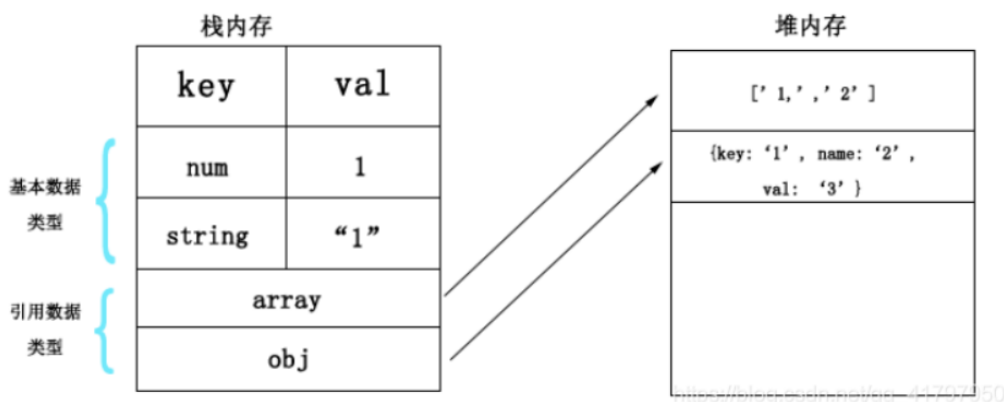
```
const PI=3.14;
```

它和let一样，也不能重复定义同一个变量，const一旦定义，无法修改

注意：const定义一个常量是不可修改的，修改会报错；

定义一个对象或者数组时，可以修改里边的数据；

注意注意：当const定义的常量为 '**基本数据类型**' 时，不能被修改；当定义的常量为 '**引用数据类型**' 时，我们可以通过其**属性进行**数据修改。'基本数据类型' 的值就保存在内存地址中，所以const定义的 '基础数据类型' 不可被改变。而 '引用数据类型' 指向的内存地址只是一个指针，通过指针来指向实际数据，也就是说，不可被改变的是指针，而不是数据，所以const定义的 " 引用数据类型的 ' 常量可以通过属性来修改值。这就牵扯出栈内存和堆内存。



16、父组件如何获取子组件里边的方法或者值

(1)、使用refs来调，首先父组件里调用子组件的地方,给子组件传个属性 ref = 'fromFather' ,然后在父组件调用this.refs.fromFather.子组件方法


```

var HelloMessage = React.createClass({
  childMethod: function(){
    alert("组件之间通信成功");
  },
  render: function() {
    return <div> <h1>Hello {this.props.name}</h1> <button onClick={this.childMethod}>子组件</button></div>
  }
});

// 父组件
var ImDaddyComponent = React.createClass({
  getDS: function(){
    // 调用组件进行通信
    this.refs.getSwordButton.childMethod();
  },
  render: function(){
    return (
      <div>
        <HelloMessage name="John" ref="getSwordButton" />
        <button onClick={this.getDS}>父组件</button>
      </div>
    );
  }
});

ReactDOM.render(
  <ImDaddyComponent />,
  document.getElementById('correspond')
);

```

(2)、直接在子组件componentDidMount方法中传递自己，在父组件调用子组件的地方，给子组件定义一个属性,并把自己的方法传过去,子组件componentDidMount中,使用调用该方法this.props.该方法(this),把子组件自己传过去.

```

import React, {Component} from 'react';

export default class Parent extends Component {
  render() {
    return(
      <div>
        <Child onRef={this.onRef} />
        <button onClick={this.click} >click</button>
      </div>
    )
  }

  onRef = (ref) => {
    this.child = ref
  }

  click = (e) => {
    this.child.myName()
  }
}

class Child extends Component {
  componentDidMount() {
    this.props.onRef(this)
  }

  myName = () => alert('xiaohesong')

  render() {
    return ('woqu')
  }
}

```

17、js遍历数组的几种方法

【<https://www.cnblogs.com/wangdashi/p/9431860.html>】

(1)、for循环

- (2)、forEach () , 三个值, item, index, arr (原数组) 无返回值, 不影响原数组
可以通过抛出异常的方式实现终止

```
try {  
    var array = ["first","second","third","fourth"];  
    array.forEach(function(item,index){// 执行到第3次, 结束循环  
        if (item == "third") {  
            throw new Error("EndIterative");  
        }  
        alert(item); // first, sencond  
    });  
} catch(e) {  
    if(e.message!="EndIterative") throw e;  
};
```

- (3)、map ()

forEach和map的区别

- 1、map()会分配内存空间存储新数组并返回, forEach()不会返回数据。
- 2、forEach()允许callback更改原始数组的元素。map()返回新的数组。
- 3、forEach() 适合并不打算改变数据的时候使用, 如: 存入数据库, 打印, 深copy;
- 4、map() 改变数据值的时候。因为它更快, 且返回一个新的数组。

- (4)、for(let item of arr){
 console.log(item)
}

- (5)、for(let item in arr){
 console.log(arr[item])
}

18、获取对象键值对中key值的方法

```
var obj = { 0: 'a', 1: 'b', 2: 'c' };  
Object.keys(obj)  
结果: ["0", "1", "2"]
```

19、字符串处理的方法有哪些

- (1)、indexOf 判断一个字符第一次在字符串中的索引, 如果包含返回索引, 不包含返回-1

```
var str = 'abcde';  
console.log(str.indexOf('e')); //4  
console.log(str.indexOf('f')); //-1
```

(2)、lastIndexOf 判断一个字符最后一次出现在字符串中的索引，存在返回索引，不存在返回-1

```
var str = 'abcdeb';  
console.log(str.lastIndexOf('b')); //5  
console.log(str.lastIndexOf('f')); //-1
```

(3)、concat 拼接两个字符串，返回新的字符串，对原字符串没有影响

```
var str='my';  
var str1='lazy';  
var str2=str.concat(str1);  
console.log(str2); //"mylazy"
```

(4)、substr(n,m) 从索引n开始，截取m个字符，将截取的字符返回，对原字符串没有影响

```
var str = 'abcde'  
var b = str.substr(1,1)  
console.log(b); //"b"
```

(4)、substring(n,m) 从索引n开始，获取到索引m，不包含m，将截取的字符返回，对原字符串没有影响

```
var str = 'abcde'  
var bc = str.substring(1,3);  
console.log(bc); //"bc"
```

(5)、slice(n,m)从索引n开始截取到索引m，不包括m，将截取的字符返回，对原字符串没有任何影响，

(6)、split() 用指定字符分割字符串，返回一个数组.对原字符串没有任何改变。

```
var str = 'abcde'  
var a = str.split("");  
console.log(a); //["a", "b", "c", "d", "e"]
```

(7)、replace('a',1)

替换指定字符，返回替换后新的字符串，对原有字符串有改变(第一个参数可以是正则表达式) 只能替换一次，配合正则式修饰符g使用

```
var str='aaaaee';  
var reg=/a/g;  
console.log(str.replace(reg,1)); //"1111ee"
```

20、react里面的props到底是什么，怎么用

概念

官网上是这么说的：

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".
意思是：当React看到表示用户定义组件的元素时，它会将JSX属性作为单个对象传递给此组件。我们称这个对象为“道具”。

```
1
2 function Welcome(props) {
3   return <h1>Hello, {props.name}</h1>;
4 } // 声明welcome 组件, 通过props.name 使用传递过来的属性。
5 const element = <Welcome name="Sara" />; // 使用welcome 组件, 将JSX属性作为单个对象传递给组件
6 ReactDOM.render(
7   element,
8   document.getElementById('root')
9 );
```

21、vue全家桶有哪些

1、vue-cli

```
# 全局安装 vue-cli
$ npm install --global vue-cli
# 创建一个基于 webpack 模板的新项目
$ vue init webpack my-project
# 安装依赖, 走你
cd my-project
$ npm install
$ npm run dev
```

2、vue-router

安装：npm install vue-router

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)
```

3、vuex

vuex主要由五部分组成：state action、mutation、getters、mudle组

成

4、axios

安装 npm install axios --save

22、react全家桶都有什么

1、react

react的核心。

2、redux

redux相当于一个数据库，可以当成一个本地的数据库使用，react-redux可以完成数据订阅，redux-thunk可以实现异步的action，redux-logger是redux的日志中间件。

3、react-router

react Router 是专为 React 设计的路由解决方案。它利用HTML5 的history API，来操作浏览器的 session history (会话历史)。

4、axios

axios是基于Promise的用于浏览器和Node.js的http客户端。可以发送get、post等http请求，用来和服务器进行交互的。

5、antd

Ant Design是个很好的React UI库

23、JavaScript在服务端 (Node.js) 和客户端的区别

1、客户端：客户端的JavaScript主要用来处理页面的交互，JavaScript需要依赖浏览器提供的JavaScript引擎解析执行，浏览器还提供了对DOM的解析，所以客户端的JavaScript不仅应用核心语法ECMAScript，还会操作DOM和BOM。

2、服务端：服务器端的JavaScript主要用来处理数据交互，JavaScript不依赖浏览器，而是由特定的运行环境提供的JavaScript引擎解析执行。服务器端的JavaScript应用核心语法ECMAScript，但是不操作DOM和BOM。它常常用来做一些在客户端做不到的事情，例如操作数据库、操作文件等等。

24、前端语义化标签

<header>表示页面中一个内容区块或整个页面的标题。

<section>页面中的一个内容区块，如章节、页眉、页脚或页面的其他地方，可以和h1、h2.....元素结合起来使用，表示文档结构。

<article>表示页面中一块与上下文不相关的独立内容，如一篇文章。

<aside>表示<article>标签内容之外的，与<article>标签内容相关的辅助信息。

<hgroup>表示对整个页面或页面中的一个内容区块的标题进行组合。

<figure>表示一段独立的流内容，一般表示文档主体流内容中的一个独立单元。

<figcaption>定义<figure>标签的标题。

<nav>表示页面中导航链接的部分。

`<footer>`表示整个页面或页面中一个内容区块的脚注。一般来说，它会包含创作者的姓名、创作日期及联系方式。



25、http和https协议的区别

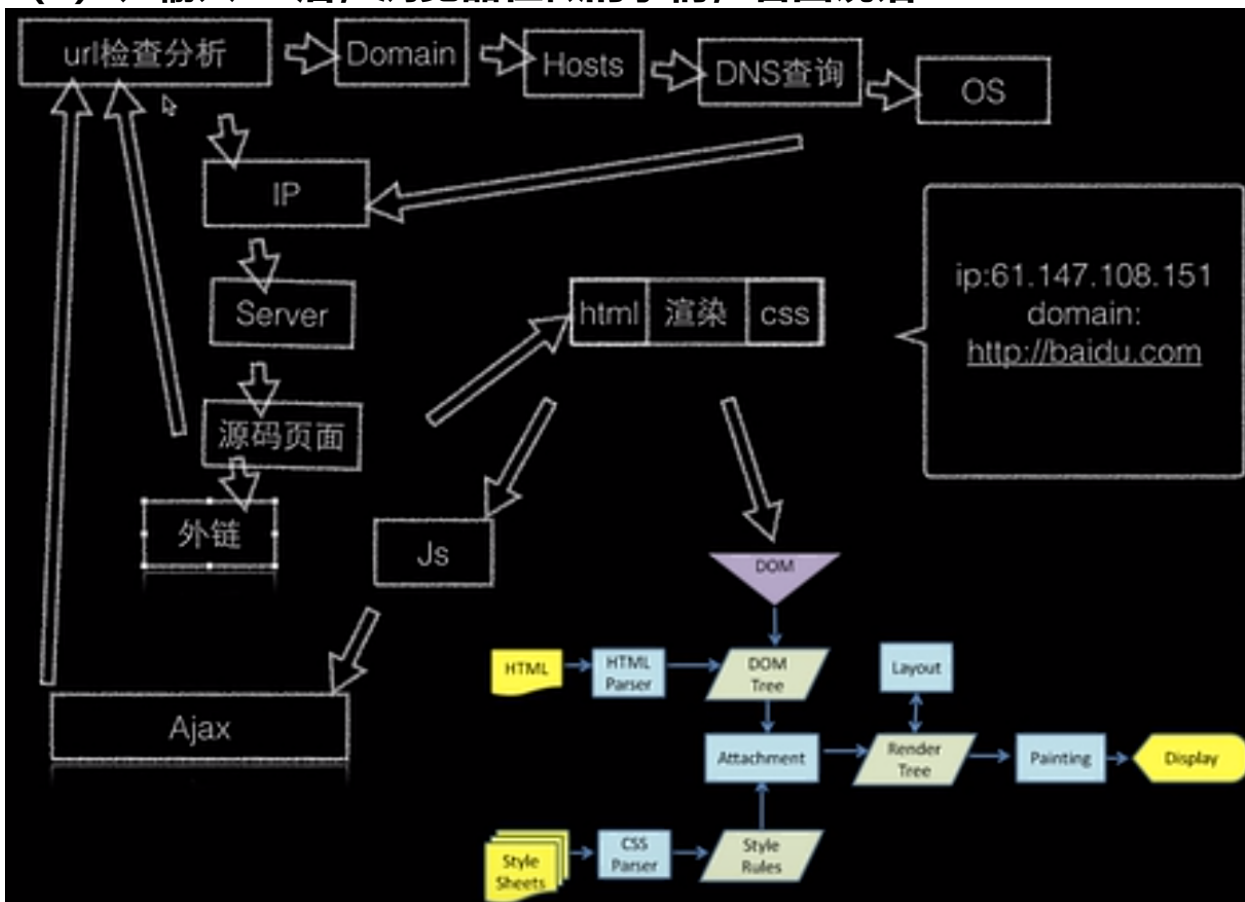
(1)、http和https

- HTTP 明文传输，数据都是未加密的，安全性较差，HTTPS (SSL+HTTP) 数据传输过程是加密的，安全性较好。
- 使用 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。证书颁发机构如：Symantec、Comodo、GoDaddy 和 GlobalSign 等。
- HTTP 页面响应速度比 HTTPS 快，主要是因为 HTTP 使用 TCP 三次握手建立连接，客户端和服务端需要交换 3 个包，而 HTTPS除了 TCP 的三个包，还要加上 ssl 握手需要的 9 个包，所以一共是 12 个包。
- 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。
- HTTPS 其实就是建构在 SSL/TLS 之上的 HTTP 协议，所以，要比较 HTTPS 比 HTTP 要更耗费服务器资源。

下面是常见的HTTP状态码：

- 200 - 请求成功
- 301 - 资源（网页等）被永久转移到其它URL
- 404 - 请求的资源（网页等）不存在
- 500 - 内部服务器错误

(2)、输入url后，浏览器在做的事情，看图说话



26、事件委托和事件冒泡

阻止事件冒泡

```
document.getElementById("c").onclick = function(e) {
    e.stopPropagation();
    console.log("c");
}
```


事件委托：

原理：事件委托是最好理解的那个，我们要给每一个按钮绑定一个事件，但是这样遍历，太消耗性能了，于是我们直接给父元素绑定即可完成。解决了以下几个问题：

- 1、遍历带来的性能问题。
- 2、button如果是动态添加的，那么必须用事件委托。
- 3、由于事件委托是通过事件冒泡实现的，所以如果子级的元素(e.stopPropagation())阻止了事件冒泡，那么事件委托也将失效！

```
1      <div class="btn-layout">
2          <button>按钮</button>
3          <button>按钮</button>
4          <button>按钮</button>
5          <button>按钮</button>
6          <button>按钮</button>
7          <button>按钮</button>
8      </div>
9      <script>
10         document.getElementsByClassName("btn-layout")[0].onclick = function(e){
11             console.log(e);
12         }
13     </script>
```

事件冒泡:

事件冒泡，就是点击最里面的元素，会触发父元素的方法，如下：

```
1      <div id="a">
2          最外层的元素
3      <div id="b">
4          中间的元素
5          <div id="c">
6              最里面的元素
7          </div>
8      </div>
9  </div>
10 <script>
11     document.getElementById("a").onclick = function(){
12         console.log("a");
13     }
14     document.getElementById("b").onclick = function(){
15         console.log("b");
16     }
17     document.getElementById("c").onclick = function(){
18         console.log("c");
19     }
20     /*document.getElementById("a").addEventListener('click', function(){
21         console.log('最外层元素 捕获阶段');
22     },true);
23     document.getElementById("b").addEventListener('click', function(){
24         console.log('中间层元素 捕获阶段');
25     },true);
26     document.getElementById("c").addEventListener('click', function(){
27         console.log('最里层元素 捕获阶段');
```

事件捕获

事件捕捉阶段：事件开始由顶层对象触发，然后逐级向下传播，直到目标元素；

27、拷贝

浅拷贝：

由于**数组内部属性值为引用对象**，因此使用slice和concat对对象数组的拷贝，整个拷贝还是浅拷贝，拷贝之后数组各个值的指针还是指向相同的存储地址。

因此，**slice()和concat()这两个方法**，仅适用于**对不包含引用对象的一维数组的深拷贝**

对于array对象的slice函数，返回一个数组的一段。（仍为数组）

```
arrayObj.slice(start, [end])
```

参数：

arrayObj 必选项。一个 Array 对象。

start 必选项。arrayObj 中所指定的部分的开始元素是从零开始计算的下标。

end可选项。arrayObj 中所指定的部分的结束元素是从零开始计算的下标。

说明：

slice 方法返回一个 Array 对象，其中包含了 arrayObj 的指定部分。

slice 方法一直复制到 end 所指定的元素，但是不包括该元素。

如果 start 为负，将它作为 length + start处理，此处 length 为数组的长度。

如果 end 为负，就将它作为 length + end 处理，此处 length 为数组的长度。

如果省略 end，那么 slice 方法将一直复制到 arrayObj 的结尾。

如果 end 出现在 start 之前，不复制任何元素到新数组中。

深拷贝：

1、JSON.stringify和JSON.parse实现深拷贝

JSON.stringify把对象转成字符串，再用JSON.parse把字符串转成新的对象

2、使用for循环创建新的数组

当然,如果是不那么复杂的数组，你可以声明一个新数组，自己写一个for循环拷贝过去。

29、重绘回流

※ DOM的重绘和回流 Repaint & Reflow

☑ 重绘：元素样式的改变（但宽高、大小、位置等不变）

如 outline, visibility, color、background-color等

☑ 回流：元素的大小或者位置发生了变化（当页面布局和几何信息发生变化的时候），触发了重新布局，导致渲染树重新计算布局和渲染

30、react版本

```
"react": "~16.6.3",  
"react-copy-to-clipboard": "^5.0.2",  
"react-dnd": "^2.6.0",  
"react-dnd-html5-backend": "^2.6.0",  
"react-dom": "~16.6.0",  
"react-dom-factories": "^1.0.2",  
"react-lazyload": "^2.6.9",  
"react-loadable": "^5.5.0",  
"react-router": "^4.3.1",  
"react-router-dom": "^4.3.1",  
"react-sortable-hoc": "^1.11.0",  
"react-virtualized": "9.21.2",
```

31、Hook

1、`setState` 函数用于更新 state。它接收一个新的 state 值并将组件的一次重新渲染加入队列

2、`useContext`

接收一个 context 对象（`React.createContext` 的返回值）并返回该 context 的当前值。当前的 context 值由上层组件中距离当前组件最近的 `<MyContext.Provider>` 的 `value` prop 决定。