# 2018.03.08

## 复习JavaScript

JavaScript历史: 1996,livescript

组成：ECMAScript: JavaScript(网景) JScript（微软）ActionScript（Adobe

ES5（变量var，流程控制if switch for (for 面向数组，for in 面向对象属性)while，运算符，常见类型，对象类型（定义：1.new、2.{字面量}，取值：1. obj.name 2.obj[“name”]，设值1. obj.name=”xx” 2.obj[“name”]=”xx”），数组类型(属性，方法)，函数类型(1.函数申明，2.函数表达式，3.new Function(形参1，形参2,....函数内容))） ES6(有兼容性问题)

BOM

复习JavaScript的内容详见：<JavaScript_self-learning.docx> 中的书签，js高阶部分

## 程序设计的两大分类：

### POP（面向过程）：process-oriented-programming

关注前因后果，以及每一项细节

### OOP（面向对象）：object-oriented-programming

关心结果和用什么来做

面向对象中两大核心概念：

类class：类就是对现实事物的一个“抽象”，把现实的特征（属性）和行为（方法），抽离出来。

实例：实例就是对某一种类型的具体实现，拥有类所有的特征和行为。

#### 创建类。

function Person(){} 类名必须大写。

构造函数：创建一个类时，必须执行的一个初始化函数。

使用function关键字

名称首字母大写

使用this定义属性，this为实例，可以在创建时，为实例定义一套初始化模板

没有return。

function Person(name,age){

this.name = name; //this指向的新创建的实例

this.age = age;

this.showName = function () {

...

}

}

new Person(“xxx”,18);

#### new做了什么？

1. 创建一个新的对象 new Object();
2. 把构造函数内部的this，指向新创建的实例
3. 把外部传入的参数，挂在属性上
4. 把创建的实例，返回到对应的变量上

## 方法过载

将属性和方法放在构造器中，其中方法只能使用匿名函数。而匿名函数导致每次新建的时候，都会创建的新的匿名方法（方法的结构一样）导致内存空间的极大浪费（内存中开辟新的空间，导致内存溢出）。

解决这个问题，引入原型

## 原型：

在每一个function函数对象上，都有一个prototype属性，代表原型。

### 原型是一个对象，使用来模拟类的结构。

注意：function首字母大写名称(){} --->类

通过类的名称，可以访问这个类，类也可以叫做函数对象

class 类名{ ---面向对象语言新建类的方法

构造函数(){}

方法1(){}

方法2(){}

}

作用：1. 解决方法过载， 2. 用于类型的扩展（！！）

用法：

定义：

类.prototype.属性

使用：

实例.新属性

function Cat(name){

this.name = name;

}

Cat.prototype.eat = function(){

alert(...);

}

var cat1 = new Cat(“hua”);

var cat2 = new Cat(“qian”);

cat1.eat === cat2.eat //true

### 属性访问的优先级

实例>构造函数>原型>继承的父类

### \_ \_proto\_ \_属性

\_\_proto\_\_是实例上的一个属性，用来建立和原型的链接

Cat.proptotype === cat.\_ \_proto\_ \_ // true

作用：让所有的实例，通过\_\_proto\_\_属性，共享类上所定义的方法（但是不能操作\_\_proto\_\_对象的内容）

常用属性：

constructor构造器

原型链：实例链接原型和父类相关特性的一种体现。

prototype：类级别属性，模拟类结构

\_\_proto\_\_：实例属性，用来建立和原型的链接

类.prototype === 实例.\_ \_proto\_ \_

扩展对象：

类.prototype.新属性 = 新值

类.prototype.新方法 = function(){}

扩展对象步骤：

1. 找到要扩展的类
2. 通过类的原型进行扩展属性或方法

2.1 定义扩展的名称

2.2 在名称下，定义属性值，或者方法内容

3. 测试

new

类（构造函数） 实例

构造属性 构造属性

原型信息 原型信息

constructor

prototype

\_ \_proto\_ \_

原型对象

构造函数

属性

方法

# 2018.03.09

## JSON

### 概念：

JavaScript Object Notation是一种数据传输格式，一般后台根据实际的请求来提供，前台以字符串方式来获取。

### 语法结构：

开始和结束，由”{” 和 “}” 包裹

内部结构以 键值对 形式存在

键值 之间以 : 冒号分隔

键（属性名）必须是字符串类型，使用**“双引号”**包裹

值（属性值）可以是js中的所有类型（字符串，数字，布尔，undefined，null，数组，对象）

每个属性之间以，逗号分隔，最后一个属性没有

获取属性值，通过 . 点的方式

var obj = {“name”: “张三”, “age”:18};

obj.name //取值

obj[‘name’] //动态取值

### 使用目的

简化前后端交互的数据结构，方便获取内容

### JSON转换

前后端交互，可以使用JSON，但是数据一定是JSON字符串

需求：假如从后台获取的返回的JSON字符串，在前端展示其内容

var jsonStr = ‘{“name”: “张三”, “age”:”18”}’;

//后台传来的数据是字符串，不能直接进行.点 获取

#### 3中操作JSON转换的方式：

1. 字符串转换成JSON

JSON.parse......JSON是window下的对象，就像是alert一样

var jsonObj = JSON.parse(jsonStr );

1. 把JSON对象转化成字符串

JSON.stringify

var jsonStr2 = JSON.stringify(jsonObj );

1. eval(代码字符串)，把参数内容，直接拷贝到当前值，有安全问题（如果字符串里可执行的代码，会自动执行）

eval(‘(’+jsonStr +’)’);

## 编译原理

程序的2种语言：

### 编译型语言：

Java编写完毕之后，不能直接使用，需要额外安装编译工具，编译后下载到本地才能使用。

### 解释型语言：

JS，编写完毕后，不需要额外翻译，在执行时，解释一行，执行一行。

### JS在执行时，2大步：

1. 预编译阶段，做变量和函数的提升（检查语法）
2. 解释执行阶段，逐行解析，逐行执行

变量提升：

声明提升，

当变量使用**var**进行声明赋值的时候，在预编译阶段，会把声明操作分离出来，提升到最前面。

函数提升：把函数声明，提到最前面

总结：

声明场景：变量的声明比函数提升更靠前。

实际场景：函数声明优先级更高（未赋值的情况）

赋值以后，始终执行赋值语句

var a = 1;

alert(a); //1

var fn = function(){

alert(a); //1

a=2;

alert()a; //2

};

var a = 1;

alert(a); //1

var fn = function(){

alert(a); //undefined

var a=2;

alert()a; //2

};

### JSON 和JS字面量

JSON:

符合JSON语法的对象（前后端交互格式！！）

{“name”:”xxx”}

JS字面量：

JS内置对象

{name:”xxx”}

区分：使用JSON.parse，如果是字面量，会报错。

## 作用域

在js程序执行的过程中，会创建一个上下文空间，来存储相关的资源（变量和函数），主要分为3种：

1. 全局作用域（window对象）---在函数外部创建的变量和函数所在的上下文空间

全局创建的变量，就是window对象的属性

全局创建的函数，就是window对象的方法

慎用，全局污染。

1. 局部作用域（函数内部）---在函数执行时，会自动为函数创建一个独立的作用域，用来保证局部执行需要
2. 块级作用域（js没有块级作用域）---块级作用域直接定义的变量和方法都会直接挂到全局（浪费资源），解决方法2---ES6: 将var换成 let（for,if,while里）

解决方法1---ES5: IIFE：Immediately Invoked Function Expression

作用域链：在当前作用域中，可以访问自己作用域的资源，同时可以**向上**访问所有相关相关层级的资源。

方向：由内向外

### IIFE：Immediately Invoked Function Expression

一个函数定义后，立即执行函数内容，所有使用的资源，在执行后自动回收

作用：优化全局资源，既不创建全局变量，也不创建全局函数；没有名字（没有指针指向）用完即回收。

语法结构：IIFE主要使用匿名函数来打包需要执行的代码

函数在定义时，没有名称，需要把函数转换成函数表达式。

（function(){

......

}）（执行函数1，执行函数2....）

方法1：（可以是！可以是+ 可以是-）

-function(){

var a = 100;

alert(a);

}()

foo();

方法2：

(function(){

alert(‘匿名函数 iife’)

})();

//解决块级作用域

(function(){

for(var i=0; i<10; i++){

var count....

}

})();

## 闭包closure

闭包：

是一个函数在创建的时候允许函数内部操作函数外部的变量时所创建的作用域。

场景：

1. 函数内部访问函数外部创建的作用域，就是闭包。
2. 函数外部资源也可以通过闭包访问内部资源的方式。

(function(){

var d = 10;

window.count = function(){

....

}

})();

写法：在函数内部，嵌套函数。

使用IIFE

把最内层函数或者需要全局化的变量，直接暴露在window对象的属性上

最外层window，就可以根据暴露的内容，进入内部作用域

应用场景：

写插件(JQuery也是这样用)，模块化

目的：

私有化变量或方法资源

延长变量在作用域的回收时间

问题：

如果都使用闭包，或者相关资源都不会回收，那么也会导致内存泄漏

在循环中，用定时器，输出1-10

问题：在定时器中，访问了外部作用域中的变量，产生了一个闭包。

也受了没有块级作用域影响

解决办法：let或者iife

# 2018.03.11

## 回顾

JSON：是一种轻量级的数据交换格式，是目前 前后端最常使用的一种格式。

语法：{“”:””, “”:15}

{}代表对象、[]代表数组

转换：

把JSON字符串转变成JSON对象

JSON.parse(JSON字符串)

把JSON对象转变为JSON字符串

JSON.stringify(JSON对象)

## this

### 全局this

在函数外部使用的this，指向window

var a = 10; ---> alert(a); alert(window.a);alert(this.a);

alert(this === window) //true, 全局this就是window

在普通情况，全局函数调用，this指向全局对象window

严格模式，全局函数调用，不会指向window，没有谁来调用，就是默认值，undefined；

### 局部this

谁调用，this就指向谁。

局部是指函数内部，函数调用时，会自动创建this，指向调用者。

### 对象方法this

在对象上，调用方法，this指向对象本身

### 构造函数中的this

指向使用new关键字创建实例，this就指向新实例。

### 修改this（借来的this）

#### 修改this的指向方法1：fn.call方法

语法：fn.call(thisTarget,arg1,arg2...,argN)

user1.phone.call(user2); // call修改this的指向，但始终不改变user2没有call之前的函数的事实。可以减少冗余代码。

#### 修改this的指向方法2：fn.apply方法

语法：fn.apply(thisTarget,[arg1,arg2...,argN])

user1.phone.apply(user2); // apply修改this的指向，但始终不改变user2没有call之前的函数的事实。可以减少冗余代码。

#### 修改this的指向方法3：fn.bind()方法

语法：fn.bind(thisTarget,arg1,arg2...,argN)

user1.phone.apply(user2); // bind修改this的指向，但是没有立即执行，返回一个修改this后的函数对象。bind适用于，指定this的修改，但是“当时”并不会“立即”使用的场景。（相当于复制，以备以后使用：定时器‘回调函数’，事件‘事件响应函数’）

#### 总结：call apply bind

相同之处：

1. 都属于“函数对象”上的方法
2. 第一个参数都是绑定作用域，修改this指向
3. 都可以传递参数

不同之处：

1. apply,call 兼容性更好，适合即刻调用
2. bind较新，低版本浏览器可能有兼容性问题
3. apply传递的参数必须是用数组进行包装

## 面向对象的三大特征

### 封装

简单封装：对程序中，对象相关的属性，方法等定义使用IIFE进行打包

真实场景：在某些场景，需要针对对象的部分特征，进行私有化，特殊化管理

适用场景：

1. 私有化变量，方法等相关的资源（在定义类的构造器中，使用var代替this）
2. 提供特权方法对现实资源进行“特别修饰后的”访问（对资源访问权限进行控制）

在私有资源后，提供一堆get（取值）set（设置）开头的方法

作用：使用特权方法，基于闭包，可以访问私有资源

在特权方法中，根据实际需要，对返回操作或者设置操作进行合理调整

1. 使用闭包IIFE的方式，把一个类定义相关的操作，全部打包起来

类的特征：1）构造函数 2）原型 3）静态方法（Math.random(); $.each()直接在类级别定义）

### 继承

面向对象的继承。让子类拥有父类的所有特征(构造器和原型上所有的特征)。

类与类之间可以通过继承来共享资源

检测类继承的关系 instanceof

(对象伪造)---for in

实现ES5继承的3大金句：

1. 先完成父类构造器的继承
2. 再让子类原型拥有父类的原型
3. 还原子类构造器

实现ES6继承

class 类名{

constructor(){

....

}

}

class 子类 extends 父类 {

constructor(arg1,agr2){

super(arg1); //先父

this.agr2 = agr2; //后子

}

}

小结：ES6继承

1. 在类声明上使用extends来实现继承（原型链）
2. 在构造器中，声明父类构造的调用

super(); -->无条件指向继承的父类

### 多态

JavaScript程序运行中，允许函数的调用，根据参数的个数、类型的不同，体现不同的效果。

举例：

$.html() //取值

$.html(‘<div>haha</div>’) //设值

多态的实践方式2种：

复写：在继承体系中，父类拥有一个方法（比如toString,在Date子类中实现不一样）

重载：方法名相同，但是根据参数的个数或者是类型，体现不同的效果(js中没有重载，但是可以模拟重载----arguments内置对象判断实现。个数length，类型typeof/instanceof)

# 2018.03.12

## 复习

### jQuery环境的搭建

1. 官网下载
2. 本地安装
   * 1. 本地引入
     2. 远程引入
3. 测试引入成功（alert($)）

### jQuery面向对象

原生JS：POP

var h1s = document.getElementsByTagName(‘h1’);

if(h1s.length>0){

var h1 = h1s.item(0);

var content = h1.innerHTML;

alert(content);

}

jQuery: OOP

alert($(‘h2’).html())

### 万能的$

$(参数)

1. 值为函数，页面加载

//jQuery简化形式

$(function(){....})

//jQuery原生

$(document).ready(function(){...})

//JS原生

window.onload

1. 选择器（字符串）

$(‘选择器’).操作

1. dom包装（把DOM对象包装为jQuery对象）---应用场景：事件监听

$(domEle)

1. 创建一个新元素加入页面（符合HTML语法的字符串）

//JS原生

var h2= document.creatElement(‘h2’);

h2.inneHTML= “xxx”;

var body = document.getElementByTagName(‘body’).item(0);

body.append(h2);

//jQuery

$(‘<h1>哎呀</h1>’).appendTo(‘body’);

### jQuery常用方法

#### .attr() 设置或返回被选元素的属性值

JS：

原生属性：

对象.属性名

对象[‘属性名’]----这两种方式都只能操作原生属性

自定义属性：

对象.getAttribute(‘原生属性’);

对象.getAttribute(‘自定义属性’);

对象.setAttribute(‘属性名’, ‘属性值’);

jQuery：

jqObj.attr(‘属性名’)----取值

jqObj.attr(‘属性名’, ’属性值’)---设置

（.attr .html .text .css）

注意: 双向取值设置的方法，对于取值，都是只取第一个匹配的内容，设置是批量操作。

#### .each()

对象方法

$(‘div’).each(function(index,obj){

$(obj).attr(‘xx’);

})

静态方法

$.each(‘div’,callback(index,obj))

$.now();

#### .find() 性能优化，进阶搜索

#### $.trim(str); //去除字符串两侧无效字符

## 事件

事件的4要素：

事件源：标签元素

事件名：需要捕获用户行为

事件响应函数（回调）：当触发了某某行为，打包需要执行的逻辑。

事件对象：封装事件上下文信息

事件监听的语法：

1. 直接在标签上，使用on..属性来定义事件监听

自动封装事件方法，div.onclick=function.onclick(){...},注意this的指向问题。

1. 在HTML DOM 对象上，直接操作onXXX...（只能监听一次）

div.onclick = function(){....} this指向事件源

1. 使用方法完成监听（对同一个事件进行多次监听）

参数：事件名，事件响应函数

p.addEventListener(‘click’,function(){....});

p.removeEventListener(‘click’,fn);

注意：移除时，传入的事件响应函数，必须和监听时指向同一个函数对象才能准确移除

## jQuery事件监听

jqObj.on(事件名，事件响应函数)...(对同一个事件进行多次监听)

jqObj.off((事件名)---->移除所有相关事件

jqObj.off((事件名,事件函数)---》移除指定相关事件

## jQuery事件的命名空间

对全局事件，进行额外名称的定义，可以结合事件名称+命名空间的方式，更好的去区分事件的种类。

jqObj.on(‘事件名称[.命名空间]’,事件响应函数)

jqObj.on(‘click.first’,function(){});

jqObj.off(‘click.second’); //一起移除，可以空格分离，多个事件名；也可以有相同的命名空间，移除命名空间就行。（添加就不能用命名空间）

## 事件委派

在指定父级范围类，监听指定（查询）规则的子元素，如果匹配则触发事件。

原理：基于事件冒泡。在父级捕获触发时，如何获取触发事件的真正‘子元素’目标

event.target （获取 事件源）

delegate(selector,type,fn);

$(‘#box’).delegate(‘div’,’click’,function(){....})

监听语法：

on(事件名称，事件响应函数);

事件委派

父类.on(事件名称，“需要委派监听的子元素”，事件响应函数)

使用场景

在一些会动态更新内容的页面，新内容也需要做监听的情况。

例如：数据列表（银行卡的添加删除）

## 自定义事件

自定义事件规则：

1. 在所有的事件无法凭空创造，所以自定义事件，先要找到一个相近的原生事件。
2. 在原生事件中，做好特定条件的判断。

if(....){

$(this).trigger(自定义事件名)

}

1. 在条件判断中，使用‘jQuery事件触发器’触发‘自定义名称’的事件
2. 通过jQueryOjb.on(自定义事件名，事件响应函数)

## 插件

### 插件的种类：

基于原型扩展的方法（最常用）

静态扩展

选择器

### 插件命名规范：

普通版： jQuery.插件名.js

压缩版：jQuery.插件名.min.js

### 插件冲突：

渡让----jQuery引进后 添加 $.noConflict();

普通渡让：$

深入deep渡让：$ 和jQuery

解决方法

var 新全局JQ名称 = $.noConfict(deep);

闭包解决：(先加上分号，以防冲突，将jQuery(渡让得到的符合的全局值)作为参数传到iife 中，重新换成$)

;(function($){

// 创建插件逻辑

function \_color () {....}

//把定义的结果暴露出去

$.fn.color = \_color;

})(jQuery)

### 插件机制：

主要扩展方式就是

#### 原型扩展；

jQuery.prototype ---》简写：$.fn

#### extend扩展

单个扩展：

批量扩展：

### jQuery 的this

插件方法的this指向jQuery对象

### 返回jQuery的返回值指向对象，可以做链式操作

jQuery的优点：

1. 简单
2. 兼容性（版本越高，对低版本的兼容越差）
3. 链式操作

# 2018.03.14

## 回顾

### jQuery回顾：

1. 环境搭建
2. 下载
3. 引入
4. 测试
5. 万能的$
   1. 函数值（页面加载完毕）
   2. 选择器字符串（搜索页面元素）
   3. DOM对象值（把原生对象转化为jQuery对象）
   4. HTML字符串（创建新元素并包装成jQuery对象）
6. 常用方法

### 事件回顾

1. 四要素：

事件源

事件名称

事件响应函数

事件对象

1. 事件监听、解绑

绑定： jQuery.on(type,function(){})

解绑： jQuery.off(type)

原生JavaScript

标签

dom.on... ---->解绑 = null

.addEvenListener(type,function(){})---->removeEventListener

1. 命名空间

事件名称.命名空间

1. 事件委派

事件监听：页面加载是同步加载（已经存在）。某些场景需要用户的添加的操作，这些也需要监听。通过事件的冒泡机制，通过事件

delegate

### 插件

插件命名：

jQuery.插件名.js jQuery.插件名.min.js

使用插件:

引入插件库

插件的命名空间

打包插件相关的所有资源，通过参数传递方式调用

# 2018.3.15

## day1总体回顾：

变量，函数，面向对象，构造函数/类，原型

### 变量：

站在内存角度，把变量分为2类：

基本类型：Number String Boolean null undefined

存在栈内存中（first in last out）

引用类型：Object Array Function

存在于堆内存中（无序）

深拷贝 浅拷贝

浅拷贝：面向对象属性拷贝，如果属性值是引用类型，只拷贝引用地址（拷贝一级）

var a = {

“name”: “zhangsan”;

“dept”: {

“name”:”sales”,

“id”:15

}

}

for..in 循环

深拷贝：面向对象属性拷贝，如果属性值为引用类型，会进入该属性值对应的对象，然后再进行属性拷贝（拷贝多级，下穿到最底层基本类型）

$.extend

### 函数

内置对象arguments

1获取实参内容和长度（伪数组：可以循环，但是没有继承数组）

1. 获取函数对象本身 arguments.callee(递归)

一切皆对象

属性：

name：函数名

length：形参长度

方法：

call

apply

bind

### 面向对象

pop: 面向过程：关注细节 progress-oriented-programming

oop：面向对象：关心结果 object-oriented-programming

类：对具有公共特征的事物进行抽象，面向特征（属性）和行为（方法）

（对象）实例：类的具体实现

### 构造函数/类：

构造函数：用来创建类的一种模板。

function Fn(arg,....){

this.xx = arg1;

}

创建实例

var p = new Fn(para1...)

new做了什么

1. 创建了一个类的新实例
2. 让构造函数内部this指向新实例
3. 获取传入的参数，执行构造函数代码（添加属性）
4. 返回新创建的实例

### 原型

原型：是函数对象上自动创建的一个属性

原型是一个对象

原型是用来模拟类

原型中自带有constructor的方法指向类的构造函数

原型的作用：解决方法过载

扩展类型（原生js，jQuery）---插件

用作继承

类原型和对象原型的体现

对象.\_\_proto\_\_ === 类.prototype （true--》引用关系）

## day2 总体回顾

JSON 编译原理 作用域 闭包

### JSON

JSON：一种轻量级的数据交换格式，是一种天然的js对象

JSON的定义

{

“name”: “str”,

“age”:15,

“isMarried”:false,

“dept”:{

“deptNo”: “001”,

“name”: “sales”

}

}

JSON转换

JSON--》string

JSON.stringify(json);

string-->JSON

JSON.parse(jsonStr);

### 编译原理

变量提升

函数提升

提升方式：

函数提升高于变量提升

场景：

变量声明

函数声明

变量赋值

预编译：（语法检测，提升）

var fn；

function fn(){}

解释执行

fn = 100;

### 作用域

在程序执行过程中，会创建一个 独立 的上下文运行空间，可以对变量，函数等资源进行存储。

作用域种类：

1.全局

在函数外部定义的所有资源，会挂在全局window对象上，可以被程序的任意位置所访问。

2.局部

在函数内部定义的所有资源，会挂到独立的内部上下文空间，只能存在内部共享。

3.块级

ES5没有块级作用域，使用IIFE实现

ES6使用let关键字可以声明一个块级（特殊的局部）作用域

作用域：

为了保证在内部程序可以访问外部作用域的资源，逐级向上建立的一种关系网，就是作用域链的体现。

闭包：

闭包是一种 作用域的体现，体现内部程序可以访问外部程序的资源，外部不能访问内部资源。

闭包可以使用一种特殊方式，把内部函数的资源暴露在全局window上，可以上外部访问内部资源，常见体现方式：函数内部嵌套函数.

作用：插件，模块化

写法:

;

function(){

function \_fn(){

...

}

window.fn = \_fn;

}

## day3 回顾

this 封装 继承

### this

全局this

window

局部this

谁调用，指向谁

对象方法this

对象本身

构造函数this

新创建的实例

事件响应函数this

事件源

修改this指向【掌握】

.....立即执行

call(thisObj, arg1,agr2....)

apply(thisObj,[arg1,arg2...])

......延迟执行（回调函数，事件）

bind(thisObj,agr1,agr2.....)

### 封装

含义：把一个类定义相关的所有操作（构造定义，原型定义，静态定义）都打包到一个地方

封装写法

;

(function(){

function \_Stundent(){

}

\_Stundent.prototype.run = function(){

...

}

\_Stundent.Study = function(){...}

window.Student = \_Stundent;

})();

继承

概念：子类拥有父类的所有特征（属性和方法）

### 步骤：

function Parent(){}

Parent.prototype.fn = function(){};

function Child(){

//让子类继承父类构造器

Parent.apply(this.arguments);

}

//让子类原型拥有父类原型的所有特征（1. for in(对象伪造，不全等) 2.new父类(构造器中包含子类不需要的垃圾属性)，3.直接==Parent.prototype引用父，但是子会修改父）

最优解：Child.prototype = Object.create(Parent.prototype);

//还原子类的构造器

Child.prototype.constructor = Child;

## day4 回顾

事件，插件编写

### 事件

事件4要素

事件源：标签

事件名称：行为（点击，鼠标）

事件响应函数：触发行为需要执行的代码

事件对象：封装事件的上下文信息

事件命名空间：

事件源.on(事件名称.命名空间, 事件响应函数)

事件委派：

把原本要监听在子元素上的事件，绑定到父类，通过冒泡机制，和事件对象反查事件源（子元素）

$(‘父’).on(‘事件名’,’子’,事件函数);

父.addEvenListener(‘’,function(e){

e.target获取事件源

e.target.tagName == “tr”

})

jQuery插件编写方式

命名：jquery.name.js jquery.name.min.js

定义：

;

(function($){

var \_target;

//默认参数

var \_defaults ={

参数1 : 默认值1

}

//提供的所有方法

var method = {

方法1：function(){...},

方法2：function(){...},

方法3：function(){...},

方法4：function(){...}

}

//

$.fn.插件名 = function(){

\_target = this;

method[方法名];

return \_target;

}

})(jQuery)

使用：$(选择器).插件名(参数对象);

## 模块化

### 多人写作问题：

全局变量冲突，命名空间过于复杂

引入资源过多

解决：模块化

### 模块化的演变

1. 孵化期 IIFE
2. 过渡期
   1. 解决命名空间问题
      1. YUI().use(‘依赖资源’,function(Y){})
      2. YUI().use(‘jquery.js’,function(Y){(‘#id’)})
      3. YUI().use(‘‘jquery.js’,‘my.js’,function(Y){})----->如果两个依赖资源有重名的方法，玩完

优点：解决依赖整合问题，开辟一块独立的运行空间

缺点：多个库，同名则挂掉

b) 引入资源过多

$LAB.script(‘script1.js’).wait()

.script(‘script2.js’)

.script(‘script3.js’).wait()

.script(‘script4.js’)

优点：简化了script标签的编写

缺点：请求过多，没有解决

当下：

服务器端规范：

ServerJS

CommonJS:国外提供服务器JS实现方式

NodeJS

浏览器端规范：

AMD：国外提供的异步的模块定义方案

RequireJS

CMD：国内支付宝 玉伯提供浏览器模块化方案（CommonJS + AMD整合）

SeaJS

### 模块化规范：

CommonJS定义服务器端JS的实现

AMD定义了浏览器异步模块化方案

实践：

CommonJS定义服务器端JS的实现

NodeJS

AMD定义了浏览器端异步模块化方案

RequireJS

RequireJS

是一个非常小巧的文件和模块化加载框架。

可以让同步加载JS代码，通过模块化打包，不阻塞主进程的情况下，异步执行。asynchronous