

instanceof方法是判断String对象的prototype属性值是否在str的原型链上。

判断str的数据类型时，String.prototype改变的话，不能准确判断出来。

第十四章 标准内置对象3(构造器RegExp)

1.正则表达式：可以匹配我们的目标

正则表达式（Regular Expression）：JavaScript 中的 RegExp 对象（正则对象，可以通过正则对象来创造正则表达式的实例）

正则：规律、规则

表达式：由一些普通字符和特殊字符组成

正则表达式是强大的字符串匹配工具

2. 正则表达式

对字符串操作的一种逻辑公式（对字符串进行匹配的）

用事先定义好的一些特定字符、及这些特定字符的组合，组成一个规则字符串，这个规则字符串用来表达对字符串的一种过滤逻辑

正则表达式是匹配模式，要么匹配字符，要么匹配位置

3. 正则表达式用途：

批量提取/替换有规律的字符串

验证客户端的输入数据

各类办公软件中使用

各种开发语言中使用（C# / Java /JS / Perl / PHP / Python等，不同语言中或许有一些不相同的地方，但是思想上完全一样）

网络爬虫（抓取机器人）的开发

4. 正则表达式测试工具（熟练掌握用法）

https://regex101.com/

<https://jex.im/regulex/>

5. 创建正则表达式：2种形式（这两种形式是等价的，但是我们一般通过字面量来创建）

（1）new 运算符 new RegExp(pattern, flags) 注意pattern中的斜杠

var box = new RegExp('box'); //第一个参数字符串（要匹配的字符）

var box = new RegExp('box','ig'); //第二个参数可选模式修饰符（模式修饰符：规定匹配的时候忽不忽略大小写、匹配方式等）

（2）字面量方式 /pattern/ flags

var box = /box/; //直接用两个反斜杠

var box = /box/ig; //在第二个斜杠后面加上模式修饰符

6.修饰符：写的顺序没有要求。

每个正则表达式都可带一个或多个标志（flags）,标明正则表达式的行为。

i代表忽略大小写，不写的话不忽略大小写，只匹配完全相同的。

g是global的意思，不写的话匹配到就结束了，写的话无论匹配到还是没有匹配到，都会从头扫到最后，把所有的都匹配出来。

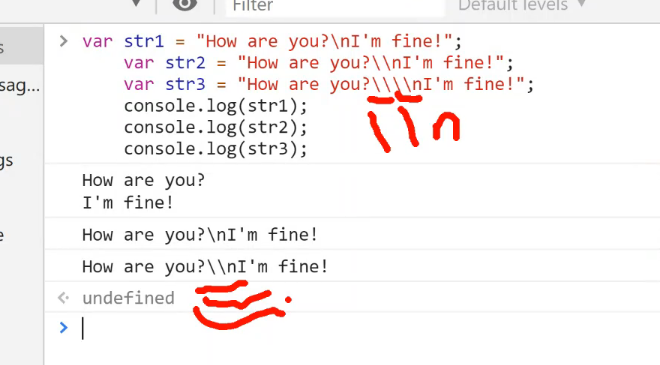
m是代表匹配多行，不写的话默认匹配一行。



7.转义字符：

* 元字符是具有特殊含义的字符，不能直接匹配使用的，必须使用反斜线（**\**） 作为前缀进行转义后才能使用。
* 字符串、正则表达式字面量、正则表达式构造函数都使用反斜线（**\**） 作为转义字符的前缀。

字符串的转义字符：每两个\\为一组，第一个为转义字符，第二个作为字符留下。



正则表达式的转义字符：一般实例化创建的时候会比通过字面量创建正则表达式多写一倍的反斜杠。



8. 一般不推荐使用构造函数生成正则，而应该优先使用字面量。

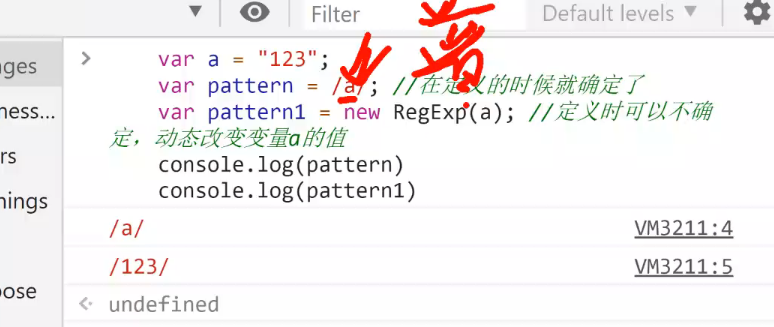
var pattern = new RegExp("^\\d+","g"); //注意，反斜杠需要转义

var pattern =/^\d+/g

构造函数会需要多写一个 \ 。

构造函数的匹配模式可以使用变量，但是字面量不能够使用变量代替。

一般动态获取值的情况下或者传入变量的情况下，会使用构造函数生成正则。



9.正则方法：

RegExp 原型方法（正则表达式的方法，注意使用时的写法）

test() 定义在RegExp.prototype上

exec() 定义在RegExp.prototype上

String 原型方法（字符串的方法，注意使用时的写法）

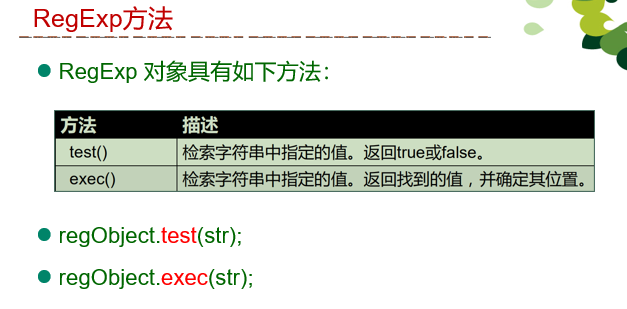
replace()

match()

search()

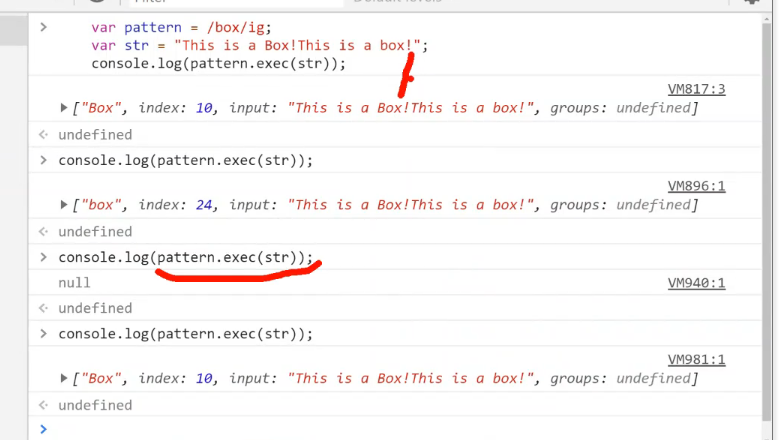
split()

10. RegExp方法：正则表达式的方法

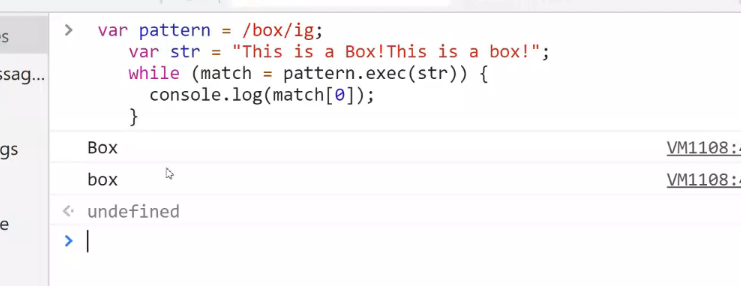


exec()方法不仅能检索字符串中指定的值，还能返回找到的位置（全局匹配模式下，从左向右匹配，每次只匹配一个，整个字符串匹配完成后，再从头开始匹配）。

如下例：



exec方法经常和while循环一起使用。（第三次调用时while里的条件为假，就不在继续执行了）



test方法：注意pattern和str的位置，以下例子返回true。



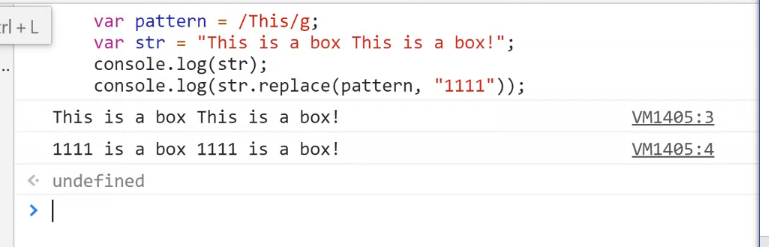
11. 模式匹配的String方法：字符串的方法

（1）replace()方法

strObject.replace(regexp,'replaceString');

对字符串中特定格式的子串进行替换，返回替换后的结果

第一个参数既可以是一个固定的子串，也可以是一个正则表达式对象

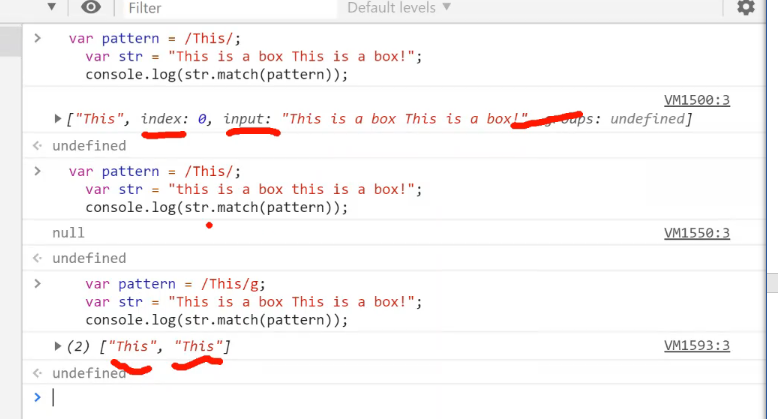


（2）match()方法

strObject.match(regexp);

一个或多个子串、正则表达式的匹配

返回一个数组

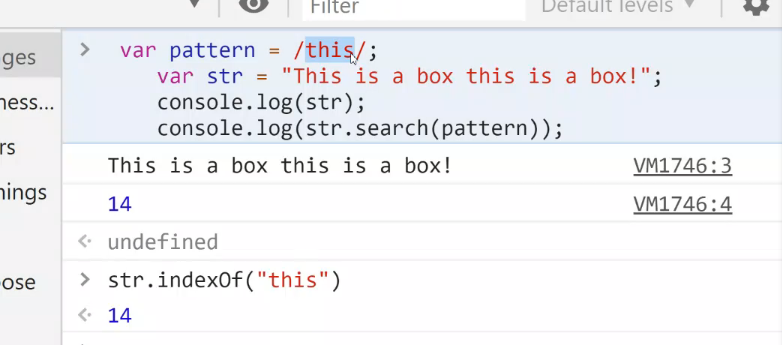


（3）search()方法

strObject.search(regexp);

返回第一次出现匹配指定正则表达式子串的下标，若没有匹配则返回 -1

indexOf() 作用类似，但 indexOf() 不支持正则表达式



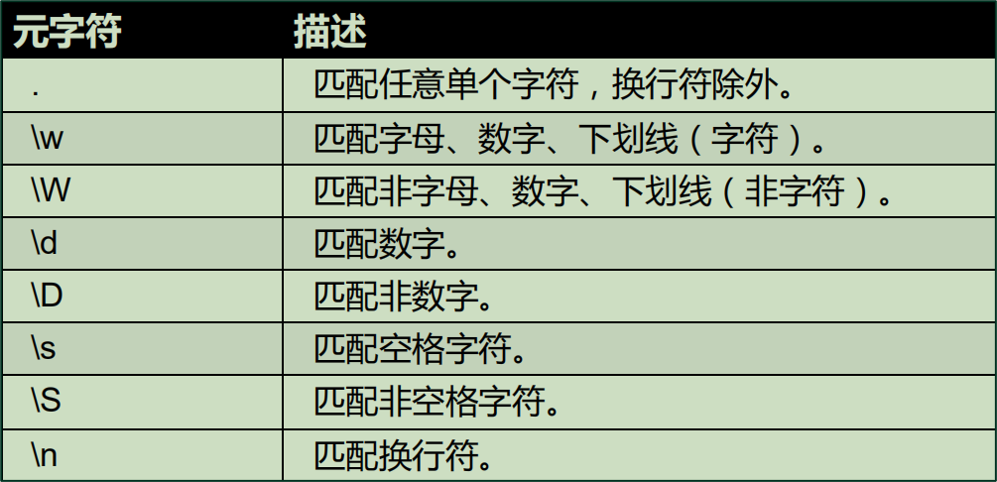
（4）split()方法：返回的是数组类型。

strObject.split(regexp,[howmany]);

用一个指定的字符串或正则表达式，对原字符串进行拆分，返回拆得的子串数组

若指定了 howmany 属性，则只返回拆得的前 howmany 个子串

12.元字符：元字符是拥有特殊含义的字符

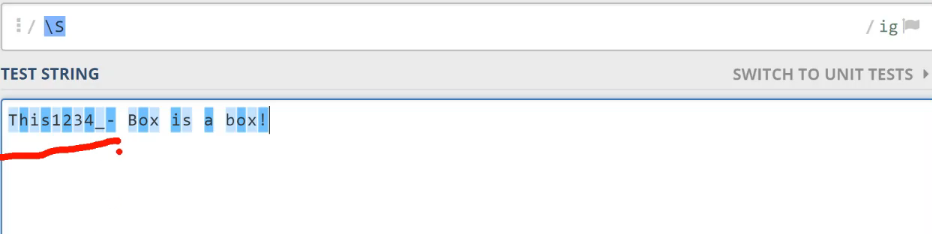


一个点（.）可以匹配任意单个字符，但是加上模式修饰符g之后，可以匹配所有字符（换行符除外）。

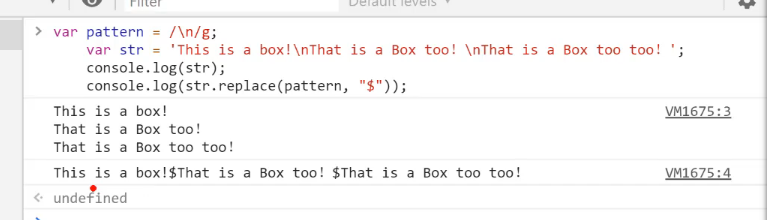
\w其实就是JS中的一个标识符（由数字、字母、下划线组成）。

\w和\W、\d和\D、\s和\S都是相反的。

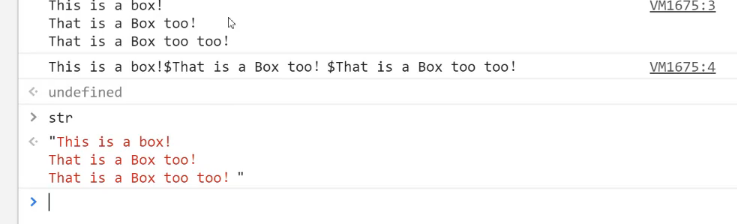
以下例子是\S匹配非空格字符并且加上模式修饰符g的情况。



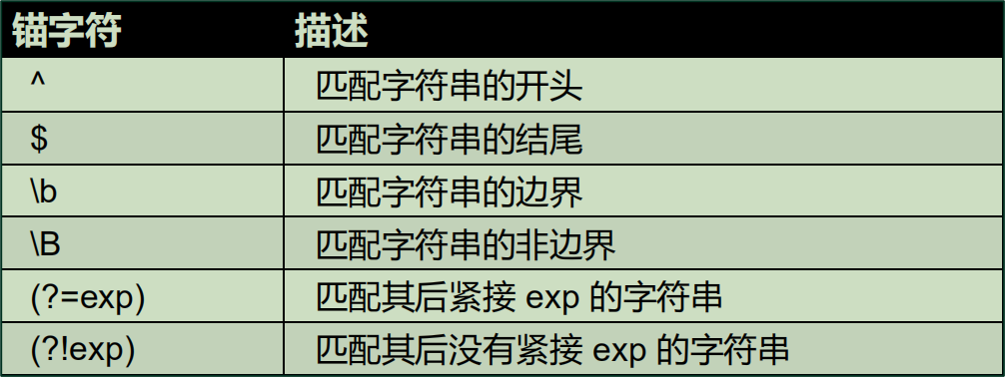
在我们用的工具里面是没法测试换行符\n的，可以去代码里查看。



通过replace方法对\n进行替换，并不是对原有的字符串进行改变，而是对于副本进行了改变。

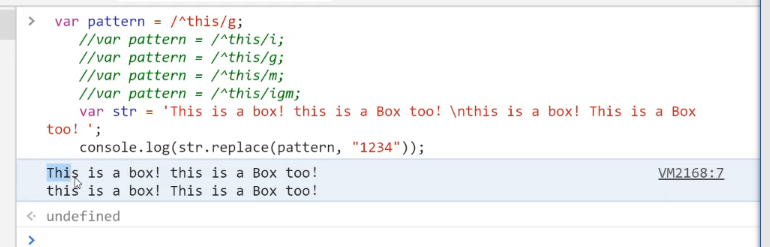
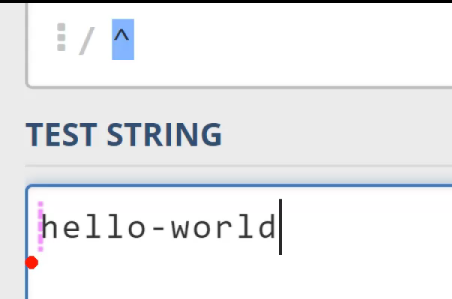


13. 锚字符：锚字符用于查找某个位置。

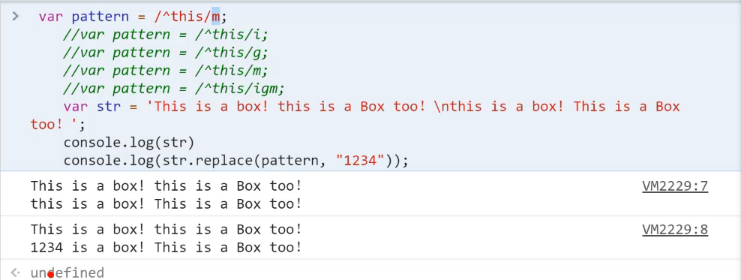


exp 在这里代表任意表达式。

（1）^字符：匹配字符串的开头



全局匹配只会匹配开头的部分，发现没有匹配成功，所以通过replace方法不会进行替换。



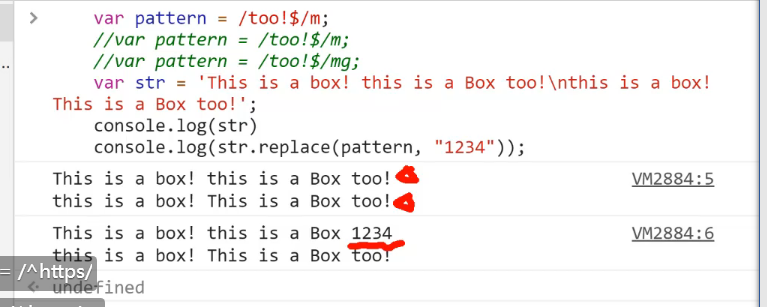
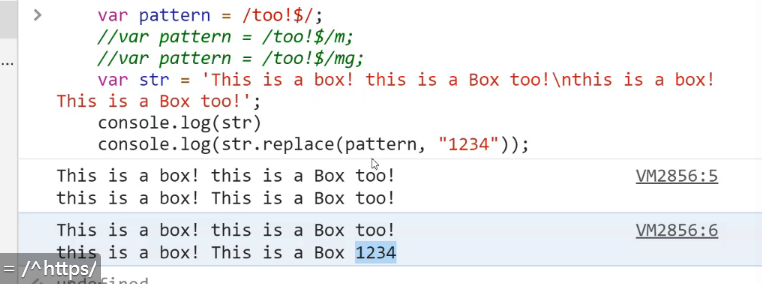
当我们匹配多行字符的时候，每一行的开头都会匹配。发现第二行开头匹配成功了，所以第二行开头的部分被替换了。

》锚字符用于查找某个位置

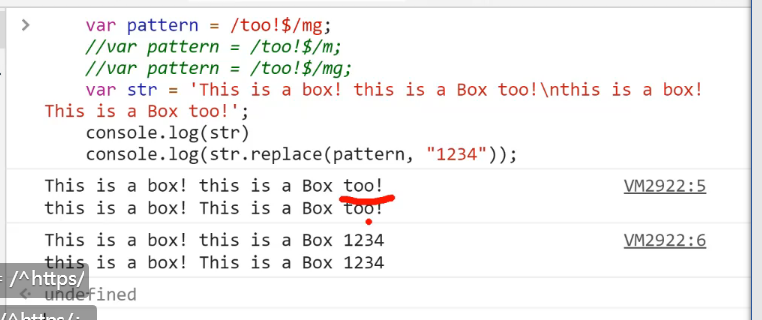
——位置是指字符串中每行第一个字符的左边、最后一个字符的右边以及相邻字符的中间。比如，下图中箭头所指的地方。

——对于位置的理解，我们也可以理解成空字符""

（2）$字符：匹配字符串的结尾。



对于多行匹配时，如果能匹配成功，只会匹配到一个就结束了，所以只替换了第一行的结尾部分。

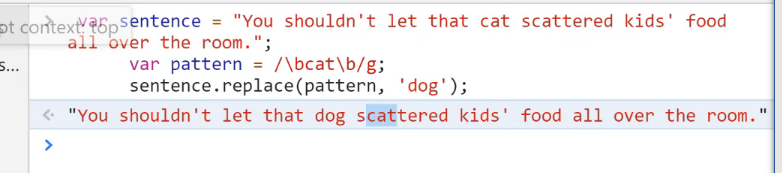


而全局匹配就会去匹配所有行的结尾部分，通过replace方法匹配，匹配成功后，会进行替换。

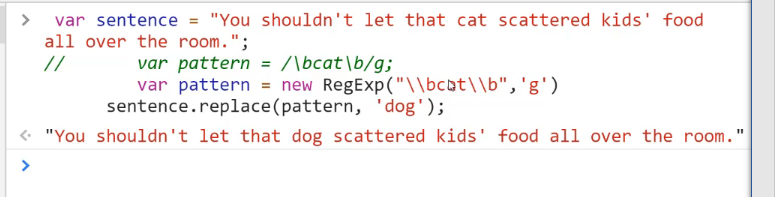
（3）\b字符：匹配字符串的边界



如下例：



使用构造实例化的形式：



（4）零宽断言：

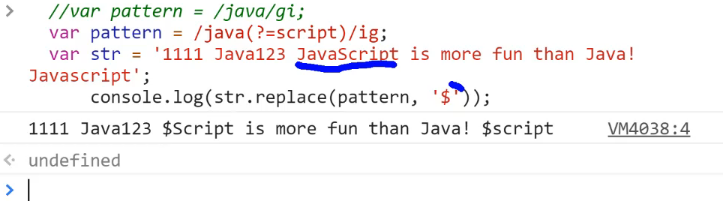
(?=exp) —— 零宽度正预测先行断言

断言此位置的后面能匹配表达式 exp

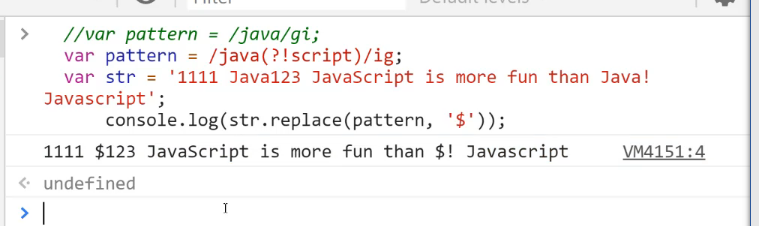
(?!exp) —— 零宽度负预测先行断言

断言此位置的后面不能匹配表达式 exp

如下例：匹配到字符java并且后面紧跟着script的时候进行替换（不区分大小写，全局匹配），发现并不会替换掉后面紧跟着的部分，匹配的是零宽度，是位置。



零宽度负预测先行断言：如下例：



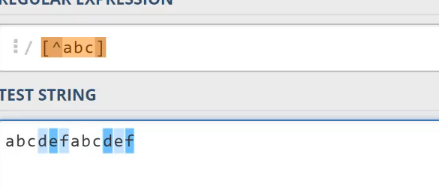
14.方括号：方括号用于查找某个范围内的字符

* [A-z0-9\_] 匹配英文字母、数字、下划线
* [A-z0-9\_] \w
* [^A-z0-9\_] \W
* [0-9] \d
* [^0-9] \D

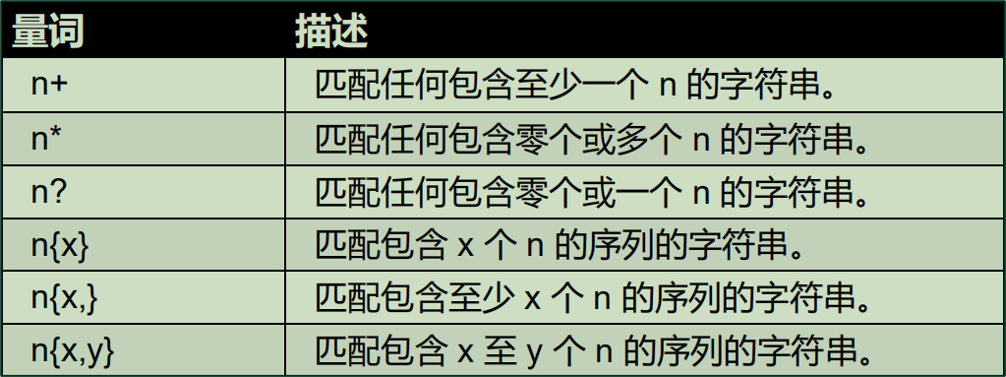


如：

注意：[A-f]表示的就是：A-F和a-f这些字符。

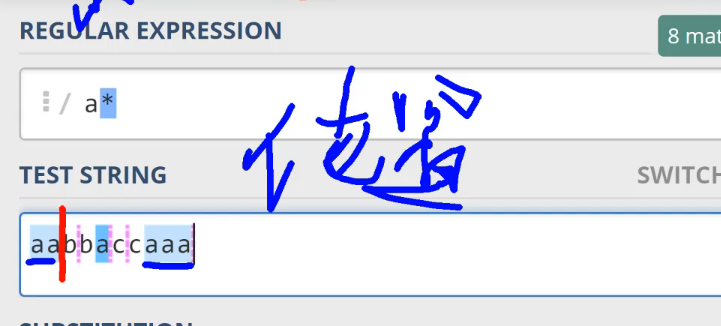


15. 量词：量词是定义字符出现频次的字符



+号匹配至少一次。

注意n\*：0个也能匹配上。尽量少用。多个字符是作为整体来匹配。

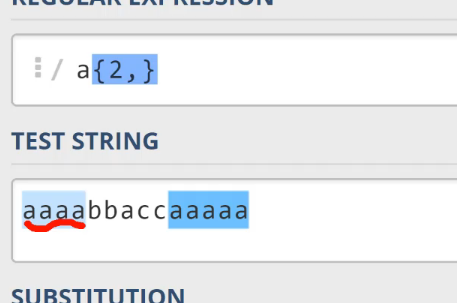
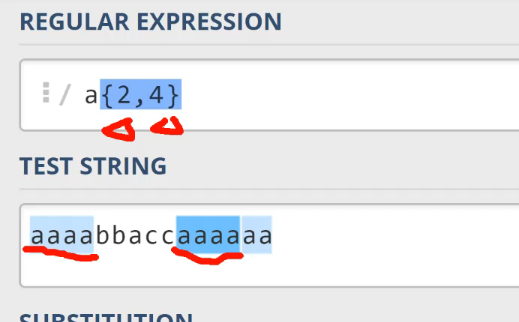


而n?：能匹配0个，且最多只能够匹配一个字符，所以出现多个能匹配到的字符是一个一个匹配的，而不是作为整体来匹配。

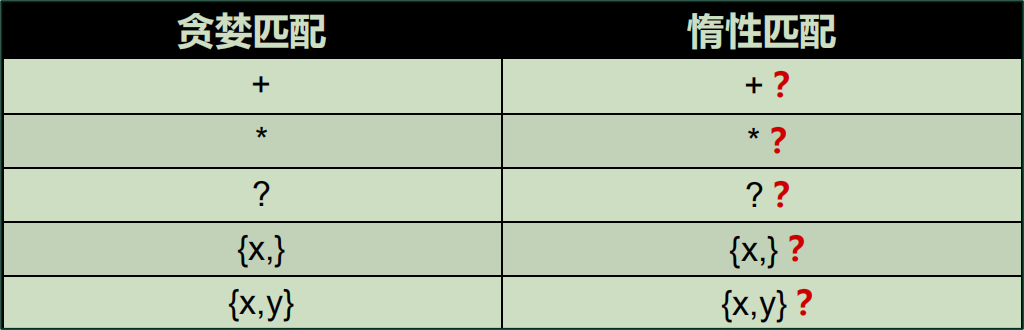


对于n{x，y}：贪婪模式，只要能匹配上，就更多的去匹配。指能匹配y个就匹配y个。

如下例：能匹配到4个就绝对不匹配2个。



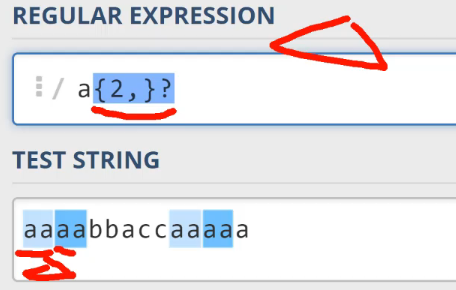
16.贪婪和惰性：默认是贪婪模式。



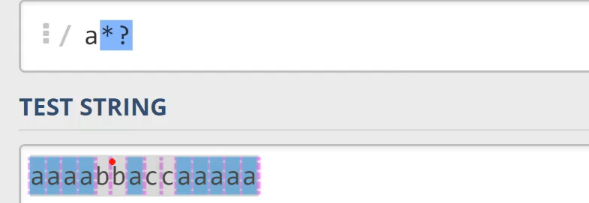
变成惰性匹配：在量词后面加一个?号，？的含义就是能匹配1个或者0个。

惰性匹配：匹配尽可能少的。

a{x,}? 在惰性匹配模式下，匹配最少的是出现x次，基本上等同于a{x}。



如果对于有字符的地方，匹配1个字符是最少的情况；对于没有字符的位置，匹配0个是最少的。



17.选择和分组：有选择的时候，是从左到右开始匹配，如果没有匹配成功的话，再往下走。

与方括号不同的是选择这里可以对一个子串进行选择，例如red| green|blue,如果匹配上red，就不会再向右匹配了，是匹配到就结束了。如果左边匹配上了，就不会向右走了。注：在匹配十六进制颜色时，一般把完整更多次的匹配放到前面。

使用（）小括号括起来的结果可以用正则静态属性RegExp.$1-$9来获取，有几个小括号就是$几。



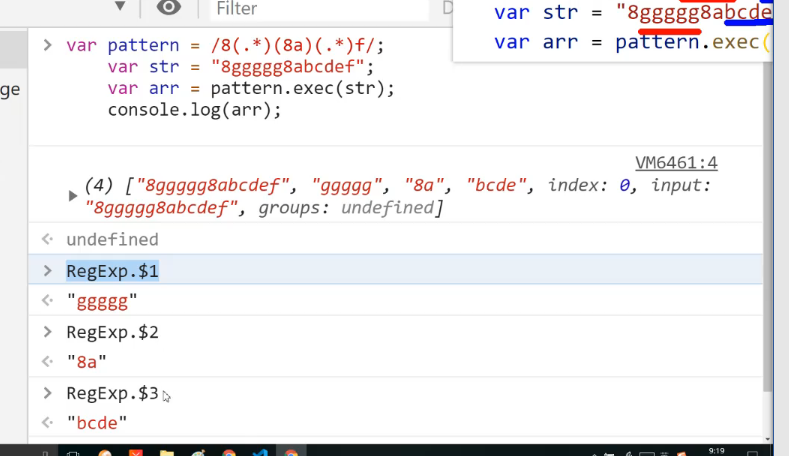
18.正则静态属性：定义在RegExp对象上。

* RegExp.$1-$9 掌握这个
* RegExp.input ($\_)
* RegExp.lastMatch ($&)
* RegExp.lastParen ($+)
* RegExp.leftContext ($`)

RegExp.$1-$9：匹配的是小括号里面的值。

如下例：使用exec方法进行匹配。

打印出来的结果，第一个值是我们匹配的字符串整体。



19. 正则实例属性：

* patten.flags
* patten.global
* patten.ignoreCase
* patten.multiline
* patten.lastIndex
* patten.source

20.正则表达式练习：看课件。

第十五章 标准内置对象4（非构造器）

1. Math：不是构造器，不能用new关键字来创建。

Math 对象包含了一系列的数学运算的功能。

Math 对象不需要创建，直接使用。

常用操作：

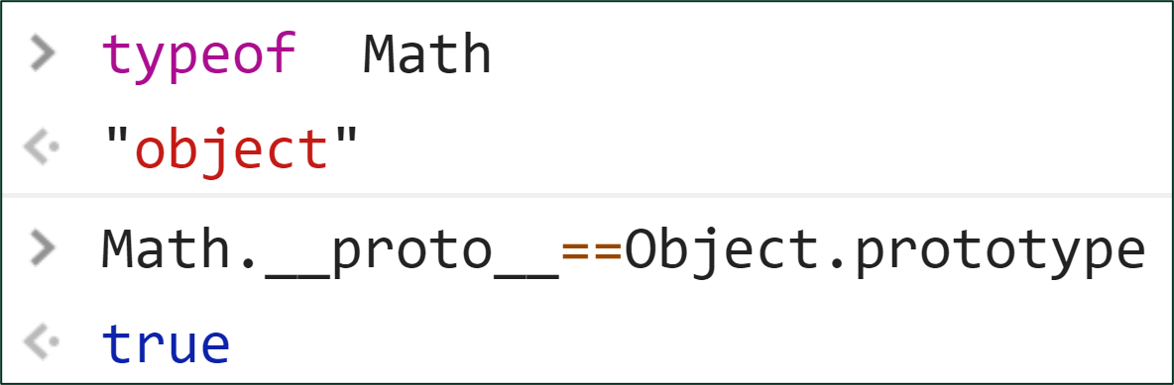
四舍五入: Math. round( )

向下取整: Math.floor( ) 向上取整: Math.ceil( )

生成 [0~1) 随机数: Math.random( ) 包括0，不包括1，等概率的生成随机数。

取最大值: Math.max( ) 取最小值: Math.min( )

1. Math 原型链



1. 数据交换格式

在日常生活中，人与人之间的交流需要彼此都听得懂得语言。同理，在计算机的不同程序之间，或者不同的编程语言之间进行交换数据，也需要一种大家都能听得懂得‘语言’，这就是数据交换格式，它通过文本以特定的形式来进行描述数据。

JSON就是一种数据交换格式。前端和后端经常打交道的数据交换格式句式JSON格式。

XML和JSON是常见的两种数据交换格式。

1. XML ：非常类似于HTML，但比HTML更严格。

XML 指可扩展标记语言（EXtensible Markup Language）

XML 是一种标记语言，很类似 HTML

XML 的设计宗旨是传输数据，而非显示数据

XML 标签没有被预定义。您需要自行定义标签。自己定义标签。

XML 被设计为具有自我描述性。

XML 被设计用来传输和存储数据。

HTML 被设计用来显示数据。

XML比HTML更严格，比如在HTML中少写一个标签的闭合标签，不会报错，但是XML就会报错。

1. XML书写准则：

（1）xml文件的第一行必须是声明该文件是xml文件以及它所使用的xml规范版本。在文件的前面不能够有其它元素或者注释。

（2）在xml文件中有且只能够有一个根元素。

（3）xml文件中，用的大多都是自定义的标记。

（4）在xml文件中的标记必须正确地关闭

（5）标记之间不得交叉。

（6）属性值必须要用“ ”号括起来。

（7）控制标记、指令和属性名称等英文要区分大小写。

1. JSON：

21世纪初，Douglas Crockford 寻找一种简便的数据交换格式，能够在服务器之间交换数据。当时通用的数据交换语言是 XML，但是 Douglas Crockford 觉得 XML 的生成和解析都太麻烦，所以他提出了一种简化格式，也就是JSON。

JSON 的规格非常简单，只用一个页面几百个字就能说清楚，而且 Douglas Crockford 声称这个规格永远不必升级，因为该规定的都规定了。

1. JSON（JavaScript Object Notation）

JSON 指的是 JavaScript 对象表示法

JSON 是轻量级的文本数据交换格式

格式由道格拉斯（ Douglas Crockford） 提出

从 JavaScript 脚本语言中演变而来

JSON 独立于语言和平台（JSON是独立于语言和平台的数据交换格式）

文件名扩展是 .json

在JSON文件中不允许有注释，因为JSON是轻量级的文本数据交换格式。

1. JSON（JavaScript Object Notation）

更小、更快，更易解析

是一种数据格式，不是编程语言

虽然具有相同的语法形式，但 JSON 并不从属于 JavaScript

并不是只有 JavaScript 才使用 JSON，JSON 只是一种数据格式

很多编程语言都有针对 JSON 的解析器和序列化器。

比如说java语言，也可以使用JSON。

1. JSON语法：

并列的数据之间用逗号（", "）分隔

并列数据的集合（数组）用方括号("[]")表示

映射用冒号（": "）表示

映射的集合（对象）用大括号（"{}"）表示

1. JSON和对象的写法类似，但JSON不是对象，也有和对象不同的地方：如以下11、12、13。
2. JSON里面的字符串必须加双引号，单引号不可以，会报错。
3. JSON 语法可以表示以下三种类型的值：和JS中的基本相同，唯一不同的是JSON不支持undefined。

（1）简单值：

与 JavaScript 语法相同，可以在 JSON 中表示字符串、数值、布尔值和 null。

JSON 不支持 JavaScript 中的特殊值 undefined（undefined是JS独有的，其他语言没有）。

（2）对象：

一种复杂数据类型，表示一组无序的键值对。

键值对中的值可以是任意类型——简单值、对象或数组。

（3）数组：

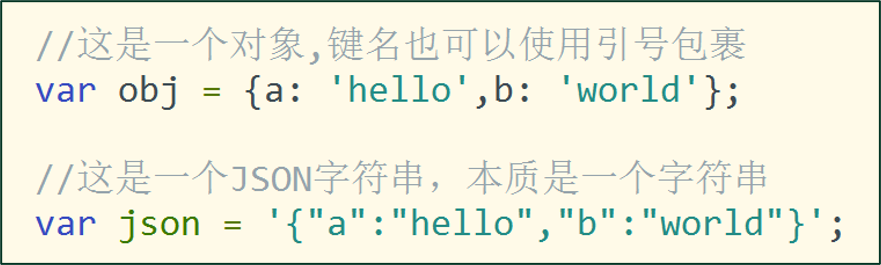
一种复杂数据类型，表示一组有序的值的列表。

数组的值可以是任意类型——简单值、对象或数组。

1. JSON与JS对象：

JSON 是 JavaScript 对象的字符串形式的表示法，使用文本表示 JavaScript 对象的信息，本质是一个字符串，用typeof检测的话是string类型。

如果里面字符串用的是双引号，外面可用单引号。如下：



1. 对象的序列化：用stringify（）方法

JSON.stringify(object [,replacer [,space] ] )方法

* 1. 把 JavaScript 对象格式转化为 JSON 字符串
  2. undefined 值会被忽略
  3. object：指定转换对象

JSON.stringify(object [,replacer [,space] ] )方法

replacer：参数过滤器为数组时，结果只包含数组中列出的属性

JSON.stringify(object [,replacer [,space] ] )方法

replacer：参数过滤器为函数时，需接受两个参数，属性键名和属性值

JSON.stringify(object [,replacer [,space] ] )方法

space：参数为数值时，表示每个级别缩进的空格数

JSON.stringify(object [,replacer [,space] ] )方法

space：参数为字符串时，使用字符串作为缩进字符

1. 反序列化：用parse（）方法

JSON.parse(json [,reviver])方法

* 1. 把JSON字符串解析为原生JavaScript对象
  2. json：参数必须是有效的 JSON，否则会报错

JSON.parse(json [,reviver])方法

reviver：参数过滤器为函数时，需接受两个参数，属性键名和属性值

16.

对象序列化（serialization）

将对象的状态转换为字符串

JSON.stringify( )

对象反序列化（deseriallization ）

将字符串还原为对象的状态

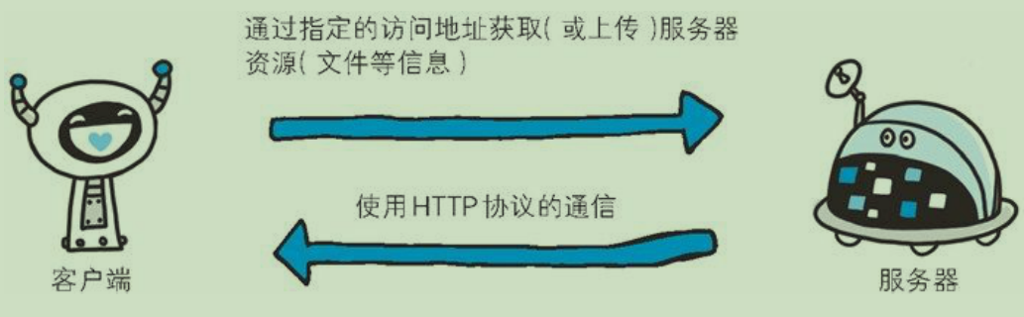
JSON.parse( )

第十六章

1. HTTP协议：默认端口号80，可以省略。

Web 使用一种名为 HTTP（HyperText Transfer Protocol，超文本传输协议）的协议作为规范，完成从客户端到服务器端等一系列运作流程。而协议是指规则的约定。可以说，Web 是建立在 HTTP 协议上通信的。

HTTP 是一种不保存状态，即无状态（stateless）协议。HTTP 协议自身不对请求和响应之间的通信状态进行保存。也就是说在 HTTP 这个级别，协议对于发送过的请求或响应都不做持久化处理。（让服务器能够识别你是谁，需要用到其它技术，如cookie session）。



1. 请求响应：

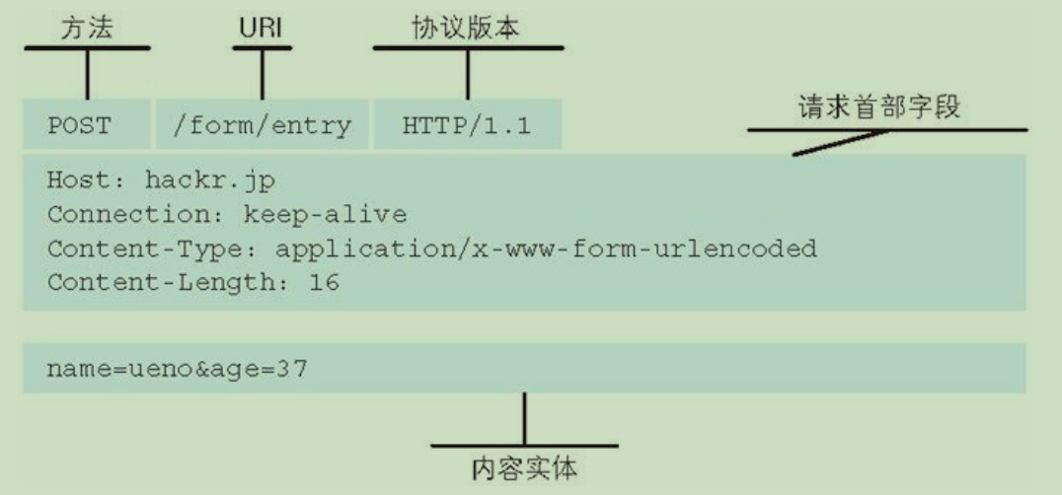
HTTP 协议规定，请求从客户端发出，最后服务器端响应该请求并返回。（HTTP协议没有办法让服务端主动去请求）

1. 请求报文：

请求报文是由请求方法、请求 URI、协议版本、可选的请求首部字段和内容实体构成的。

每个URI都对应服务器的某些资源，每一个资源都要有一个对应的URI。

Host：域名。Connection：长连接。Content-type：指定数据的格式或类型，content-length：内容实体的长度。



1. 响应报文：

响应报文基本上由协议版本、状态码（表示请求成功或失败的数字代码）、用以解释状态码的原因短语、可选的响应首部字段以及实体主体构成。



1. URI（Uniform Resource Identifier）统一资源标识符

URI不是任何的字符都可以应用的。发给服务器的时候不可以出现中文字符，在地址栏

中显示中文，是因为URL进行了转义。

1. 全局方法：

encodeURI —— 把字符串作为 URI 进行编码（把中文转换成）

decodeURI —— 可对 encodeURI() 函数编码过的 URI 进行解码

1. 方法（Method）—— 告知服务器意图的 HTTP 方法

GET ：获取资源

POST：传输实体主体

PUT：传输文件

DELETE：删除文件

HEAD：获得报文首部

OPTIONS：询问支持的方法

指定请求的资源按期望产生某种行为，使用方法下达命令

1. 状态码：

HTTP 状态码负责表示客户端 HTTP 请求的返回结果、标记服务器端的处理是否正常、通知出现的错误等工作。

1. 表单：

HTML <form> 元素表示了文档中的一个区域，此区域包含有交互控制元件，用来向 Web 服务器提交信息。

action —— 一个处理此表单信息的程序所在的 URL

method —— 浏览器使用这种 HTTP 方式来提交表单

enctype —— 将表单的内容提交给服务器的 MIME 类型

1. enctype：

当 method 属性值为 get时, enctype 的取值为:（get请求提交的数据会显示在地址栏上。）

application/x-www-form-urlencoded: 未指定属性时的默认值

当 method 属性值为 post 时, enctype 可能的取值有:

application/x-www-form-urlencoded: 未指定属性时的默认值

multipart/form-data: 用于一个 type 属性设置为 "file" 的 <input> 元素（提交的数据有文件或者图片时用这个）

text/plain (HTML5)

1. 当 method 属性值为 get时, 提交的数据会显示在地址栏上。query部分就是地址栏中？后面#前面的部分。想要拿到query中自己写的内容，可以截取字符串，解码，会使用split方法。

第十七章

1. 事件绑定：

事件绑定分为两种：一种是传统事件绑定(内联模型，脚本模型)，一种是现代事件绑定(DOM2级模型)

》》内联模型

<button onclick="tell();">弹出提示框</button>

违反了“内容与行为相分离的原则”，尽量少用

》》脚本模型

document.getElementById('btn').onclick = function(){ }

实现了“内容与行为相分离”但元素只能绑定一个监听函数（当绑定了多个监听函数时，后面的会覆盖前面的）

2. DOM2 模型

实现了“内容与行为相分离”

元素可以绑定多个监听函数

每一个元素都可以去调用addEventListener方法。执行的顺序就是我们定义的监听函数的顺序。

element.addEventListener(type， listener[， useCapture])

type —— 表示监听事件类型的字符串，不需要加“on”前缀（例如使用的时候写click、blur）

listener —— 指定事件触发时执行的函数

useCapture —— 布尔值，指定事件是否在捕获或冒泡阶段触发（后面讲什么是捕获或者冒泡阶段）

3. EventTarget 对象：

EventTarget 是一个由可以接收事件的对象实现的接口，并且可以为它们创建侦听器。

Element，document 和 window 是最常见的事件目标，但是其他对象也可以是事件目标，比如 XMLHttpRequest。

许多事件目标（包括元素，文档和 window）还支持通过 on... 属性设置事件处理程序。

4.不是所有的事件侦听器都能被删除，只能移除具名函数的侦听器。



addEventListener()

注册特定事件类型的事件处理程序

removeEventListener(type，listener)

删除事件侦听器

具名函数可删除，匿名函数不可删除

dispatchEvent()

向一个指定的事件目标派发一个事件

通常与自定义事件结合使用，使用Event构造函数去自定义事件

可以用dispatchEvent()去进行一个事件的触发



5. 事件三要素说明

事件是被动触发的，并不是我们能主动“加”上去的。（事件是本身某个元素具有的，事件函数是我们加上去的，事件并不是我们主动加的，是本身就有的，默认值是null，只是触发事件之后会发生什么由我们来决定，这时就用到了我们添加的事件函数）

元素只要符合某种事件的触发条件（比如鼠标点击），事件就必定会触发。

我们所谓的“加”上去的是 JavaScript 对事件的处理函数（没有加的话就不会对事件做任何处理，但并不等于说事件就没有触发）。

6. 获取DOM元素：

querySelector() 方法

返回文档中匹配指定 CSS 选择器的一个元素（括号里面写的是CSS的选择器）

返回值是一个元素，单个值

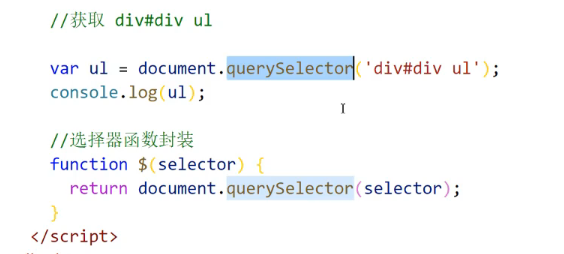
querySelectorAll() 方法

返回文档中匹配指定 CSS 选择器的所有元素

返回值是一个数组，是复数。



以后我们会经常用到选择器函数封装。如下例：



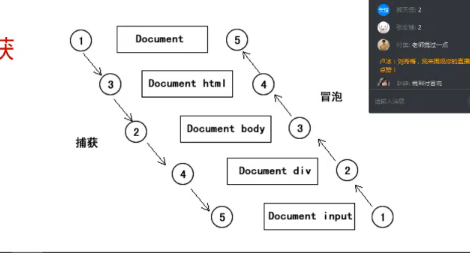
7. 事件流：描述从页面接受事件的顺序

当几个都具有事件的元素层叠在一起的时候，那么你点击其中一个元素，并不是只有当前被点击的元素会触发事件，而层叠在你点击范围的所有元素都会触发事件。

事件流包括两种模式：冒泡和捕获（冒泡就是从里向外执行，向上传递，捕获就是从外向里执行，从上往下走）

>现代浏览器默认情况下都是冒泡模型，默认就是冒泡形式。

>IE 不支持捕获，只支持冒泡（旧版本IE浏览器）



8.冒泡：

JavaScript 中的事件冒泡

事件按照从目标元素到根元素（document对象）的顺序触发。

即事件发生的顺序为：button — div — body — html — document

使用 stopPropagation() 方法阻止事件冒泡

event.stopPropagation();

9. 事件对象：

一般地，事件在传递过程中会有一些信息，这些是事件的组成部分：事件发生的时间+事件发生的地点+ 事件的类型+事件的当前处理者+其他信息。这些信息被封装到 Event 对象中。事件触发是有顺序的，所以在某一个时刻，只有一个事件对象，当某个事件执行完了，它的事件对象也就消失了。例：如果事件是按冒泡顺序触发的，那么事件对象的顺序也是冒泡顺序，向上传递的。

10.冒泡事件：

event.bubbles

返回一个布尔值，表明当前事件是否会向 DOM 树上层元素冒泡

onmouseover onmouseout —— 支持冒泡

onmouseenter onmouseleave —— 不支持冒泡

不是所有的事件都是支持事件冒泡的。

11. 冒泡应用 —— 事件委托（event delegation）

需求 —— 如果想要在大量子元素中单击任何一个都可以运行一段代码，可以将事件监听器设置在其父节点上，并将事件监听器的影响设置为每个子节点，而不是每个子节点单独设置事件监听器。

事件委托的原理：很多事件是支持冒泡的。

12. element.addEventListener(type， listener[， useCapture])

useCapture：布尔值，指定事件是否在捕获或冒泡阶段触发

true，指定事件在捕获阶段执行

false，指定事件在冒泡阶段执行，默认

13.事件处理周期：

触发一个事件后，在 HTML 元素间进行传播过程

第一阶段：事件的捕获，事件对象沿 DOM 树向下传播

第二阶段：目标触发，执行事件监听函数（目标元素，就不区分是捕获还是冒泡了，按定义顺序来执行）

第三阶段：事件冒泡，事件沿 DOM 树向上传播



14.进程是系统分配内存的单位，一个进程是由一个或多个线程组成。

系统分配的内存（独立的一块内存）

进程之间相互独立

多个线程在进程中协作完成任务

一个进程由一个或多个线程组成

同一进程下的各个线程之间共享程序的内存空间

15. 浏览器是多进程的

Browser 进程 （浏览器的主进程，由它来控制浏览器的协作，比如浏览器的前进，后退等）

Plugin 进程

GPU 进程

Renderer 进程（渲染进程）

以谷歌浏览器为标准来讲解。

16. Renderer 进程 —— 浏览器内核（浏览器的差异主要体现在这些方面）

GUI 渲染线程：渲染HTML，CSS代码

JS 引擎线程：用来执行JS代码的（功能是操作DOM，改变HTML，CSS的，所以不能和GUI渲染线程同时执行）

事件触发线程：管理我们的事件执行的

定时触发器线程：管理定时器的

HTTP 请求线程

线程之间相互合作，但是GUI 渲染线程与 JS 引擎线程互斥，不能同时执行。

17. JavaScript引擎线程：

JavaScript引擎线程是单线程

JavaScript 的主要用途是与用户互动，以及操作DOM（改变HTML和CSS）。

假定 JavaScript 同时有两个线程，一个线程在某个 DOM 节点上添加内容，另一个线程删除了这个节点，这时浏览器就不知道该以哪个线程为准，所以JS引擎是单线程的。

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。

18. JS代码并不是都是按照书写顺序执行的，其实定时器就是异步执行的。

19. 如果前一个任务耗时很长，后一个任务就不得不一直等着。为了解决这个问题，提出了异步。

20. 同步/异步任务：

任务可以分成两种，一种是同步任务，另一种是异步任务。

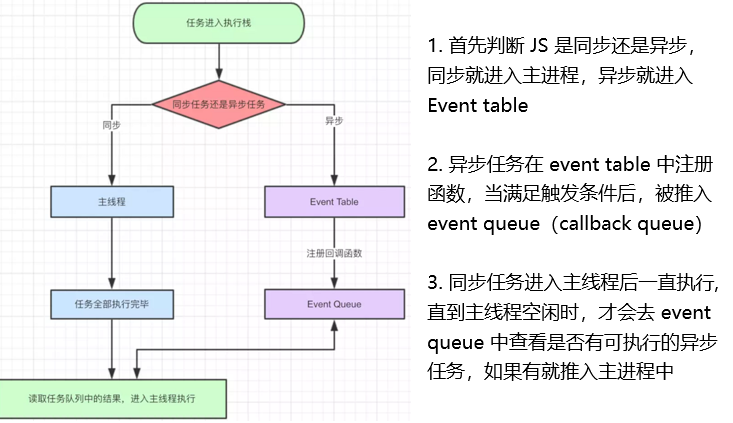
》同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；（如console.log、for循环等等）

》异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有"任务队列"通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。（如定时器、Ajax HTTP请求、一些（点击）事件函数等都属于异步任务），主线程只做同步任务。执行过程：**比如5秒后要执行定时器，先将定时器放入任务队列去等待时机，主线程先执行同步任务，当5秒的时间到了，任务队列通知主线程，定时器这个异步任务可以执行了，再把定时器这个异步任务的回调函数放到消息队列的消息里面，等到所有同步任务执行完了，就会去消息队列中取任务，取到一个最先进入的任务后放到主线程执行，执行完之后再去消息队列中取，再执行。**

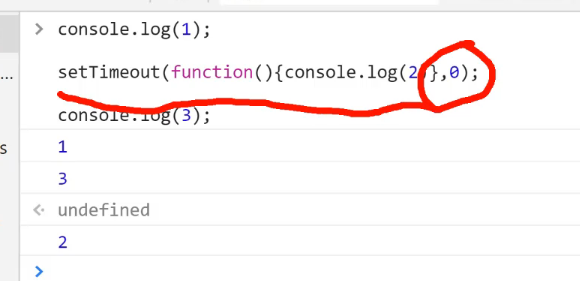
21.时间轮询：

消息队列：消息队列是一个先进先出的队列，它里面存放着各种消息（任务）。（放着的是可以执行的消息）

事件循环：事件循环是指主线程重复从消息队列中取消息、执行的过程。

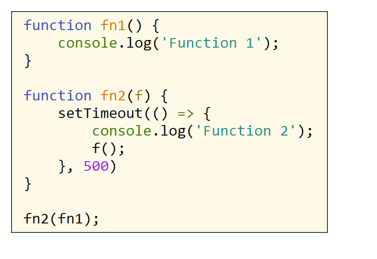


22.例：



并不是定时器设置了0秒执行，就先执行定时器，而是先执行主线程上的同步任务，等到所有同步任务执行完之后，才会去消息队列中取任务。

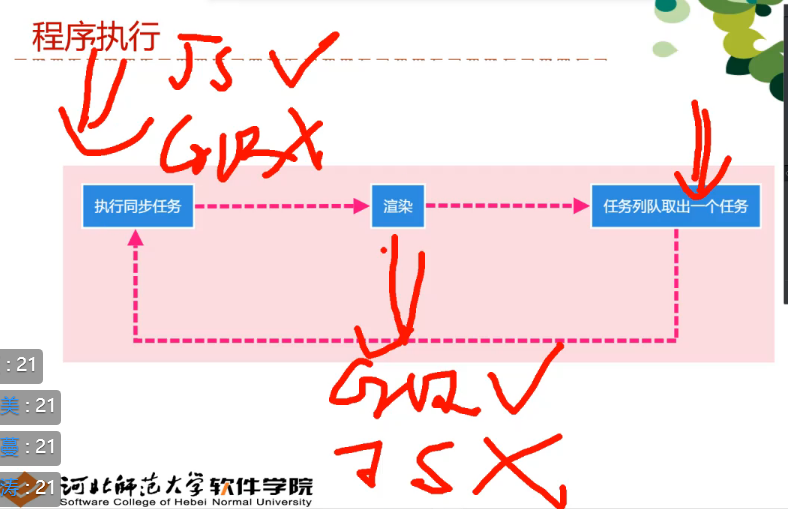
23.例：输出结果为Function 2 Function1



首先调用fn2时，因为定时器是异步任务，所以先放到任务队列当中，但是此时并没有同步任务，所以等待时机到了，从消息队列中取出任务执行，这是先输出Function 2，然后调用fn1，输出Function 1。

24. GUI 渲染线程与 JS 引擎线程互斥，不能同时执行。

以下为程序执行过程： 具体看课件例题 第17章 DOM渲染部分。



第十八章

1. 什么是 Javascript 库？

Javascript 库封装了很多预定义的对象和实用函数，能帮助使用者轻松建立有高难度交互的Web2.0特性的客户端界面，并且兼容各大浏览器。

jQuery 是一个优秀的轻量级的 Javascript 库类，是由John Resig 创建于 2006 年 1 月的开源项目（github上可以看到原码）。

一般原生JS是相对于我们的JS库来说的，原生JS就是我们之前学的JS，符合W3C的规范。以后写程序的时候要注意客户的需求，是使用原生JS来写，还是用JS库（比如jQuery）来写。

2. jQuery 的特点

轻量级（一般只有几K左右）

强大的选择器（能够准确的获取我们想要的元素）

出色的 DOM 操作的封装（能够快速方便的完成我们想要的效果）

可靠的事件处理机制

出色的浏览器兼容性（从而在jQuery中的事件流只有冒泡，没有捕获，IE不支持捕获）

3. jQuery 的核心理念是 write less，do more（写的少，做的多）

jQuery官网：www.jquery.com

4.在jQuery中获取一个节点可以使用$()来获取。



5. jQuery 编程步骤：

-引入 jQuery 的 js 文件（使用相对路径去引进）

-使用选择器定位要操作的节点

-调用 jQuery 的 DOM 方法进行操作

6. 例：

$(“button”)的返回值的类型是一个对象。

当一个函数的参数或者返回值还是一个函数时，为高阶函数，这里的click就是高阶函数。

这里的css方法是以行内样式的方式来使用的。



7. jQuery选择器：

jQuery 选择器语法

$(selector).action()

-美元符号: 定义 jQuery 对象，等价于 jQuery 对象（$符就等价于是jQuery，把$符换成jQuery也是可以的，如$(“#show”) === jQuery(“#show”)，这样写也对）

-选择符（selector）:“查询”和“查找” HTML 元素

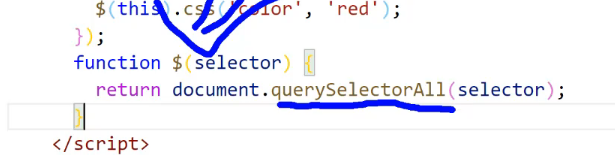
-jQuery 的 action() 执行对元素的操作

基本功能

-快速、精确地选择某一个（某一组）元素

-返回该元素（组）的 jQuery 包装集（集合）

8.自己来封装$符号：



9.jQuery选择器划分：

基本、层次、过滤、表单选择器

10. 基本选择器

-标签选择器： var div = $(‘div’);

-类选择器： var className = $(‘.toHidden’);

-ID选择器： var toExten = $(‘#toExten’);

-群选择器： var nodes = $(‘div, p, ul, ol’); （群组选择器用引号引上，中间用逗号隔开）

-通用选择器： var anyNodes = $(‘\*’);

如//选择所有 span 元素和 id 为 two 的元素

$(‘span,#two’).css(‘background’,’#bfa’)

11.在jQuery的文件中，如jquery.min.js这个文件，文件名中带有min的是压缩文件，里面的空格什么的都没有了。

12. 层次选择器（写法和CSS中的类似）

-后代选择器： var td = $("table td")

-子选择器： var td = $("table > td")

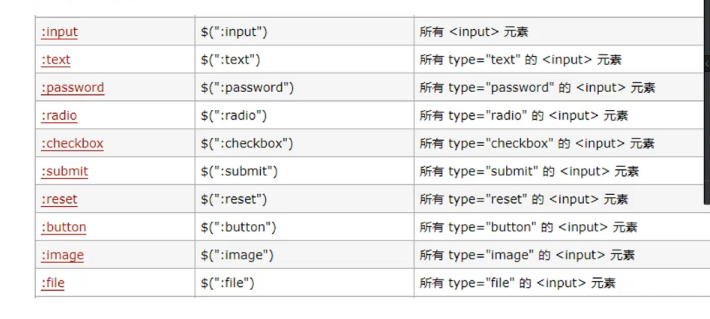
-兄弟选择器： var p = $("div + p");

13. 过滤选择器 —— 索引 index 从0开始



:not（选择器）是否定选择器，可以过滤出我们想要的元素。

14. 表单选择器：都是根据表单来说的。



15.

jQuery 包装集

-使用 jQuery 选择器获取的对象为 jQuery 包装集（jQuery 对像）

-在 jQuery 对象中无法使用 DOM 对象的任何方法

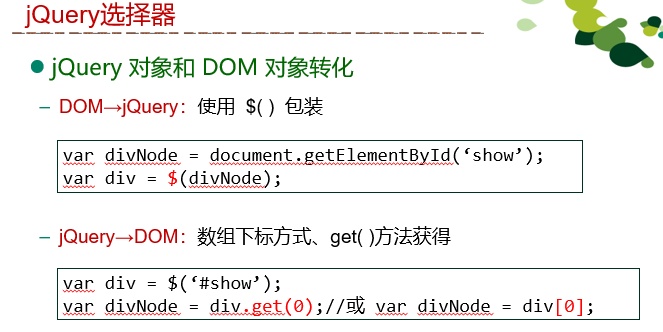
-反之 DOM 对象也无法使用任何 jQuery 的方法

注：jQuery包装集和原生的DOM元素不是等价的，jQuery包装集只能使用jQuery的方法，不能使用原生DOM的方法，DOM 对象也无法使用任何 jQuery 的方法，但是jQuery对象和原生DOM 对象可以相互转化。

包装集的 length 属性

-作用：获取元素个数

-使用：$(“.username”).length



16. jQuery 中的 DOM 操作

第一步：获取节点 （强大的选择器）

第二步：DOM操作

文本内容

属性操作 （指的是HTML标签里面的属性，如id属性、class属性）

样式操作 （指的就是样式，比如背景颜色啊，等等）

遍历节点

节点操作

位置操作

17. jQuery的DOM操作：

（1）文本内容

获取: html() 设置: html(value) 类似 于原生js的innerHTML 属性的用法（非表单元素使用）

获取: val() 设置: val(value) 应用于表单元素

html()获取的是集合中第一个匹配元素的HTML内容。



（2）属性操作

获取: attr(attributeName)

设置: attr(attributeName，value)

attr({

attributeName:value,

attributeName:value

})

删除: removeAttr("title")

removeAttr("title id class")

（3）样式操作

jQuery 样式操作可以对行内样式、页内样式、外部样式操作

获取: css("propertyName") css([propertyName,propertyName])

设置: css("propertyName"，"value")

css({

propertyName : value,

propertyName : value

})

操作样式除了 css() 方法之外，我们还可以把样式写在 class 中，通过添加、切换、删除 className 的方式操作样式

添加: addClass("red") addClass("red green blue")

删除: removeClass(“red”) removeClass(“red green”) removeClass() 当传值为空时，表示删除所有的类名

判断: hasClass("red")

切换: toggleClass("red") 自动检测类名，有的话删除，没有就添加，从而进行类名的切换

（4）遍历节点操作

children() —— 获取匹配元素的子元素集合 （用的最多之一）

eq(index) —— 获取匹配元素的集合中指定索引的那一个元素 （用的最多之二）

parent() —— 获取匹配元素集合中，每个元素的父元素（第一个父）

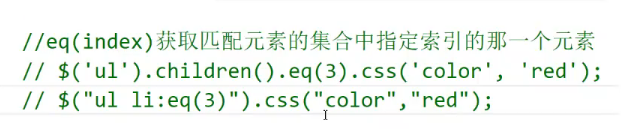
parents() —— 获得集合中每个匹配元素的祖先元素（所有祖先元素）

next() —— 获得匹配的元素紧邻的后面同辈的元素 （就只有后面紧邻的那个）

nextAll() —— 获得匹配元素所有下面的同辈元素 （所有同辈元素）

siblings() —— 获得匹配元素集合中每个元素的兄弟元素（选中除了本身之外所有的同辈元素） （用的最多之三）

prev() —— 获取匹配元素紧邻的前一个同辈兄弟元素



以上两种写法作用相同。

（5）节点操作

创建节点: var li = $("<li title='香蕉' >香蕉</li>") ;

插入节点: 一般都是在末尾处插入

$("p").append("<span>最后一个子节点</span>")

$("<span>最后一个子节点</span>").appendTo("p") （这两种写法作用差不多）

删除节点: $("ul li:eq(1)").remove() $("ul li").remove(":eq(1)")

替换节点: $("p").replaceWith("<span>替换节点</span>")

如下例：

jQuery本身是支持链式写法的。







（6）位置操作

offset()

相对于当前窗口的偏移距离（注意是相对于窗口左上角的偏移距离）

返回值为对象，含有 top、left 两个属性

position()

相对于定位父元素的偏移距离 （相对于父元素的偏移距离）

返回值为对象，含有 top、left 两个属性

滚动条的相关效果：（一般水平的滚动条用不到，垂直的用的较多）

scrollTop() —— 获取匹配的元素集合中第一个元素的当前垂直滚动条的位置

scrollTop( value ) —— 设置每个匹配元素的垂直滚动条位置

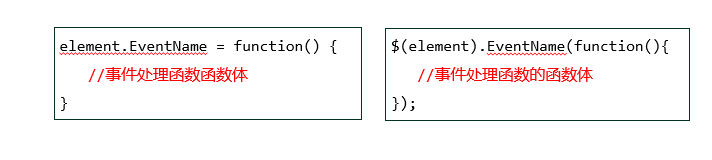
scrollLeft() —— 获取匹配的元素集合中第一个元素的当前水平滚动条的位置

scrollLeft( value ) ——设置每个匹配元素的水平滚动条位置

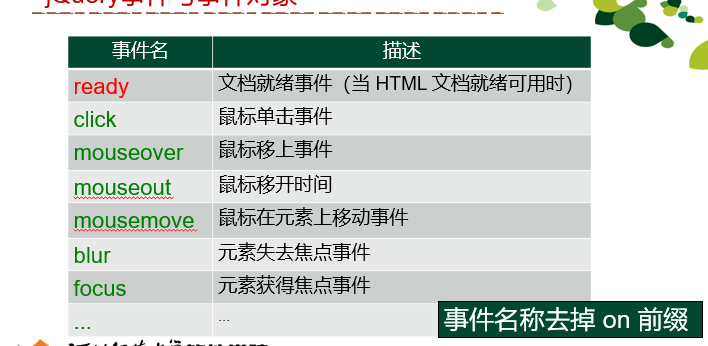
18. jQuery事件与事件对象

事件是指可以被 JavaScript 侦测到的行为。

JavaScript 原生事件和 jQuery 中的事件是一致的，唯一不同是调用和处理事件的方式不同。



Ready事件经常和window对象的onload事件做对比。



$(document).ready( ) 方法和 window.onload 事件功能相似，但是也有区别：

》执行时机

window.onload: 在网页中所有的元素（包括所有关联文件）完全加载到浏览器后执行

$(document).ready( ): 在DOM完全加载以后就可以调用（HTML结构加载完成就会执行）

ready比onload执行时机靠前

》事件编写个数

window.onload 事件不能同时编写多个（写了多个的话，生效的只有一个，其它的是不生效的）

$(document).ready( ) 方法能同时编写多个（多个都会生效，会按书写顺序依次执行）

$(document).ready(function (){

alert（“hello world”）;

})

$(document).ready(function

(){

alert（“hello again”）;

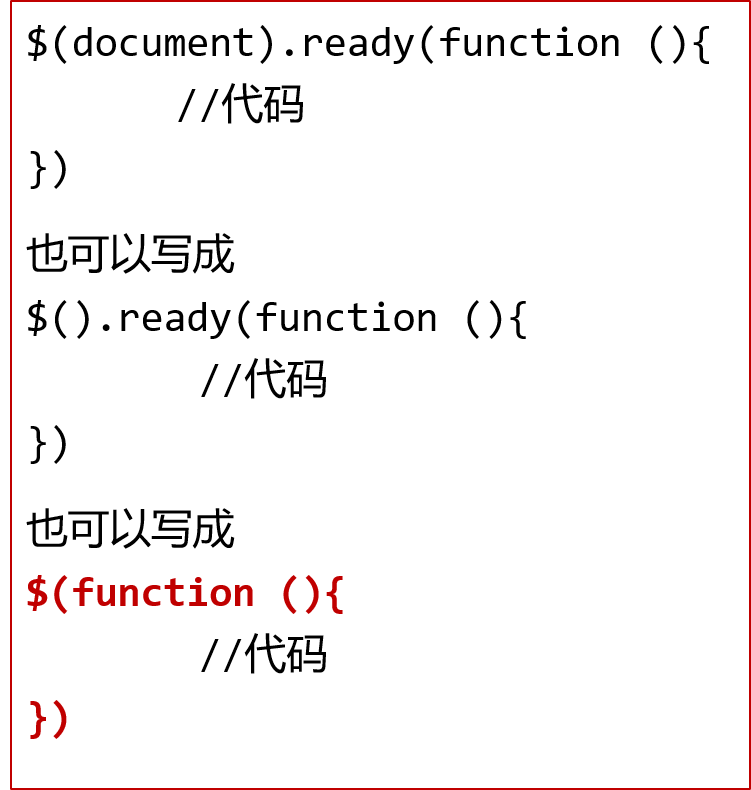
})

结果两次都输出

》简化写法

window.onload 事件没有简化写法

$(document).ready( ) 方法有简化写法



19. jQuery事件与事件对象

给对象添加事件方法

$("selector") . on( EventName ，handler）

$("selector") . EventName ( handler );

移出事件方法

$("selector") . off( events [, handler ]）

off() 方法移除用.on()绑定的事件处理程序

注：通过on添加事件，可以用off来移除事件。

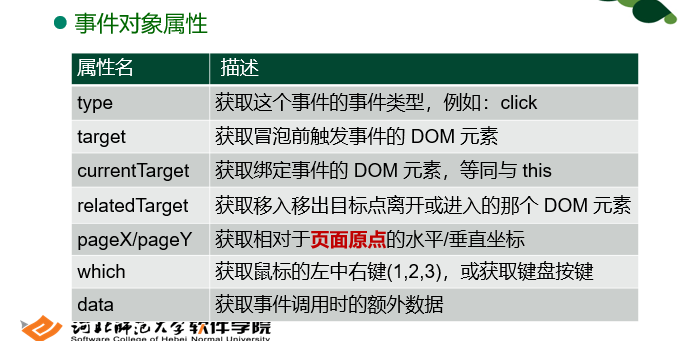
但是用$("selector") . EventName ( handler );这种方式添加事件，无法用off移除。

hover(fn1，fn2)

当鼠标指针进入和离开元素时被执行

fn1和fn2都是函数。fn1中写鼠标移入代码，fn2中写鼠标移出代码。

20. jQuery 在封装 event 对象的时候，解决了兼容性问题，并且还创建了一些非常好用的属性和方法。



21. 事件对象方法

阻止冒泡行为 event.stopPropagation()

阻止默认行为 event.preventDefault()

22.show()和hide()其实就是控制display属性。

23.animate进行动画效果的实现是改变我们的行内样式。

24. 只有数字值可以创建动画（比如我们常见的width，height，top，opacity，left等），字符串值无法创建动画（比如'background:red')。

25.animate参数中最后一个参数是一个回调函数，在前面所有的动画效果执行完之后才会去调用这个回调函数。

第十九章

1.ES6是2015年发布的，ES5，ES6等这些都是一个规范。到现在为止，主流浏览器大部分已经支持了ES6的规范，但并不是所有。

2.ES6中新增的内容，大部分是用来解决目前已经发现的ES5中出现的问题或者是给我们提供更加快捷的方式。如变量声明，var关键字存在一定的问题：在循环中，存在变量污染；存在声明提升；在全局作用域下，使用var关键字生命的变量会挂载在widow对象下。因此ES6中出现了let和const。

3.变量声明：ES6 中新增了let和const。let不允许重复声明，指的是当你已经用const或者var声明了变量，就不能再用let去声明变量。或者是当你已经用let声明了变量，就不能再用var或者const再去声明变量。

4.解构赋值的出现是因为JSON 数据应用比较广泛。

5.解构赋值中，一旦有了冒号，真正的赋值是后面的变量。冒号前面的内容只是用来模式匹配。当属性值严格等于 undefined，默认值才会生效。

6.扩展运算符用三个点表示。功能是把数组或类数组对象展开成一系列用逗号隔开的值。

扩展运算符可以进行数组，对象的合并。

7.rest运算符也是三个点，但是是把逗号隔开的值序列组合成一个数组，与扩展运算符作用相反。rest运算符是用于获取函数的多余参数，是获取剩下的所有的值，一般写在后面，而且也不能写两个。

8.模板字符串是用反引号（` `）表示。在键盘左上角1键前面的键。在反引号里面出现的变量要写成${变量}的形式，表达式也是按照变量的写法，而字符串就可以直接写。

9.includes（）和startsWith（）都是从开始位置进行搜索，而endsWith（）是表示参数字符串是否在原字符串的尾部。第二个参数表示开始搜索的位置。具体内容看课件。

10. 创建一个函数的（函数定义）三种书写形式：函数声明；函数表达式；构造函数

11.箭头函数：是一个匿名函数。

我们一般用then来获取成功，用catch来获取失败。