JS学习笔记

数据类型

变量

```
//全局变量
i = 1;  //通过

//严格检查
'use strict'
i = 1;  //不通过

//局部变量使用let定义
let j = 2;
```

比较运算符

```
=

==  等于（类型不一样，值一样，会判断为true）

=== 绝对等于（类型一样，值一样，结果为true）
```

注意点：

- **这是一个**JS**的缺陷，坚持使用**===**比较**
- NaN===NaN，**结果为**false，NaN**与所有数值不相等，包括自己**
- **只能通过**isNaN(NaN)**来判断这个数是否为**NaN

浮点数

尽量避免使用浮点数数值运算，存在精度问题。

数组

```
var arr = [1,2,3,"hello",null,true];

var elem = new Array(1,2,3,"hello",null,true);
```

对象

```
var person = {
    name: "wll",
    sex: "男",
    age: 20,
    hobby: ['1', '2', '3']
};
//取值
person.name
person.hobby[0]
person['age']
```

字符串

- **多行字符串：**

```
//反引号包裹
let str = `this
    hello
    我
`;
```

- **模板字符串**

```
let name = "wll";
let age = 20;
let msg = `"person:"${ name, age }`;
console.log(msg);
```

for循环

forEach循环

```
let po = [1, 2, 3, 4, 5, 6, 7, 8];
po.forEach(function(value) {
    console.log(value);
})
```

for...in

```
for (let i in po) {
    console.log(po[i]);
}
```

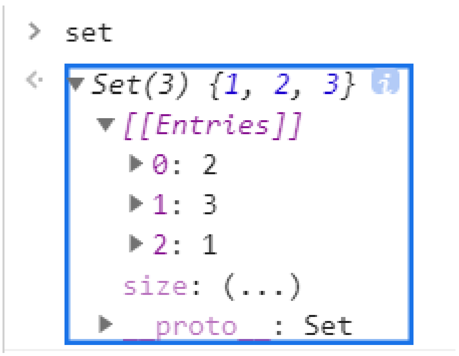
Map和Set

Map

```
let map = new Map([
    ['tom', 20],
    ['jack', 21]
]);
let r = map.get('tom');
console.log(r);
```

Set

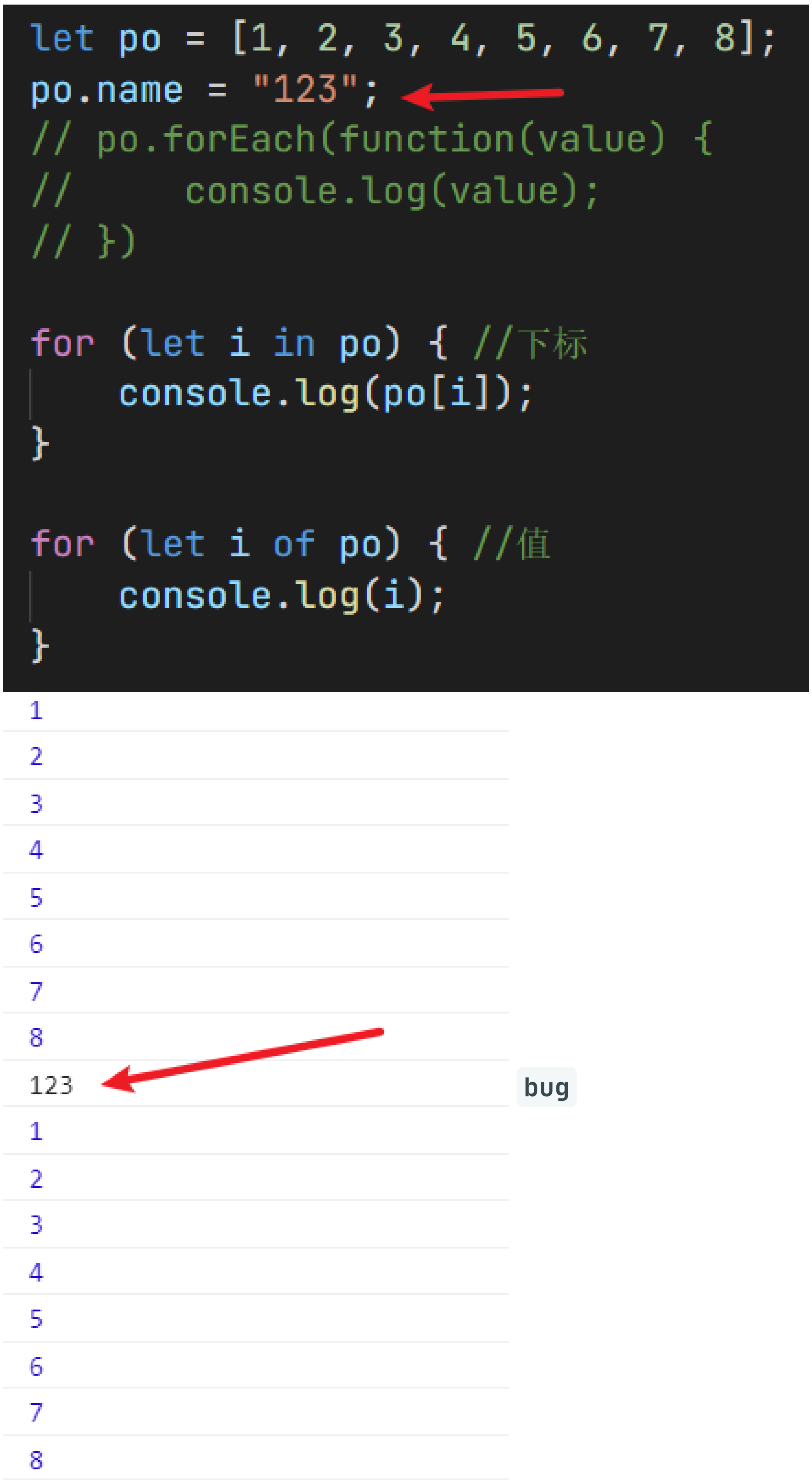
```
let set = new Set([1, 2, 3, 3, 3, 3]);  //可以去重
```



```
> set
<· ▼Set(3) {1, 2, 3}
    ▼[[Entries]]
      ▶0: 2
      ▶1: 3
      ▶2: 1
     size: (...)
    ▶__proto__: Set
```

iterator

```
//for...in是下标，for...of是值，也可以便利Map和Set
for (let i of po) {
    console.log(i);
}
for (let i of set) {
    console.log(i);
}
```



```
let po = [1, 2, 3, 4, 5, 6, 7, 8];
po.name = "123";
// po.forEach(function(value) {
//      console.log(value);
// })

for (let i in po) { //下标
    console.log(po[i]);
}

for (let i of po) { //值
    console.log(i);
}
```

```
1
2
3
4
5
6
7
8
123
1
2
3
4
5
6
7
8
```

`bug`

函数

定义方式

```
//方式一
function abs(i) {
    if (i >= 0) {
        return i;
    } else {
        return -i;
    }
}
//方式二
let abs2 = function(i) {
    if (i >= 0) {
        return i;
    } else {
        return -i;
    }
}
```

参数问题

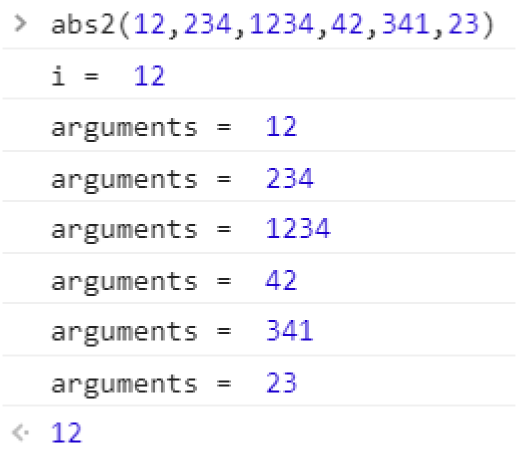
可以传任意个参数，也可以不传递参数，不会报错!

```
//手动抛出异常
function abs(i) {
    if (i !== 'number') {
        throw 'Not a number';
    }
    if (i >= 0) {
        return i;
    } else {
        return -i;
    }
}
```

arguments

保存传递进函数的参数

```
let abs2 = function(i) {
    console.log("i = ", i);
    for (let j in arguments) {
        console.log("arguments = ", arguments[j]);
    }
    if (i >= 0) {
        return i;
    } else {
        return -i;
    }
}
```

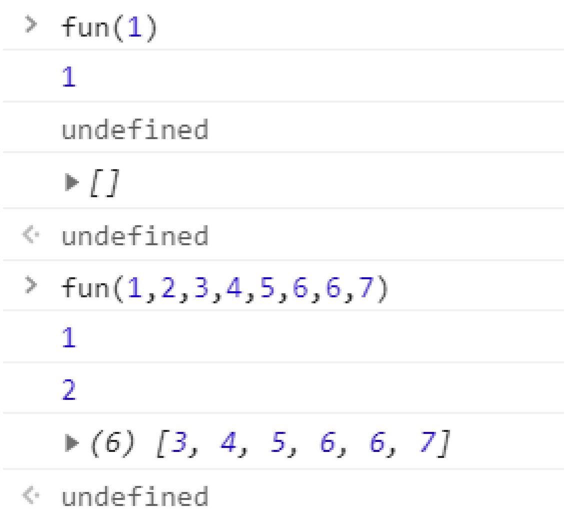


```
> abs2(12,234,1234,42,341,23)
  i =  12
  arguments =  12
  arguments =  234
  arguments =  1234
  arguments =  42
  arguments =  341
  arguments =  23
< 12
```

rest

获取除了已经定义的参数之外传入函数的所有参数

```
function fun(a, b, ...rest) {
    console.log(a);
    console.log(b);
    console.log(rest);
}
```



```
> fun(1)
  1
  undefined
  ▶ []
< undefined
> fun(1,2,3,4,5,6,6,7)
  1
  2
  ▶ (6) [3, 4, 5, 6, 6, 7]
< undefined
```

默认所有的全局变量都自动绑定在window对象下

```
let x = 1;
window.alert(window.x);

let po = window.alert();
window.alert = function(){};
```

由于所有的全局变量都会绑定到window上，如果不同的js文件，使用了相同的全局变量，如何减少冲突?

```
//唯一全局变量
let wll = {};
//定义全局变量
wll.name = 'wulele';
wll.add = function(a, b) {
    return a + b;
}
```

把自己的代码全部放入自己定义的唯一空间的名字中，降低全局命名冲突的问题。

方法

```
let wusir = {
    name: 'wll',
    birth: 2000,
    //方法
    age: function() {
        //获取当前年份
        let now = new Date().getFullYear();
        return now - this.birth;
    }
}
```

apply

```
function getAge() {
    var y = new Date().getFullYear();
    return y - this.birth;
}

var xiaoming = {
    name: '小明',
    birth: 1990,
    age: getAge
};

xiaoming.age(); // 25
getAge.apply(xiaoming, []); // 25, this指向xiaoming, 参数为空
```

内部对象

Date

```
let time = new Date();
time.getFullYear(); //年
time.getMonth(); //月  0~11代表月
time.getDate(); //日
time.getDay(); //星期几
time.getHours(); //时
time.getMinutes(); //分
time.getSeconds(); //秒
time.getTime(); //时间戳  1970-1-1~现在的毫秒数
console.log(new Date(time.getTime()));  //时间戳转时间

let now = new Date();
>undefined
now.toGMTString();
>"Wed, 24 Mar 2021 13:49:33 GMT"
now.toLocaleDateString()
>"2021/3/24"
now.toLocaleString()
>"2021/3/24下午9:49:33"   //前端展示
```

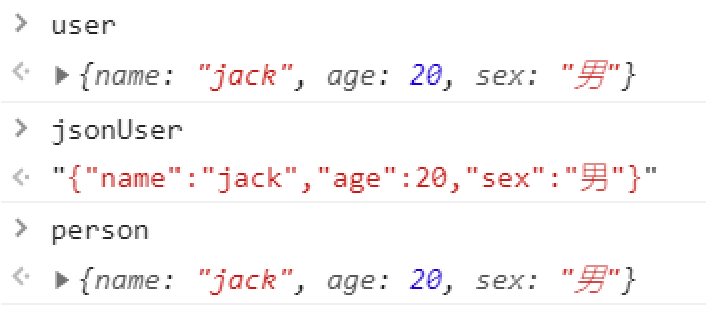
JSON

一种轻量级的数据交换格式。

JS中，任何JS支持的类型都可以用JSON来表示

```
格式：

• 对象都用{}
• 数组都用[]
• 所有的键值对都是用key:value
```

```
let user = {
        name: "jack",
        age: 20,
        sex: '男'
    }
    //对象转化为JSON字符串
let jsonUser = JSON.stringify(user);
//JSON字符串转对象
// let person = JSON.parse(jsonUser);
let person = JSON.parse('{"name":"jack","age":20,"sex":"男"}');
```

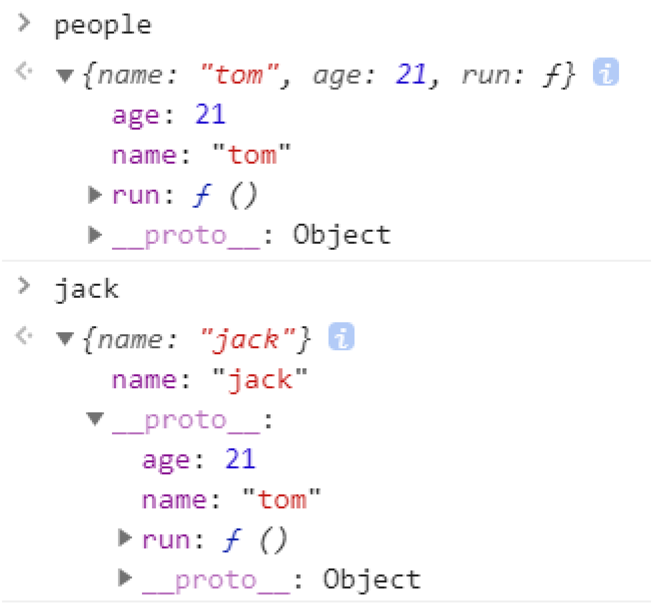


```
> user
< ▶{name: "jack", age: 20, sex: "男"}
> jsonUser
< "{"name":"jack","age":20,"sex":"男"}"
> person
< ▶{name: "jack", age: 20, sex: "男"}
```

面向对象编程

原型

```
let people = {
    name: "tom",
    age: 21,
    run: function() {
        console.log(this.name + "run...");
    }
};
let jack = {
    name: "jack"
};
//jack 的原型是 people
jack.__proto__ = people;

//ES6之前
function Student(name) {
    this.name = name;
}
//给Student新增一个方法
Student.prototype.hello = function() {
    alert('Hello')
};
```



```
> people
< ▼{name: "tom", age: 21, run: ƒ}
    age: 21
    name: "tom"
  ▶ run: ƒ ()
  ▶ __proto__: Object
> jack
< ▼{name: "jack"}
    name: "jack"
  ▼ __proto__:
      age: 21
      name: "tom"
    ▶ run: ƒ ()
    ▶ __proto__: Object
```

class继承

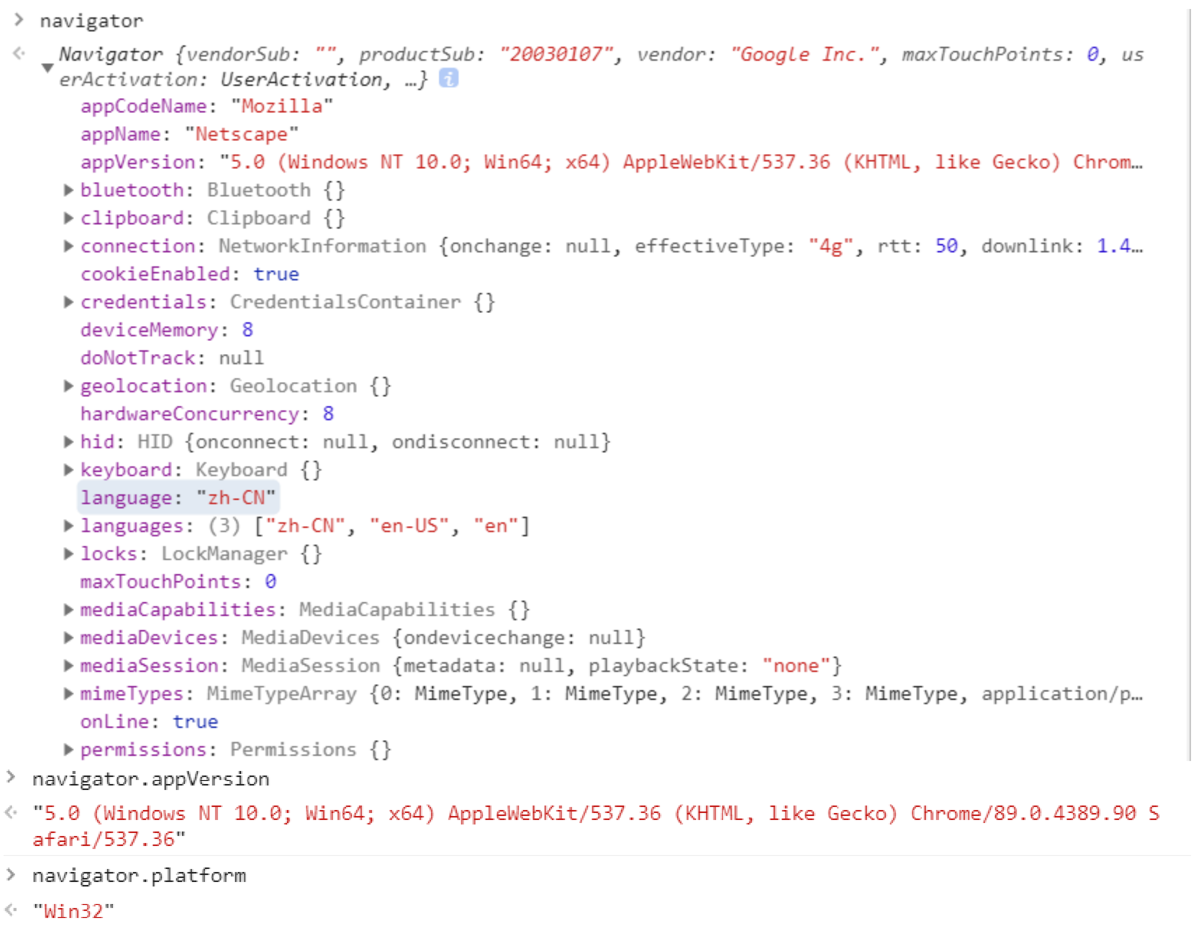
```
//ES6之后
class Student {
    constructor(name) {
        this.name = name;
    }
    hello() {
        alert(this.name);
    }
}
let student = new Student("tom");
```

继承

```
class Primary extends Student {
    constructor(name, grade) {
        super(name);
        this.grade = grade;
    }
}
```

操作BOM对象

navigator



```
> navigator
< ▼Navigator {vendorSub: "", productSub: "20030107", vendor: "Google Inc.", maxTouchPoints: 0, us
  erActivation: UserActivation, …}
     appCodeName: "Mozilla"
     appName: "Netscape"
     appVersion: "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom…
   ▶bluetooth: Bluetooth {}
   ▶clipboard: Clipboard {}
   ▶connection: NetworkInformation {onchange: null, effectiveType: "4g", rtt: 50, downlink: 1.4…
     cookieEnabled: true
   ▶credentials: CredentialsContainer {}
     deviceMemory: 8
     doNotTrack: null
   ▶geolocation: Geolocation {}
     hardwareConcurrency: 8
   ▶hid: HID {onconnect: null, ondisconnect: null}
   ▶keyboard: Keyboard {}
     language: "zh-CN"
   ▶languages: (3) ["zh-CN", "en-US", "en"]
   ▶locks: LockManager {}
     maxTouchPoints: 0
   ▶mediaCapabilities: MediaCapabilities {}
   ▶mediaDevices: MediaDevices {ondevicechange: null}
   ▶mediaSession: MediaSession {metadata: null, playbackState: "none"}
   ▶mimeTypes: MimeTypeArray {0: MimeType, 1: MimeType, 2: MimeType, 3: MimeType, application/p…
     onLine: true
   ▶permissions: Permissions {}
> navigator.appVersion
< "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 S
  afari/537.36"
> navigator.platform
< "Win32"
```

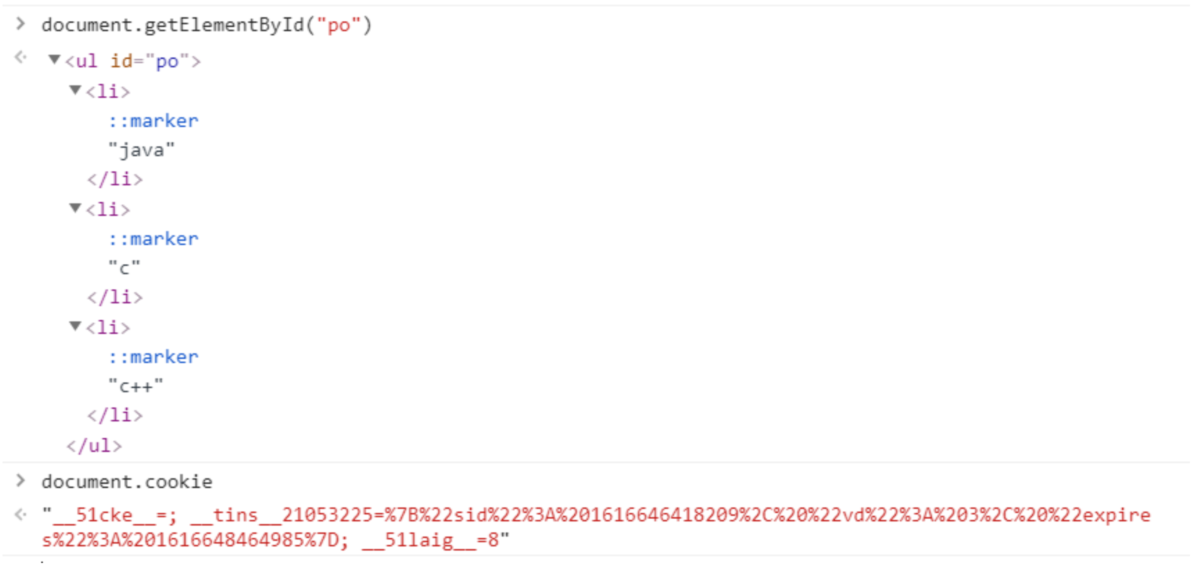
location

```
location.reload()    //刷新网页
location.assign("http://192.144.231.70:8080/")   //跳转
```

document

DOM**文档树**

```
<ul id="po">
    <li>java</li>
    <li>c</li>
    <li>c++</li>
</ul>
```



```
> document.getElementById("po")
<· ▼<ul id="po">
    ▼<li>
        ::marker
        "java"
      </li>
    ▼<li>
        ::marker
        "c"
      </li>
    ▼<li>
        ::marker
        "c++"
      </li>
    </ul>
> document.cookie
<· "__51cke__=; __tins__21053225=%7B%22sid%22%3A%201616646418209%2C%20%22vd%22%3A%203%2C%20%22expire
   s%22%3A%201616648464985%7D; __51laig__=8"
```

history

浏览器历史记录

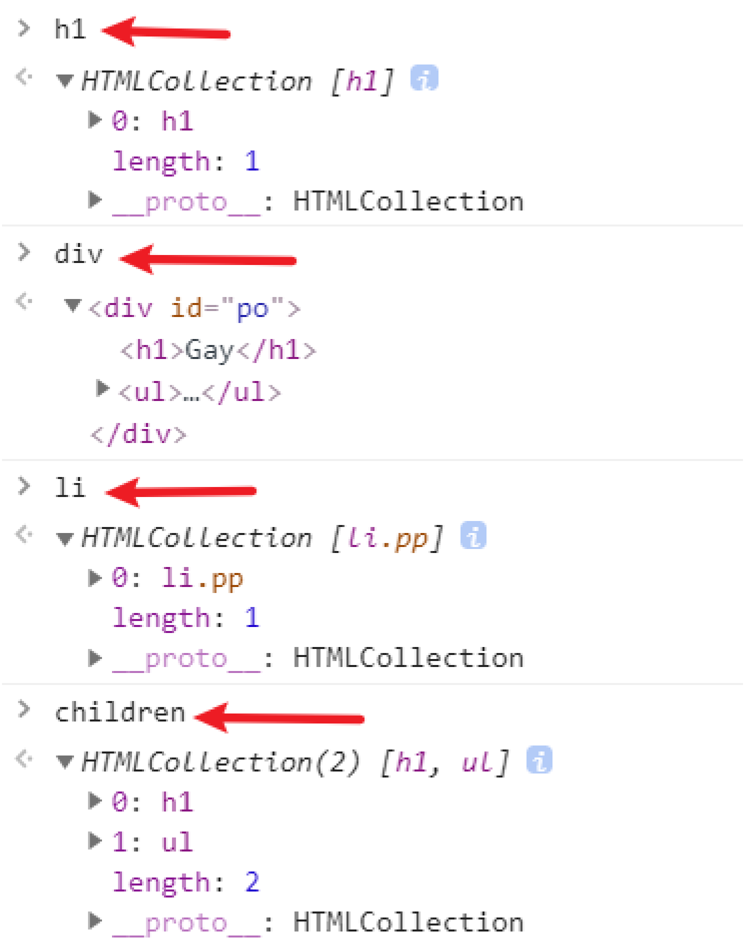
```
history.back()   //后退

history.forward()    //前进
```

操作DOM对象

获取DOM结点

```
<div id="po">
    <h1>Gay</h1>
    <ul>
        <li class="pp">java</li>
        <li>c</li>
        <li>c++</li>
    </ul>
</div>
```

```
let h1 = document.getElementsByTagName("h1");
let li = document.getElementsByClassName("pp");
let div = document.getElementById("po");
//  获取父节点下的所有子节点
let children = div.children;
let children1 = div.children[0];
```



```
> h1
< ▼HTMLCollection [h1]
    ▶0: h1
     length: 1
    ▶__proto__: HTMLCollection
> div
<  ▼<div id="po">
      <h1>Gay</h1>
     ▶<ul>…</ul>
    </div>
> li
< ▼HTMLCollection [li.pp]
    ▶0: li.pp
     length: 1
    ▶__proto__: HTMLCollection
> children
< ▼HTMLCollection(2) [h1, ul]
    ▶0: h1
    ▶1: ul
     length: 2
    ▶__proto__: HTMLCollection
```

更新DOM结点

```
<div id="po">

</div>
<script>
    let po = document.getElementById('po');
</script>
```

`po.innerText = '123'` //修改文本的值

`po.innerHTML = <strong>123</strong>` //解析HTML文本标签

`po.style.color = 'red'` //修改css样式

删除DOM结点

步骤：先获取父节点，再通过父节点删除

`let father = p1.parentElement` //获取父节点

`father.removeChild(p1)` //删除子节点

创建和插入DOM结点

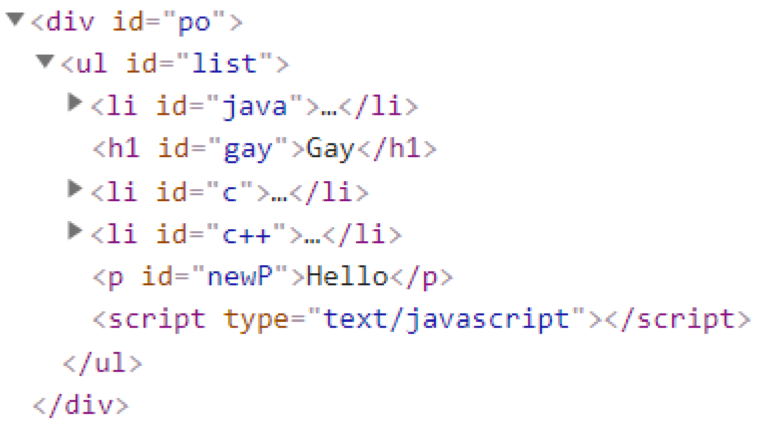
追加

```html
<div id="po">
    <h1 id="gay">Gay</h1>
    <ul id="list">
        <li id="java">java</li>
        <li id="c">c</li>
        <li id="c++">c++</li>
    </ul>
</div>
```

```javascript
let gay = document.getElementById("gay");
let list = document.getElementById("list");
list.append(gay); //追加到后面
```

创建

```javascript
let gay = document.getElementById("gay");
let list = document.getElementById("list");
list.append(gay); //追加到后面

let newP = document.createElement('p');
newP.id = 'newP';
newP.innerText = "Hello";
list.append(newP);
let myScript = document.createElement('script');
myScript.setAttribute('type', 'text/javascript');
list.append(myScript);
let c = document.getElementById('c');
list.insertBefore(gay, c); //追加到list里面的c前面
```



```
▼<div id="po">
  ▼<ul id="list">
    ▶<li id="java">…</li>
     <h1 id="gay">Gay</h1>
    ▶<li id="c">…</li>
    ▶<li id="c++">…</li>
     <p id="newP">Hello</p>
     <script type="text/javascript"></script>
   </ul>
 </div>
```

操作表单

```
    <form action="get">
        <span>用户名: </span><input type="text" id="username">
        <p>爱好</p>
        <input type="checkbox" name="hobbies" id="gay"
value="gay">gay
        <input type="checkbox" name="hobbies" id="computer"
value="computer">computer
        <input type="checkbox" name="hobbies" id="game"
value="game">game
    </form>

    <script>
        let username = document.getElementById('username');
        let gay = document.getElementById('gay');
        username.value = '456'; //修改输入框的值
               //对于单选框radio、多选框checkbox等固定的值，value只能取
得当前的值：true（选中）false（未选中）
        gay.checked = 'true'; //选中
    </script>
```

表单提交、加密

```
    <!-- onsubmit绑定一个函数，函数将结果返回给表单，使用onsubmit接受 -->
    <form action="#" method="POST" onsubmit="return check()">
        <p>
            <span>用户名: </span><input type="text" id="username"
name="username">
        </p>
        <p>
            <span>密码: </span><input type="password" id="false-
password">
        </p>
        <input type="hidden" name="password" id="password">
        <button type="submit">登录</button>
    </form>
    <script>
        function check() {
            let username = document.getElementById('username');
            let falsePassword = document.getElementById('false-
password');
            let password = document.getElementById('password');
            //密码加密
            password.value = md5(falsePassword.value);
            //校验表单，true表单提交，false表单不提交
            return true;
        }
    </script>
```

jQuery

格式

```
<a href="" id="test">点我</a>
```

```
// $(selector).action()
        $('#test').click(function() {
            alert('Hello');
        })
```

选择器

```
https://jquery.cuishifeng.cn/index.html
```

鼠标响应事件

```
<style>
        #div {
            width: 500px;
            height: 500px;
            border: 2px solid rebeccapurple;
        }
</style>
<p id="po"></p>
<div id="div"></div>
```

```
<script>
        //  当网页元素加载完后响应事件。也可写为:
$(document).ready(function(){});
        $(function() {
            $('#div').mousemove(function(e) {
                $('#po').text('x=' + e.pageX + 'y=' + e.pageY);
            });
        });
    </script>
```

操作DOM

```
$('#ul li[name=python]').text();    //获得值
$('#ul li[name=python]').text('123123');    //设置值
$('#ul').html();    //获得值
$('#ul').html('<strong>123</strong>');  //设置值

$('#ul li[name=python]').css({"color","red"});  //操作css
$('#ul li[name=python]').show();    //显示
$('#ul li[name=python]').hide();    //隐藏    display:none
```