26 Node的应用场景

- 特点：
  - 1、它是一个 Javascript 运行环境
  - 2、依赖于 Chrome V8 引擎进行代码解释
  - 3、事件驱动
  - 4、非阻塞 I/O
  - 5、单进程， 单线程
- 优点：
  - 高并发（最重要的优点)
- 缺点：
  - 1、只支持单核 CPU ，不能充分利用 CPU
  - 2、可靠性低，一旦代码某个环节崩溃，整个系统都崩溃

27 谈谈你对AMD 、CMD的理解

- CommonJS 是服务器端模块的规范， Node.js 采用了这个规范 。 CommonJS 规范加载模块是同步的，也就是说， 只有加载完成，才能执行后面的操作 。 AMD 规范则是非同步加载模块， 允许指定回调函数
- AMD 推荐的风格通过返回一个对象做为模块对象， CommonJS 的风格通过对 module.exports 或 exports 的属性赋值来达到暴露模块对象的目的

es6模块 CommonJS、AMD、CMD

- CommonJS 的规范中，每个 JavaScript 文件就是一个独立的模块上下文 （ module context ）， 在这个上下文中默认创建的属性都是私有的 。也就是说，在一个文件定义的变量（还包括函数和类)， 都是私有的，对其他文件是不可见的。
- CommonJS 是同步加载模块,在浏览器中会出现堵塞情况，所以不适用
- AMD 异步， 需要定义回调 define 方式
- es6 一个模块就是一个独立的文件，该文件内部的所有变量，外部无法获取 。如果你希望外部能够读取模块内部的某个变量，就必须使用 export 关键字输出该变量 es6 还可以导出类 、方法， 自动适用严格模式

28 那些操作会造成内存泄漏

- 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在
- `setTimeout` 的第一个参数使用字符串而非函数的话，会引发内存泄漏
- 闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

29 web开发中会话跟踪的方法有哪些

- `cookie`
- `session`
- `url` 重写
- 隐藏 `input`
- `ip` 地址

30 介绍js的基本数据类型

- `Undefined`、`Null`、`Boolean`、`Number`、`String`

31 介绍js有哪些内置对象

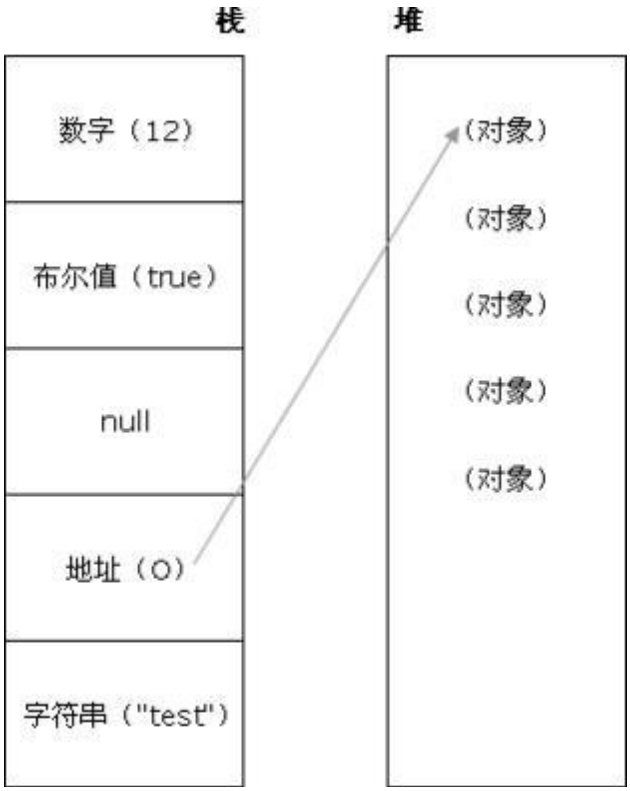
- `Object` 是 `JavaScript` 中所有对象的父对象
- 数据封装类对象：`Object`、`Array`、`Boolean`、`Number` 和 `String`
- 其他对象：`Function`、`Arguments`、`Math`、`Date`、`RegExp`、`Error`

32 说几条写JavaScript的基本规范

- 不要在同一行声明多个变量
- 请使用 `===/!==` 来比较 `true/false` 或者数值
- 使用对象字面量替代 `new Array` 这种形式
- 不要使用全局函数
- `Switch` 语句必须带有 `default` 分支
- `If` 语句必须使用大括号
- `for-in` 循环中的变量 应该使用 `var` 关键字明确限定作用域，从而避免作用域污

33 JavaScript有几种类型的值

- 栈：原始数据类型（ Undefined ， Null ， Boolean ， Number 、 String ）
- 堆：引用数据类型（对象 、数组和函数)
- 两种类型的区别是：存储位置不同；
- 原始数据类型直接存储在栈(stack)中的简单数据段， 占据空间小 、大小固定，属于被频繁使用数据，所以放入栈中存储；
- 引用数据类型存储在堆(heap)中的对象, 占据空间大 、大小不固定,如果存储在栈中，将会影响程序运行的性能；引用数据类型在栈中存储了指针，该指针指向堆中该实体的起始地址 。当解释器寻找引用值时，会首先检索其
- 在栈中的地址， 取得地址后从堆中获得实体



34javascript创建对象的几种方式

> javascript 创建对象简单的说,无非就是使用内置对象或各种自定义对象，当然还可以用 JSON ；但写法有很多种，也能混合使用

- 对象字面量的方式

```js
person={firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};
```

- 用 function 来模拟无参的构造函数

```js
function Person(){}
    var person=new Person();//定义一个function，如果使用new"实例化",该function可
        person.name="Mark";
        person.age="25";
        person.work=function(){
        alert(person.name+" hello...");
}
person.work();
```

- 用 function 来模拟参构造函数来实现（用 this 关键字定义构造的上下文属性)

```js
function Pet(name,age,hobby){
        this.name=name;//this作用域:  当前对象
        this.age=age;
        this.hobby=hobby;
        this.eat=function(){
            alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");
        }
}
var maidou =new Pet("麦兜",25,"coding");//实例化、创建对象
maidou.eat();//调用eat方法
```

- 用工厂方式来创建（内置对象)

```js
var wcDog =new Object();
     wcDog.name="旺财";
     wcDog.age=3;
     wcDog.work=function(){
       alert("我是"+wcDog.name+",汪汪汪.....");
     }
     wcDog.work();
```

- 用原型方式来创建

```js
function Dog(){}
Dog.prototype.name="旺财";
Dog.prototype.eat=function(){
    alert(this.name+"是个吃货");
}
var wangcai =new Dog();
```

```
wangcai.eat();
```

- 用混合方式来创建

```js
function Car(name,price){
    this.name=name;
    this.price=price;
}
Car.prototype.sell=function(){
    alert("我是"+this.name+", 我现在卖"+this.price+"万元");
}
var camry =new Car("凯美瑞",27);
camry.sell();
```

35 eval是做什么的

- 它的功能是把对应的字符串解析成 `JS` 代码并运行
- 应该避免使用 `eval` ，不安全，非常耗性能（ `2` 次，一次解析成 `js` 语句，一次执行)
- 由 `JSON` 字符串转换为JSON对象的时候可以用 `eval, var obj =eval('('+ str +')')`

36 null， undefined 的区别

- `undefined` 表示不存在这个值。
- `undefined` :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义 。当尝试读取时会返回 `undefined`
- 例如变量被声明了，但没有赋值时，就等于 `undefined`
- `null` 表示一个对象被定义了，值为“空值”
- `null` ：是一个对象(空对象, 没有任何属性和方法)
- 例如作为函数的参数，表示该函数的参数不是对象；
- 在验证 `null` 时，一定要使用 `===` ， 因为 `==` 无法分别 `null` 和 `undefined`

37 ["1", "2", "3"].map(parseInt) 答案是多少

- [1, NaN, NaN] 因为 parseInt 需要两个参数 (val, radix) ， 其中 radix 表示解析时用的基数。
- map 传了 3 个 (element, index, array) ，对应的 radix 不合法导致解析失败。

38javascript 代码中的"use strict";是什么意思

- use strict 是一种 ECMAscript 5 添加的（严格）运行模式,这种模式使得 Javascript 在更严格的条件下运行 ,使 JS 编码更加规范化的模式,消除 Javascript 语法的一些不合理、不严谨之处，减少一些怪异行为

39 JSON 的了解

- JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式
- 它是基于 JavaScript 的一个子集 。数据格式简单, 易于读写, 占用带宽小
- JSON 字符串转换为JSON对象:

```js
var obj =eval( '('+ str +')');
var obj = str.parseJSON();
var obj = JSON.parse(str);
```

- JSON 对象转换为JSON字符串:

```
var last=obj.toJSONString();
var last=JSON.stringify(obj);
```

40js延迟加载的方式有哪些

- defer 和 async 、动态创建 DOM 方式（用得最多）、按需异步载入 js

41 同步和异步的区别

- 同步：浏览器访问服务器请求，用户看得到页面刷新， 重新发请求,等请求完， 页面刷新，新内容出现，用户看到新内容,进行下一步操作
- 异步：浏览器访问服务器请求，用户正常操作， 浏览器后端进行请求 。等请求完， 页面不刷新，新内容也会出现，用户看到新内容

42 渐进增强和优雅降级

- 渐进增强：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。
- 优雅降级：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容

43 defer和async

- defer 并行加载 js 文件，会按照页面上 script 标签的顺序执行
- async 并行加载 js 文件，下载完成立即执行，不会按照页面上 script 标签的顺序执行

44 说说严格模式的限制

- 变量必须声明后再使用
- 函数的参数不能有同名属性， 否则报错
- 不能使用 with 语句
- 禁止 this 指向全局对象

45 attribute和property的区别是什么

- attribute 是 dom 元素在文档中作为 html 标签拥有的属性；
- property 就是 dom 元素在 js 中作为对象拥有的属性。
- 对于 html 的标准属性来说， attribute 和 property 是同步的， 是会自动更新的
- 但是对于自定义的属性来说，他们是不同步的

46 谈谈你对ES6的理解

- 新增模板字符串（为 JavaScript 提供了简单的字符串插值功能)
- 箭头函数
- for-of （用来遍历数据—例如数组中的值。）
- arguments 对象可被不定参数和默认参数完美代替。
- ES6 将p romise 对象纳入规范，提供了原生的 Promise 对象。
- 增加了 let 和 const 命令，用来声明变量。
- 增加了块级作用域。
- let 命令实际上就增加了块级作用域。

- 还有就是引入 module 模块的概念

47 ECMAScript6 怎么写class么

- 这个语法糖可以让有 OOP 基础的人更快上手 js ， 至少是一个官方的实现了
- 但对熟悉 js 的人来说， 这个东西没啥大影响；一个 Object.creat() 搞定继承， 比 class 简洁清晰的多

48 什么是面向对象编程及面向过程编程， 它们的异同和优缺点

- 面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了
- 面向对象是把构成问题事务分解成各个对象， 建立对象的目的不是为了完成一个步骤， 而是为了描叙某个事物在整个解决问题的步骤中的行为
- 面向对象是以功能来划分问题， 而不是步骤

49 面向对象编程思想

- 基本思想是使用对象， 类， 继承， 封装等基本概念来进行程序设计
- 优点
  - 易维护
    - 采用面向对象思想设计的结构， 可读性高， 由于继承的存在， 即使改变需求，那么维护也只是在局部模块，所以维护起来是非常方便和较低成本的
  - 易扩展
  - 开发工作的重用性 、继承性高， 降低重复工作量。
  - 缩短了开发周期

50 对web标准、可用性、可访问性的理解

- 可用性（Usability)： 产品是否容易上手 ，用户能否完成任务，效率如何， 以及这过程中用户的主观感受可好， 是从用户的角度来看产品的质量 。可用性好意味着产品质量高， 是企业的核心竞争力
- 可访问性（Accessibility)： Web内容对于残障用户的可阅读和可理解性
- 可维护性（Maintainability)：一般包含两个层次，一 是当系统出现问题时，快速定位并解决问题的成本，成本低则可维护性好 。二是代码是否容易被人理解， 是否容易修改和增强功能。

51 如何通过JS判断一个数组

- instanceof 方法
  - instanceof 运算符是用来测试一个对象是否在其原型链原型构造函数的属性

```js
var arr = [];
arr instanceof Array; // true
```

- constructor 方法
  - constructor 属性返回对创建此对象的数组函数的引用，就是返回对象相对应的构造函数

```js
var arr = [];
arr.constructor == Array; //true
```

- 最简单的方法
  - 这种写法， 是 jQuery 正在使用的

```js
Object.prototype.toString.call(value) == '[object Array]'
// 利用这个方法， 可以写一个返回数据类型的方法
var isType = function (obj) {
     return Object.prototype.toString.call(obj).slice(8,-1);
}
```

- ES5 新增方法 isArray()

```js
var a = new Array(123);
var b = new Date();
console.log(Array.isArray(a)); //true
console.log(Array.isArray(b)); //false
```

52 谈一谈let与var的区别

- let 命令不存在变量提升， 如果在 let 前使用，会导致报错
- 如果块区中存在 let 和 const 命令，就会形成封闭作用域

- 不允许重复声明， 因此，不能在函数内部重新声明参数

53 map与forEach的区别

- `forEach` 方法， 是最基本的方法，就是遍历与循环， 默认有3个传参：分别是遍历的数组内容 `item` 、数组索引 `index` 、和当前遍历数组 `Array`
- `map` 方法， 基本用法与 `forEach` 一致，但是不同的， 它会返回一个新的数组，所以在callback需要有 `return` 值， 如果没有，会返回 `undefined`

54 谈一谈你理解的函数式编程

- 简单说， "函数式编程"是一种"编程范式"（programming paradigm)， 也就是如何编写程序的方法论
- 它具有以下特性： 闭包和高阶函数 、惰性计算 、递归 、函数是"第一等公民"、只用"表达式"

55 谈一谈箭头函数与普通函数的区别?

- 函数体内的 `this` 对象，就是定义时所在的对象， 而不是使用时所在的对象
- 不可以当作构造函数，也就是说，不可以使用 `new` 命令， 否则会抛出一个错误
- 不可以使用 `arguments` 对象，该对象在函数体内不存在 。如果要用， 可以用 `Rest` 参数代替
- 不可以使用 `yield` 命令， 因此箭头函数不能用作 `Generator` 函数

56 谈一谈函数中this的指向

- this的指向在函数定义的时候是确定不了的， 只有函数执行的时候才能确定this到底指向谁， 实际上this的最终指向的是那个调用它的对象
- 《javascript语言精髓》 中大概概括了4种调用方式：
- 方法调用模式
- 函数调用模式
- 构造器调用模式

```
graph LR
A-->B
```

- apply/call调用模式

57 异步编程的实现方式

- 回调函数
  - 优点：简单 、容易理解
  - 缺点：不利于维护，代码耦合高
- 事件监听(采用时间驱动模式， 取决于某个事件是否发生)：
  - 优点：容易理解， 可以绑定多个事件，每个事件可以指定多个回调函数
  - 缺点：事件驱动型， 流程不够清晰
- 发布/订阅(观察者模式)
  - 类似于事件监听，但是可以通过 ‘消息中心’， 了解现在有多少发布者， 多少订阅者
- Promise对象
  - 优点：可以利用then方法， 进行链式写法；可以书写错误时的回调函数；
  - 缺点：编写和理解，相对比较难
- Generator函数
  - 优点： 函数体内外的数据交换 、错误处理机制
  - 缺点：流程管理不方便
- async函数
  - 优点： 内置执行器 、更好的语义 、更广的适用性 、返回的是Promise、结构清晰。
  - 缺点：错误处理机制

58 对原生Javascript了解程度

- 数据类型 、运算 、对象 、Function、继承 、闭包 、作用域 、原型链 、事件 、 `RegExp` 、 `JSON` 、 `Ajax` 、 `DOM` 、 `BOM` 、内存泄漏 、跨域 、异步装载 、模板引擎 、前端 `MVC` 、路由 、模块化 、 `Canvas` 、 `ECMAScript`

59 Js动画与CSS动画区别及相应实现

- `CSS3` 的动画的优点

  - 在性能上会稍微好一些， 浏览器会对 CSS3 的动画做一些优化
  - 代码相对简单
- 缺点
  - 在动画控制上不够灵活
  - 兼容性不好
- JavaScript 的动画正好弥补了这两个缺点，控制能力很强， 可以单帧的控制 、变换， 同时写得好完全可以兼容 IE6 ， 并且功能强大 。对于一些复杂控制的动画，使用 javascript 会比较靠谱 。而在实现一些小的交互动效的时候，就多考虑考虑 CSS 吧

60 JS 数组和对象的遍历方式， 以及几种方式的比较

> 通常我们会用循环的方式来遍历数组 。但是循环是 导致js 性能问题的原因之一 。一般我们会采用下几种方式来进行数组的遍历

- for in 循环
- for 循环
- forEach
  - 这里的 forEach 回调中两个参数分别为 value ， index
  - forEach 无法遍历对象
  - IE不支持该方法； Firefox 和 chrome 支持
  - forEach 无法使用 break ， continue 跳出循环，且使用 return 是跳过本次循环

- 这两种方法应该非常常见且使用很频繁 。但实际上， 这两种方法都存在性能问题
- 在方式一中， for-in 需要分析出 array 的每个属性， 这个操作性能开销很大 。用在 key 已知的数组上是非常不划算的 。所以尽量不要用 for-in ， 除非你不清楚要处理哪些属性，例如 JSON 对象这样的情况
- 在方式2中，循环每进行一次，就要检查一下数组长度 。读取属性 （数组长度） 要比读局部变量慢， 尤其是当 array 里存放的都是 DOM 元素， 因为每次读取都会扫描一遍页面上的选择器相关元素， 速度会大大降低

61 gulp是什么

- gulp 是前端开发过程中一种基于流的代码构建工具， 是自动化项目的构建利器；它不仅能对网站资源进行优化， 而且在开发过程中很多重复的任务能够使用正确的工具自动完成

- Gulp的核心概念：流
- 流， 简单来说就是建立在面向对象基础上的一种抽象的处理数据的工具 。在流中，定义了一些处理数据的基本操作， 如读取数据， 写入数据等，程序员是对流进行所有操作的， 而不用关心流的另一头数据的真正流向
- gulp正是通过流和代码优于配置的策略来尽量简化任务编写的工作
- Gulp的特点：
  - 易于使用：通过代码优于配置的策略，gulp 让简单的任务简单， 复杂的任务可管理
  - 构建快速 利用 `Node.js` 流的威力 ，你可以快速构建项目并减少频繁的 `IO` 操作
  - 易于学习 通过最少的 `API` ， 掌握 `gulp` 毫不费力 ，构建工作尽在掌握：如同一系列流管道

62 说一下Vue的双向绑定数据的原理

- `vue.js` 则是采用数据劫持结合发布者-订阅者模式的方式， 通过 `Object.defineProperty()` 来劫持各个属性的 `setter` ， `getter` ，在数据变动时发布消息给订阅者，触发相应的监听回调

63 事件的各个阶段

- 1：捕获阶段 ---> 2： 目标阶段 ---> 3： 冒泡阶段
- `document` ---> `target` 目标 ----> `document`
- 由此， `addEventListener` 的第三个参数设置为 `true` 和 `false` 的区别已经非常清晰了
  - `true` 表示该元素在事件的“捕获阶段”（由外往内传递时）响应事件
  - `false` 表示该元素在事件的“冒泡阶段”（由内向外传递时）响应事件

64 let var const

let

- 允许你声明一个作用域被限制在块级中的变量 、语句或者表达式
- let绑定不受变量提升的约束， 这意味着let声明不会被提升到当前
- 该变量处于从块开始到初始化处理的“暂存死区”

var

- 声明变量的作用域限制在其声明位置的上下文中， 而非声明变量总是全局的
- 由于变量声明（以及其他声明）总是在任意代码执行之前处理的，所以在代码中的任意位置声明变量总是等效于在代码开头声明

const

- 声明创建一个值的只读引用 (即指针)
- 基本数据当值发生改变时，那么其对应的指针也将发生改变， 故造成 `const` 申明基本数据类型时
- 再将其值改变时，将会造成报错， 例如 `const a = 3 ; a = 5` 时 将会报错
- 但是如果是复合类型时， 如果只改变复合类型的其中某个 `Value` 项时， 将还是正常使用

65 快速的让一个数组乱序

```js
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(function(){
    return Math.random() - 0.5;
})
console.log(arr);
```

66 如何渲染几万条数据并不卡住界面

> 这道题考察了如何在不卡住页面的情况下渲染数据，也就是说不能一次性将几万条都渲染出来， 而应该一次渲染部分 `DOM` ，那么就可以通过 `requestAnimationFrame` 来每 `16 ms` 刷新一次

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <ul>控件</ul>
  <script>
    setTimeout(() => {
      // 插入十万条数据
      const total = 100000
      // 一次插入 20 条，如果觉得性能不好就减少
      const once = 20
      // 渲染数据总共需要几次
      const loopCount = total / once
      let countOfRender = 0
      let ul = document.querySelector("ul");
```

```
      function add() {
        // 优化性能，插入不会造成回流
        const fragment = document.createDocumentFragment();
        for (let i = 0; i < once; i++) {
          const li = document.createElement("li");
          li.innerText = Math.floor(Math.random() * total);
          fragment.appendChild(li);
        }
        ul.appendChild(fragment);
        countOfRender += 1;
        loop();
      }
      function loop() {
        if (countOfRender < loopCount) {
          window.requestAnimationFrame(add);
        }
      }
      loop();
    }, 0);
  </script>
</body>
</html>
```

67 希望获取到页面中所有的checkbox怎么做?

> 不使用第三方框架

```
var domList = document.getElementsByTagName('input')
var checkBoxList = [];
var len = domList.length;    //缓存到局部变量
while (len--) {    //使用while的效率会比for循环更高
    if (domList [len].type == 'checkbox' ) {
        checkBoxList.push(domList [len]);
    }
}
```

68 怎样添加、移除、移动、复制、创建和查找节点

创建新节点

```js
createDocumentFragment()    //创建一个DOM片段
createElement()    //创建一个具体的元素
createTextNode()    //创建一个文本节点
```

添加、移除、替换、插入

```js
appendChild()      //添加
removeChild()      //移除
replaceChild()     //替换
insertBefore()     //插入
```

查找

```js
getElementsByTagName()     //通过标签名称
getElementsByName()     //通过元素的Name属性的值
getElementById()     //通过元素Id，唯一性
```

69 正则表达式

> 正则表达式构造函数 `var reg=new RegExp("xxx")` 与正则表达字面量 `var reg=//` 有什么不同？匹配邮箱的正则表达式?

- 当使用 `RegExp()` 构造函数的时候，不仅需要转义引号（即 `\` ”表示”），并且还需要双反斜杠（即 `\\` 表示一个 `\`）。使用正则表达字面量的效率更高

邮箱的正则匹配：

```js
var regMail = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+((. [a-zA-Z0-9_-]{2,3}){1,2
```

70 Javascript中callee和caller的作用?

- `caller` 是返回一个对函数的引用，该函数调用了当前函数；
- `callee` 是返回正在被执行的 `function` 函数，也就是所指定的 `function` 对象的正文

> 那么问题来了？如果一对兔子每月生一对兔子；一对新生兔，从第二个月起就开始生兔子；假定每对兔子都是一雌一雄，试问一对兔子， 第n个月能繁殖成多少对兔子？(使用 callee 完成)

```js
var result= [];
  function fn(n){  //典型的斐波那契数列
     if(n==1){
          return 1;
     }else if(n==2){
             return 1;
     }else{
          if(result [n]){
                   return result [n];
          }else{
                  //argument.callee()表示fn()
                  result [n]=arguments.callee(n-1)+arguments.callee(n-2);
                  return result [n];
          }
     }
 }
```

71 window.onload和$(document).ready

> 原生 JS 的 window.onload 与 Jquery 的 $(document).ready(function(){}) 有什么不同？如何用原生JS实现Jq的 ready 方法？

- window.onload() 方法是必须等到页面内包括图片的所有元素加载完毕后才能执行。
- $(document).ready() 是 DOM 结构绘制完毕后就执行，不必等到加载完毕

```js
function ready(fn){
      if(document.addEventListener) {          //标准浏览器
          document.addEventListener( 'DOMContentLoaded', function() {
              //注销事件，避免反复触发
              document.removeEventListener('DOMContentLoaded',arguments.cal
              fn();             //执行函数
          }, false);
      }else if(document.attachEvent) {         //IE
          document.attachEvent( 'onreadystatechange', function() {
             if(document.readyState == 'complete') {
                 document.detachEvent( 'onreadystatechange', arguments.calle
```

```
                fn();          //函数执行
            }
        });
    }
};
```

72 addEventListener()和attachEvent()的区别

- addEventListener() 是符合W3C规范的标准方法; attachEvent() 是IE低版本的非标准方法
- addEventListener() 支持事件冒泡和事件捕获; - 而 attachEvent() 只支持事件冒泡
- addEventListener() 的第一个参数中,事件类型不需要添加 on ； attachEvent() 需要添加 'on'
- 如果为同一个元素绑定多个事件，addEventListener() 会按照事件绑定的顺序依次执行，attachEvent() 会按照事件绑定的顺序倒序执行

73 获取页面所有的checkbox

```js
var resultArr= [];
var input = document.querySelectorAll( 'input');
for( var i = 0; i < input.length; i++ ) {
    if( input [i].type == 'checkbox' ) {
        resultArr.push( input [i] );
    }
}
//resultArr即中获取到了页面中的所有checkbox
```

74 数组去重方法总结

方法一 、利用ES6 Set去重（ES6中最常用)

```js
function unique (arr) {
  return Array.from(new Set(arr))
}
var arr = [1,1, 'true', 'true',true,true,15,15,false,false, undefined,undefin
console.log(unique(arr))
 //[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {}, {}
```