JS第二月

服务器

简单来说服务器就是用来提供服务器的一台计算机。该计算机与普通计算机没什么本质区别，性能，安全性有所提高。

服务器也分软件服务器和硬件服务器。
硬件服务器指的就是看的见摸得着的计算机。
软件服务器指的就是服务器程序。
它们运行在硬件服务器上。

浏览器。

与服务器对应，也分软硬件。硬件指的就是计算机。软件指的是一阶段学习过的各种浏览器：chrome、firefox、ie、opera、safari等

域名

域名是IP地址的另一个名称，关系类似于手机号与备注。

域名服务器

根域名服务器全球一共13台，整个亚洲地区只有一台，在日本。

HTTP协议

HTTP协议是浏览器与服务器之间协商得到的一个通讯规则。
大致规定：浏览器发送信息该如何发送，服务器接受信息该如何解读。服务器返回信息该如何发送，浏览器接受信息该如何解读。

HTTP请求

协议只是一个概念，是两方或者多方约定共同遵守的规则，具体到HTTP协议就是规定：前后端之前如何通信。
HTTP请求，就是浏览器根据HTTP协议所执行的具体的一次行为。
HTTP请求由四部分组成。
请求首行
请求头
请求空行
请求正文

当 浏览器输入网址按下回车后都发生了什么

1、接受到地址字符串，并尝试转换为IP地址
2、转换IP步骤1： 取浏览器缓存中找IP地址的缓存，如果有，找到ip地址，并向目标服务器发起TCP连接请求，如果没有找到，就去系统中找系统缓存，如果还没有找到，就找网关缓存。如果还没有找到，就由操作系统发起一个调用远程DNS服务器的请求域名的信息。最终得到了IP地址。
3、找到IP地址之后，发送TCP链接请求，经过三次握手，TCP通道建立。此时发送HTTP请求。请求到达服务器。
4、服务器接受到请求。进行响应。将响应信息返回给浏览器。(响应头和响应正文)。第一次请求通常是请求页面，所以返回的是HTML字符串代码。
5、浏览器接受到返回的代码，经过HTTP协议中的规定的方式查看，知道只是HTML代码。于是按照HTML的解析规则开始解析。
6、渲染DOM树，渲染样式树。
7、从上至下执行，遇见外链的资源，会发起新的HTTP请求来请求这些资源。

HTTP请求的分类

根据HTTP请求的目的进行分类
最流行的两种：GET和POST。其他的还有PUT,DELETE
根据语义：

- GET是从服务器上得到内容，POST是往服务器上推送内容。
  + GET和SET区别：
    - GET请求没有请求正文。
    - GET请求数据放在QUERY部分。
    - GET请求携带的数据量受地址栏的长度限制。
    - GET请求的保密性差
    - GET请求触发方式多，便于分享。
      - 触发方式：【地址栏，a标签，img标签，link标签，script标签，video标签，audio标签，】表单，iframe，ajax。【】里的必定是get请求。
    - POST请求拥有请求正文。
    - POST请求数据没有上限。
    - POST请求保密性较强。
    - POST请求触发方式少，不便于分享。
    - 触发方式：表单，ajax

URL

统一资源定位符。
他就是一个字符串，用于定位网络上的唯一资源。
它符合一定的格式：

```
# URL
统一资源定位符
它就是一个字符串,用于定位网络上的唯一的资源.
它符合一定的格式:
    协议            域名              端口     路径         查询串            哈希
    https:   //   sub.example.com : 8080   /p/a/t/h  ?  query=string   #hash
协议: 指的是本次请求使用的协议类型
域名: 指的是本次请求的目标服务器域名
端口: 指的是操作系统中提供的虚拟端口. 一共65535个. 每一个程序占据一个或者多个.
    http协议使用的端口默认是80
    https协议使用的端口默认是443
路径: 指的是资源所在的目录层级以及文件名
查询串: 该部分可选,专门给GET请求使用的.
哈希: 该部分可选,专门给前端使用的字符串.
```

缓存

缓存是浏览器的一种存储机制，具体地存储方式由后台进行控制。
缓存分为两种：强制缓存与协商

BS 和 CS

BS 为 Browser / Server 浏览器对应服务器
CS 为 Client / Server 客户端对应服务器。

PHP服务器

phpStyle 是一个一键安装PHP服务器和MySQL数据库的安装包。安装完毕之后可以启动，此时，你的电脑就是一个服务器了。放在该目录下的文件，可以通过URL进行访问。

PHP

简介

PHP是一门嵌入式的语言，可以嵌入在别的语言中。比如HTML。比如HTML，CSS，JS等

代码书写位置。

<?php ?>

变量声明

$a = 10;

代码运行时机

JS是运行在浏览器的，php运行在服务器上。

语法

PHP每一句语句后需要加分号，如果不加分号，则视为重大错误。后面代码不予执行。

echo

echo既可以当做关键字使用，也可以当做函数使用。作用是将代码的运行结果以字符串的形式留在当前位置。

数组

PHP中没有对象这个JS中的数据结构。但是数组分两种：普通数组，关联数组。

- 普通数组就是JS中的Array
  - $arr = array(1,2,3,4,5,6)
- 关联数组就是JS中的Object
  - $arr = array(“key”=>”value”,”key1”=>”value1”)

> php语法集合
> mysql_connect(“localhost”,”root”,”root”)连接数据库
> mysql_select_db(“chatroom”) 选择数据表
> mysql_query($sql) 执行数据库语言
> mysql_fetch_assoc — 从结果集中取得一行作为关联数组
> setcookie(key,value,expires/maxAge,path,domain)
> `setcookie("login",1,time()+3600,"/");`
>
> 1、 key cookie的字段名
> 2、 value cookie的值
> 3、 expires/maxAge cookie的有效期。expires是一个较老的属性，属性值是一个日

> 期格式的，maxAge比较新，属性值是数字。
> 4、 path 生效路径
> 5、 domain生效域名

数据库

数据库就是存储数据的仓库。
为了操作数据方便，于是人们开发了各种不同的数据库应用程序。
常见的数据库应用程序：oracle,mysql,sqlserver,access,mongo等。
从类型上去区分：关系型（结构性），非关系型（非结构型）数据库。
关系型数据库：oracle,mysql,sqlserver,access
他们的特点是：
数据库结构由： 数据库=> 表 = > 字段
每一个表中的数据的字段都是一致的
非关系型数据库：mongo
他们的特点是： 数据库 => 集合 => 字段
每一个集合中的数据的字段可以不一致（第一条有5个字段，第二个有3个字段）

SQL语句

SQL语句是Structured Queryy Language结构化查询语言。
它是专门用于查询结构型数据库的语言。

- 增
  - insert into [table] (column1,column2,…) values(value1,value2,…);
  - 如果增加的是全部字段的数据，可以省略table后边括号的内容。
- 删
  - delete from [table] where column = value;
  - where是子句，用于更精确的描述删除的内容。
- 改
  - update [table] set column1 = value1,column2 = value2,… where column = value and column1 value1 and …;
- 查
  - select column1,column2,…from [table] where column = value;

连接数据库

- 连接数据库
  - mysql_connect(“localhost”,”root”,”root”)
- 选择数据库
  - mysql_select_db(“1908”)

AJAX

Asynchronous JavaScript And XML 异步的JS和XML。
XML是一种文档，也是一种数据格式。他叫做可拓展标记语言。与HTML很相似，也是由标签书写，但它的标签而已自定义。
现在XML在前端，已经被JSON代替了。但是称呼上依旧延续AJAX。
`AJAX的本质是通过JS悄悄地发起HTTP请求，取服务器请求数据，回到浏览器中进行处理数据，更新页面。所以有的人管AJ AX的特点叫做：局部刷新。`

AJAX的使用。

- 原生
  XMLHttpRequest 高级浏览器
  ActiveXObject IE7及以下

responseText是ajax后台的程序发送xmlhttp请求的时候, 后台程序接到请求会进行处理,处理结束后返回的一串数据。前端需要将它用JSON.parse()方法转换成一个对象之后使用。

xhr.readystate属性表示xhr对象的状态变化（注意不是请求状态码。该属性一共有 0 1 2 3 4 ）

> 0 未初始化
> 1 调用了open方法
> 2 已经接收到响应头
> 3 接受了一部分响应正文的数据
> 4 接受完毕响应正文的数据。

```
var xhr = null; //初始化
if(window.XMLHttpRequest){ //判定兼容性问题。
    xhr = new XMLHttpRequest();
}else if(window.ActiveXObject){
    xhr = new ActiveXOBject("Microsoft.XMLHttp")
}else{
    throw new Error("您的浏览器不支持AJAX，请下载最新版本的浏览器。")
}
```

```
xhr.open("GET","/regist.php",true);//建立TCP连接，true表示是否异步，一般为true

xhr.send("");//发送正文，如果发送方式是get，则不用填。

xhr.onreadystatechange = function(){
    if(xhr.readystate ==== 4 && xhr.status===200){
        console.log(xhr.respontext)
    }
} //绑定事件
```

封装AJAX的get和post

```
var QF = {
get(url, data, callback) {
    // 判定data具体的数据类型 如果是字符串则直接使用 如果是对象则将对象手动格式化成字符串
    var str = "";
    if (typeof data === "object") {
        for (var i in data) {
            str += i + '=' + data[i] + "&";
        }
        str = str.slice(0, -1);
    } else if (typeof data === "string") {
        str = data;
    }

    // 初始化
    var xhr = null;
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
```

```
        xhr = new ActiveXObject('Microsoft.XMLHttp');
    } else {
        throw new Error("您的浏览器不支持AJAX，请升级");
    }
    xhr.open("get", url + "?" + str, true);
    xhr.send();
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
            var obj = JSON.parse(xhr.responseText);
            callback && callback(obj);
        }
    }
},
post(url, data, callback) {
    // 判定data具体的数据类型 如果是字符串则直接使用 如果是对象则将对象手动格式化成字符串
    var str = "";
    if (typeof data === "object") {
        for (var i in data) {
            str += i + '=' + data[i] + "&";
        }
        str = str.slice(0, -1);
    } else if (typeof data === "string") {
        str = data;
    }

    // 初始化
    var xhr = null;
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        xhr = new ActiveXObject('Microsoft.XMLHttp');
```

```
    } else {
        throw new Error("您的浏览器不支持AJAX，请升级");
    }
    xhr.open("post", url , true);
    xhr.setRequestHeader("content-type", "application/x-www-form-urlencoded"); //这句是因为某些浏览器有兼容问题，不支持AJAX传输数据，只支持表单。这里就是把post请求头伪装成表单提交请求头。
    xhr.send(str);
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
            var obj = JSON.parse(xhr.responseText);
            callback && callback(obj);
        }
    }
}
```

}

JSON

JSON是一种文档，也是一种数据格式。在JS环境下，对象和数组都是JSON格式。
JSON是ES5中新增的一个对象，它的作用很简单，就是将字符串和对象互相转换。（一定是符合JSON格式的字符串才可以转为对象）
JSON格式：必须是双引号包裹属性名，双引号包裹字符串。每一个对象和数组的最末尾的逗号不能有逗号

`为什么要用JSON`

JSON是纯文本格式，是独立于语言和平台的。
生成和解析相对于XML而言要简单。
读写的速度更快。

- JSON.parse(str)
  - 返回值是格式化之后的对象
  - 如果str不符合JSON格式，将会报错。
- JSON.stringfy(obj)
  - 返回值 是格式化之后的字符串

```
> var s = {"name":"zishengy","password":123}
< undefined
> JSON.stringify(s,"undifiend",1);
< "{
   "name": "zishengy",
   "password": 123
  }"
>
```

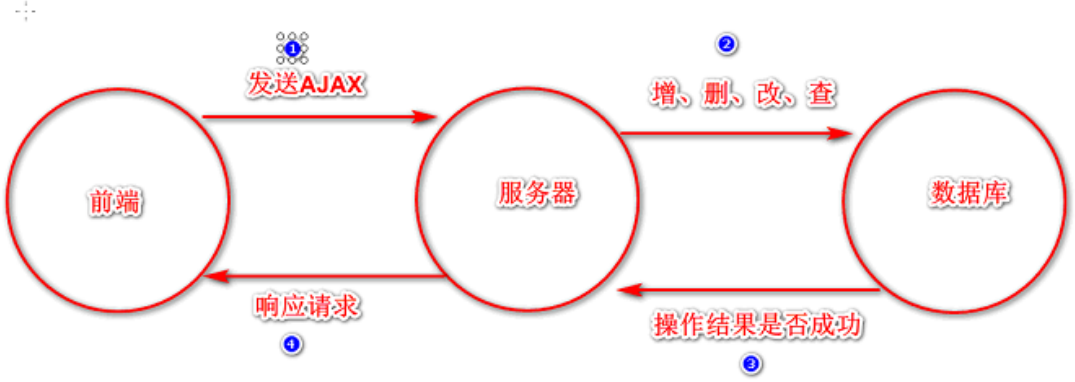
接口

接口在前后端之间，指的是前端给后端提供的返回数据的功能URL。接口其实就是后台语言运行完毕的处理结果，它会动态变化。

`URL分类：目前为止，url分两类：静态资源，接口`

- 静态资源
  - 指的是所有的文件资源，他们的特点是与后台代码无关。所以访问多少次都不会发生变化。比如HTML,图片，CSS,JS

    AJAX一定要发送到接口上,因为接口返回的是JSON字符串，当然，如果某一个静态文件中的字符串



`情况4：new关键字可以改变this的指向。`

this永远指向的是最后调用它的对象`

getAttribute方法

getAttribute方法是所有DOM元素都有的方法，用于获取DOM元素的自定义属性和HTML标准属性。

同样的，setAttribute用于设置属性。他需要两个参数。1、属性名 2、属性值。

removeAttribute 用于移除属性。
`

promise函数

ES6在增加了promise函数，允许我们将异步代码的回调写法，转换成从上往下的同步写法。promise只是解决了代码的书写风格，没有改变请求的发送方式。
promise是ES6中增加的构造函数之一。
使用方式：

- Promise接受一个函数作为参数，通常我们会在该函数内发送异步请求。
- 参数函数有两个参数,resolve,reject，这两个参数也是函数。resolve执行时会将promise实例状态从pendding修改为resolved。reject执行时会将promise实例的状态修改为rejected。状态一旦发生变化再也无法改变。
- promise的实例有then方法。该方法可以接收两个函数作为参数，
  - 第一个参数表示成功时（resolved）执行的函数。
    - 参数是resolve执行时传递的参数（只能传递一个参数）
  - 第二个参数表示失败时（rejected）执行的函数。
    - 参数是reject执行时传递的参数，只能传递一个
- then方法可以连续打点调用。当Promise实例为同一个时，第一个then会根据Promise的状态变化而决定执行第一个函数还是第二个函数，后续的then只有第一个参数函数生效。
  var p = new Promise(function(resolve,reject){

});

Promise提供了静态方法all

`静态方法是由构造函数本身调用的方法，即Promise.all()`

> 该方法接收一个数组作为参数，数组中的每一个成员都是Promise实例
> 该方法的作用：同时监听多个Promise实例的状态变化，返回值是一个新的Promise实例，如果被监听的Promise实例中有任何一个失败了，all返回的Promise实例状态变为rejected（则失败），被监听的Promise实例都resolved了，all返回的Promise实例状态才变为resolved（则成功）
> 参数数组中如果某一个不是promise实例，会默认使用promise.resolve将它转换为多个promise实例。

```
' let p = Promise.all([p1, p2]);
' p.then(function(arr) { // 成功时，唯一的参数是数组 数组的每一项是all接收的数组的每一项对应的传递的数据
'     console.log(arr)
'     console.log("两个都成功");
' })
' .catch(function(msg) { // 失败时， 唯一的参数是失败的第一个reject传递的数据
'     console.log(msg)
'     console.log("有一个失败了")
' });
```

Promise提供了静态方法race

> 该方法接受一个数组作为参数，每一个成员应当是Promise的实例。
> 该方法的作用：返回一个Promise实例，该实例的状态跟随数组中第一个发生状态变化的Promise实例。

```
Promise.race
该方法接收一个数组作为参数 每一个成员应当是Promise的实例
该方法的作用： 返回一个Promise实例，该实例的状态跟随数组中第一个发生状态变化的Promise实例。
let p = Promise.race([p1, p2])
p.then(function(msg) {
    console.log("成功", msg);
})
.catch(function(msg) {
    console.log("失败", msg);
});
```

Promise提供了静态方法resolve

该方法接受3种可能性的参数。

> 1、非Promise 比如数字，字符串，布尔值等
> 2、Promise实例
> 3、thenable 对象
> 返回值是一个Promise实例。
> 如果是非Promise作为参数，则Promise实例的状态为resolved并且接收到的值就是该参数。

```
如果是非Promise作为参数 则Promise实例的状态为resolved 并且接收到的值是该参数
let p = Promise.resolve(1);
p.then(function(msg) {
    console.log(msg);
})
```

如果是Promise实例作为参数，则跟随它的状态。

```
如果是Promise实例作为参数 则跟随它的状态
let p = Promise.resolve(sendPromise1());
p.then(function(data) {
    console.log("success", data)
})
.catch(function(msg) {
    console.log("failed", msg)
})
```

如果是thenable对象，

```
let p = Promise.resolve({
    then: function(resolve, reject) {
        let num = parseInt(Math.random() * 2);
        if (num) {
            resolve();
        } else {
            reject();
        }
    }
})
p.then(function() {
    console.log("成功");
})
.catch(function() {
    console.log("失败")
})
```

Promise提供了静态方法reject

参数是什么都可以，但通常是描述失败原因的字符串。

```
let p = Promise.reject("失败了，也不知道为什么");
p.then(function() {

})
.catch(function(a) {
    console.log(a)
})
```

域和跨域

域指的是区域，对于前端开发者来说，域指的是服务器在网络上占据的地址。
域名，指的是该地域的名称。
跨域值得是浏览器请求了服务器A的某个页面，在操作该页面时，发送了请求，请求服务器B的资源。这种现象就叫做跨域。

判定跨域
协议，域名，端口这三者任何一个不同，就视为跨域。

如何跨域？
jsonp是一种跨域方式。
利用script标签无视同源策略的特点，并且还能执行代码的特点。（注意：该方法需要要求接口返回的是函数的执行）

同源策略
浏览器针对跨域设置了一个策略：原则上只允许请求同一个服务器上的资源。
注：静态资源不受同源资源的限制。AJAX受同源策略的限制。

async 和 await

这两个是ES8新增的关键字，用于彻底解决异步编程的书写问题。

- async
  - 它放在函数的前面，用于表示函数内部有异步代码。
  - async函数的返回值是promise实例。
    - 该promise的状态取决于这个函数是否能顺利执行完毕。
    - 如果该函数正常执行完毕，则promise实例的状态变为resolved，该函数的返回值是promise实例的then接受的参数。
    - 如果该函数执行出错，则promise实例的状态变为rejected。出错信息是promise实例的catch接收的参数。
- await
  - 它是一个关键字。放在async函数内部，表示等待。
  - await后跟promise实例。如果不是promise实例，也会用promise.resolve将它转换为promise实例。
  - 当程序执行到await之后，会停止执行，等到await后面的promise实例为resolved时，继续往下执行。如果后面的promise实例为rejected，则整个async函数终止。

HTTP协议的无状态

前端解析域名，建立与服务器的TCP连接，发送请求，得到数据，断开连接。因为断开了连接，导致服务器与浏览器之间互相不知道对方的情况，这种情况就叫做无状态。
所谓的无状态指的就是连接最终会被断开。

COOKIE

cookie是HTTP协议中规定的一个请求头字段。用于在浏览器与服务器之间互相传递数据。

cookie优缺点

1、存储在浏览器
2、容量有限 4kb
3、 每一个域名下数量有限
4、一定会将cookie携带到服务器上。可能会浪费带宽。

coolie的流程

当服务器登陆的时候，前端发送请求登录，服务器验证。验证通过，设置响应头（set-cookie）。响应达到浏览器，浏览器检测响应头。发现set-cookie字段于是生成了cookie。当下一次请求的时候，会自动把对应的cookie带到服务器上。

- JS设置cookie
  - cookie格式
    - key = value;key1 = value1;
  - document.cookie = “a=1” 它一次只能设置一个cookie字段，而且不会对原有的其他cookie字段产生影响。

```
function setCookie(key, value) {
        if (typeof key === "object" && key != nul
l) {
            for (var i in key) {
                // setCookie(i, key[i]);
                document.cookie = i + "=" + key[i];
            }
        } else {
            document.cookie = key + "=" + value;
        }
    }
```

- 获取cookie

```
function getCookie(key) {
    var arr = document.cookie.split(";");
    for (var i = 0; i < arr.length; i++) {
        if (arr[i].split("=")[0].trim() === key) {
            console.log(arr[i].split("=")[1])
```

```
            return arr[i].split("=")[1];
        }
    }
}
```

本地存储

HTML5新增的除了一堆标签之外，还有一大堆新 功能。比如：canvas、事件推送、webworker、拖拽事件、离线缓存、本地存储、svg等。
本地存储是其中之一。用于代替cookie
本地存储是浏览器的一种存储数据的方式。分两种，一种是永久存储，一种是会话存储。

永久存储localstorage对象
localstorage.setItem(key,value) 它的作用是将数据以key名称，以value字符串的形式存储起来。存储时，如果是对象或者数组，请先使用JSON.stringify()把它转为字符串格式。
localstorage.getItem(key) 它的作用是根据key找到存储的数据。如果找的到就是字符串，找不到就是null。这个方法的返回值找到的字符串
localstorage.removeItem(key) 它的作用是根据key移除一项，没有返回值。
localstorage.clear() 它的作用是清空本地存储对象。

```
localStorage.setItem("a" , 123)
localStorage.setItem("obj",JSON.stringify({"a":"123","b":"nbb","c":"asf"}))
localStorage.setItem("arr",JSON.stringify([1,2,4,6,7,,5,43,3]))
var obj = localStorage.getItem("obj");
console.log(JSON.parse(obj))
var arr = localStorage.getItem("arr");
console.log(JSON.parse(arr))
localStorage.removeItem("a")
localStorage.clear()
```

会话存储sessionStorage对象
同上，唯一不同的是，sessionStorage建立的是临时连接，关闭窗口后自动清除sessionStorage

事件队列

1 执行栈
2 任务队列 宏任务队列 + 微任务队列
执行规则：将所有的代码放入执行栈 一条一条执行
遇见setIntervl setTimeout 就将它们的第一个函数交给window并开始计时 计时完成之后将函数放入宏任务队列

点击到事件 将事件函数放入宏任务队列
AJAX的回调函数也会放入宏任务队列
遇见Promise实例状态发生变化时会把then、catch 它们的参数函数放入微任务队列
直到所有的执行栈中的代码都执行完毕 执行栈空了 先看一下微任务队列中是否有任务 有就拿出来执行掉
开始循环
宏任务队列中的代码依次被拿出来放入执行栈中执行 在这个过程中如果有其它宏任务产生，则放入宏任务队列 有微任务产生放入微任务队列 执行栈完毕之后 再执行微任务队列中的代码
依次循环

闭包

所谓闭包，就是能访问私有作用域变量的函数

- 闭包的三大特点
  - 函数嵌套函数
  - 内部函数可以访问外部函数的变量
  - 函数和变量不会被回收
- 当函数执行的时候，内存中会发生什么？
  - 1、根据作用域，在内存中开辟一个空间（变量，函数什么的都存在与该空间里）
  - 2、声明提升
  - 3、执行里面的代码（该定义变量就定义变量，该定义函数就定义函数）
  - 4、当执行完毕之后，就销毁该空间以及空间中的变量。

继承

继承是面向对象的一个很重要的概念，用于说明类与类之间的管理。A类继承B类，这时候就说A类是子类，派生类。B类是父类，基类。
封装、继承、多态

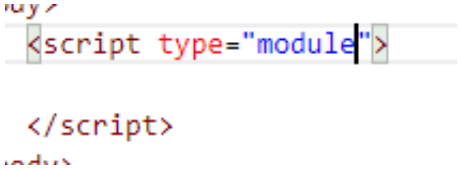
- 构造函数继承
  在子类中调用父类的构造函数并通过call，apply改变this指向，复用父类的构造函数。
- 原型式继承
  将子类的原型指向父类的实例。
- 组合式继承
  构造函数就继承加原型式继承。
- 寄生式继承。
  定义一个空类，将原型指向父类的原型，初始化一个实例，交给子类的原型。（得到的对象上没有任何的属性）
- **ES6中的继承**

```
class Student extends People {
        constructor(name, age, sex, studentNO) {
            // 必须调用super
            super(name, age, sex);
            this.studentNO = studentNO;
            console.log("hahahaha")
        }
    }
```

设计模式

设计模式：是一种编程的思想的总结。
设计模式的特点：1、编目分明 2、广为人知 3、效率有所提升。
工具库：指的是一些功能性函数的集合对象，这些方法之间关联性不大。
框架：框架是一套半成品代码。里面提供了各种功能，甚至于把开发一个程序的顺序都内定。这里边是由设计模式思想以及面向对象思想，算法， 数据结构支撑。内部也提供了许多的方法，这些方法之间是有关联。
架构：是开发大型程序的思想。

- **单例模式**：单个实例
  - 思想：无论如何对一个构造函数进行初始化，总是返回一个实例。
- **观察者模式**
  - 又叫做：消息管道模式、发布-订阅模式，自定义事件。
  - 主要功能：观察者模式可以很好的实现2个模块之间的解耦和跨模块进行通信
  - 耦合的定义：一处发生变化，另一处也发生变化。
  - 示例代码：
- **策略模式**
  - 为例避免太多的if else判定，简化了条件分支语句。
- **ES6中的模块化**
  - 定义模块
    - 一个js文件，就是一个模块。



```
import {isZhi} from "./b.js";
function findZ(start,end){
    var arr =[];
    for(var i = start;i<end;i++){
        if(isZhi(start)){
            arr.push(start)
        }
    }
    return arr;
}
export{
    findZ
}
```

    - 暴露方式
        - 第一种 export 变量|函数
        - 第二种 var 变量 = XXX; var 变量1 = XXX; export{ 变量, 变量1 }
        - 第三种 var 变量 = XXX; var 变量1 = XXX; export{ 变量 as default, 变量1 }
    - 引入方式
        - 第一种 import * as xxx from “路径”
        - 第二种 import {x,y,z} from “路径”
        - 第三种 import x from “路径” ,这种方法要求路径模块中必须有as default ，也只会引用

- **requirejs**
    - requirejs 是国外的一个异步模块定义规范的框架（ AMD ），它向全局暴露了3个函数：
        - define 用于定义模块
        - require 用于引入模块
        - requirejs 用于引入模块，与require全等。
    - 定义模块
        - 两个参数
            - 第一个参数是依赖的模块数组
            - 第二个参数是函数。函数的每一个参数是第一个数组的每一个依赖模块暴露的内容，依次往后排列。
    - 暴露接口
        - AMD的推荐暴露方式是通过return向外暴露内容。
    - 引入模块
        - 推荐通管局哦依赖数组（也有叫做依赖集合的）引入其他模块的内容。

> **模块化规范**：
> AMD async Module Definition 异步的模块定义规范
> 依赖后置， 等真的需要这个模块的时候才去加载模块

> CMD common Module Definition 同步的模块定义规范
> 依赖前置， 在代码运行之前，先把模块加载完毕

迭代器

- 迭代器的定义。
  - 能够提供一种方式，顺序遍历目标内部的数据。比如forEach
    - for和for in循环都必须知道目标结构的情况下才可以循环内部数据。

```
function each(target, handler) {
        // 判定target是数组还是对象
        // 方式1
        // console.log(target.constructor.name)
        // 方式2
        // console.log(target instanceof Array ? "数组": "");
        // console.log(target instanceof Object ? "对象": "")
        // 方式3
        console.log(Object.prototype.toString.call(target)); // 通过这种方式我们可以精确的判定目标的构造函数是哪个(自定义的构造函数的实例除外)
        if (Object.prototype.toString.call(target) === "[object Array]") {
            // 说明是数组
            for (var i = 0; i < target.length; i++) {
                handler(target[i], i)
            }
        } else if (Object.prototype.toString.call(target) === "[object Object]") {
            // 说明是对象
```

```
            for (var i in target) {
                handler(target[i], i)
            }
        } else if (target[0] && target.length) {
            // 说明是类数组
            for (var i = 0; i < target.length; i++) {
                handler(target[i], i)
            }
        }
    }
```

判定数据类型的方式

```
var arr = []
//第一种方式
arr.constructor.name
//第二种方式
arr instanceof Array
//第三种方式
    Object.prototype.toString.call(arr)
```

jQuery

- **简介**
  - jQuery是一个前端工具库，它主要解决了操作DOM元素。比如选择元素，创建元素，追加元素，移除元素，修改元素属性，克隆元素，替换元素。jQuery向全局中添加了$和jQuery 两个对象是同一个地址，为了防止$或者jQuery被占用
  - $的用法
    - 1、选择元素
    - 2、创建元素
    - 3、将原生元素转换为jQuery对象。

- 自定义jQuery对象
  - ```
    // $.noConflict();
    var my$ = jQuery.noConflict(true); // 解决纷争 解决变量名称冲突的问题 不带true 放弃$ 带参数true 表示两个都放弃
    ```

- **JQuery对象**
  - JQuery是一个函数，是一个类。调用jQuery($)方法返回一个jQuery的实例，叫做jQuery对象，jQuery对象是一个类数组对象。 数字下标和length属性都有，数字下标对应的是原生元素，length表示当前的jQuery对象中一共包含1多少个原生元素。
- **元素操作**
  - **选择元素** $(selector) 返回值是一个jQuery对象，jQuery对象才拥有一下所有的jQuery方法。
    - var $a = $(“.a.b”);
    - 将jQuery对象转化为原生对象的方式。$a[0],$a.get(0)。转化之后只能用原生的方法。
  - **创建元素** $(html) 返回值是一个jQuery对象
    - var $a = $(“< div>哈哈哈”)
  - **追加元素**
    - $(xxx).appendTo(parent);将选中的元素追加到父元素中。子是最后一个孩子
    - $(xxx).append(child) 将子元素添加到选中的父元素中。
    - $(xxx).prependTo() 子追加到父 ， 子是第一个儿子。
    - $(xxx).prepend() 父添加子，子是第一个儿子。
- **移除元素**
  - $xx.remove() 将选中的元素都移除。
- **修改元素属性**
  - addclass() 添加类
  - removeclass() 移除类
  - hasclass() 判定是否有类
  - toggleClass() 切换类
- 克隆元素
  - clone() 负责克隆元素。
- **CSS操作**
  - $xx.css() 设置行内样式。使用方式有三种
    - 如果不传递参数，会报错，如果传递一个参数，并且是css属性字符串，会返回该元素的计算后样式的值，值是字符串。

      ```
      var $a = $(" ").appendTo("body").css("border");
      ```
    - 如果传递两个参数，就表示设置属性。

      ```
      var $a = $(" ").appendTo("body").css("border", "211px solid red");
      ```
    - 如果传递了一个参数并且是对象，会将该对象的属性名当做css属性名，对应的属性值当做css属性值，可以一次设置多条属性。
      ```

```
var $a = $("<div></div>").appendTo("body").css({
    width: 10,
    height: 20,
    backgroundColor: "red"
});
```

- **操作HTML属性**
  - 设置： attr(key,value)
  - 获取: attr(key)
  - 删除: removeAttr(key)
- **事件绑定**
  - jQuery默认批量处理，如果选中了多个元素，默认会给所有元素都添加点击事件。
  - $xx.事件名称() 使用的是DOM2级绑定方式，所以可以绑定多个同类型事件。事件函数中的this，指的是原生元素。
  - $xx.off() 用来移除事件，如果不传递事件，会将所有的事件都清空。如果传递一个参数，并且是字符串，会清空该类型的参数。如果传递两个参数，第一个是事件类型字符串，第二个是函数名，将只移除该函数名的时间函数。
- **快捷尺寸**
  - $xxx.width() 内容宽
  - $xxx.innerWidth() 内容宽+左右padding
  - $xxx.outerWidth() 内容宽+左右padding+左右border
  - $xxx.outerWidth(true) 内容宽+左右padding+左右border+左右maigin
  - $xxx.offset() 返回的是该元素到页面的距离组成的对象，对象两个属性，left,top
  - $xxx.position() 返回的是该元素定位信息组成的对象，对象两个属性，left,top
- **委托模式**
  -
```
// on方法也是可以绑定事件的
// $tbody.on("click", function() {
//     console.log("点击事件发生")
//     console.log(this)
// });
// $tbody.on("mousemove", function() {
//     console.log("移动事件发生")
//     console.log(this)
// });
```
  -
```
$tbody.on('click', "td:last-child", function() {
    console.log("点击事件发生");
    console.log(this)
});
// 这种使用方式就是委托模式
```
- **迭代器**
  - each方法接受一个函数作为参数，会循环每一个选中的元素并执行这些函数，函数的第一个参数是序号，第二个参数是当前循环到的元素。
  - each方法这个迭代器是jQuery的原型上的方法，是给jQuery的实例调用的。
  - $.each是静态方法，接受两个参数，第一个参数是被迭代的对象或者数组，第二个参数是函数。

```
$.each([1, 2, 3, 4, 5], function(index, value) {
    console.log(index, value)
});
$.each({a: 1, b: 2, c: 3, d: 4, e: 5}, function(index, value) {
    console.log(index, value);
})
```

- **动画方法**
  - fadeIn() 淡入 .第一个参数是动画时间，第二个参数是动画的透明度目标值，第三个参数是动画曲线，第四个是回调函数。
  - fadeOut() 淡出
  - fadeToggle() 切换
  - slideDown() 向下侧滑
  - slideUp() 向上侧滑
  - slideToggle() 切换侧滑
  - show() 显示 hide和show方法用于控制元素的隐藏和显示，如果没有参数则立即执行，如果有参数则动画完成。
  - hide() 隐藏
  - animate()

```
$("div:nth-child(1)").animate({
    left: 1000
}, 1000, "linear", function() {
    console.log("动画结束")
})
```

- **节点关系**
  - children() 选中所有的亲儿子元素
  - find() 选中后代元素
  - next() 下一个兄弟元素
  - prev() 前一个兄弟元素
  - prevAll() 前面所有的兄弟元素
  - siblings() 选中除了自己的所有兄弟元素
- **表单序列号**
  - 指的是将表单中的数据，按照一定格式序列化成字符串或者其他内容
  - 好处：解除了HTML结构与JS之间的耦合。快速的帮助开发者整理数据。
  - `var str = $("#form").serialize()` 收集表单的数据转换成字符串。
    + `var str = $("#form").serializeArray()` 收集表单的数据存在数组里。数组的每个成员都是一个对象，对象里存放着填写的数据对。
- **index方法**
  - index方法返回当前元素，在自己的所有兄弟中的排名

```
$("li").click(function() {
    console.log($(this).index())
})
```

- **插件**
  - 给jQuery添加插件，就是在jQuery添加功能。
  - 因为jQuery是面向对象的，所以增加功能就是给原型或者类本身添加方法。

- 浅复制和深复制
  - $.extend(obj,obj1,obj2) 把obj1，和obj2身上的所有东西浅复制到obj，返回值是接受完数据的obj。
  - $.extend(true,obj,obj1,obj2) 把obj1和obj2身上的东西深复制到obj。返回值是接受完数据的obj。

```
// 如果想要给原型添加方法
$.extend($.prototype, {
    hello: function() {
        console.log(1);
    }
})
// 此时，每一个jQuery的实例都拥有了该方法
$("body").hello();


// 如果想要给$本身添加方法
$.extend($, {
    sayHi() {
        console.log("大家好，我是JQuery")
    }
});

$.sayHi()
```

- **AJAX**
  - $.ajax()方法接受一个对象作为参数，对象中应该有url,type,data,datatype,success,error等方法。

```
$.ajax({
    url:'',
    type:'get|post',
    data : {} | "query字符串",
    dataType : 'json', //
    success(data){  //成功执行

    },
    error(){  //失败执行
    },
    complete(){  //不管成功失败都执行
    },
    contentType:''  //内容类型，请求头。jq已经默认把该属性设置为表单类型了。
```

```
14.     }),
15.         async: ''  //为false时候会发送同步ajax请
    求。
16.         beforeSend(){ //该函数会在请求被发送之前执行
17.         },
18.         cache:false,    //是否缓存。
```

```
$("button[type = button]").click(function(){
    console.log($("form").eq(0).serialize())
    $.ajax({
        url : "/AJAX/register.php",
        type : "get",
        dataType: 'json',
        data :$("form").eq(0).serialize(),
        success(data){
            console.log(data)  //{error: 0, msg: "注册成功"}
            alert("注册成功")
        },
        error(){
            alert("注册失败")

        }
    })
})
```

```
$.get("/AJAX/register.php",$("form").eq(0).serialize(),function(data){
    console.log(data)
},"json")
```

Node

- Node是一个js的运行环境，与浏览器一样。
- Node是一个应用程序，需要安装。
- 官网：nodejs.org 中文官网是: nodejs.cn
- 进入官网之后，会自动看到两个下载面板，左侧的是稳定版本，右侧的是测试版本。
- 严重声明：因为脱离了浏览器所以BOM没有了，DOM没有了，只有ECMAScript语法了。
    - console,setinterval,settimeout 都有,location,window,history,document,alert等对象都没有了
- nodejs的特点
    - 1、单线程

    - 2、非阻塞I/O
    - 3、事件驱动
- NodeJS的使用场景：高并发，IO发
- IO指的是内存和硬盘数据交换的过程。
- NodeJS的不适用场景：计算多