互联网软件开发技术与实践

JavaScript基础

《互联网软件开发技术与实践》

课程建设小组

北京大学

二零一九年　北京

北京大学

JavaScript和Java的区别

- 这两个没有任何的血缘关系，Java是由Sun 公司于1995年5月推出的，而JavaScript是于1995年由Netscape公司设计实现而成的，由于Netscape公司与Sun公司合作，Netscape高层希望它看上去能够像Java，因此取名为JavaScript。
- Java是SUN公司推出的新一代面向对象的程序设计语言，特别适合于Internet应用程序开发；
- JavaScript是Netscape公司的产品，其目的是为了扩展Netscape Navigator（一款网络浏览器）功能，而开发的一种可以嵌入Web页面中的基于对象和事件驱动的解释性语言。

不安全 | view-source:www.ss.pku.edu.cn/index.php/educatior

```
24    <script src="/media/jui/js/jquery.min.js" type="text/javas
25    <script src="/media/jui/js/jquery-noconflict.js" type="tex
26    <script src="/media/jui/js/bootstrap.min.js" type="text/ja
27    <script type="text/javascript">
28  window.addEvent('load', function() {
29                  new JCaption('img.caption');
30              });jQuery(document).ready(function()
31                  {
32                      jQuery('.hasTooltip').tooltip({});
33                  });
34    </script>
35
36      <script type="text/javascript">
37        var timeout         = 500;
38        var closetimer      = 0;
39        var ddmenuitem      = 0;
40        var linkitem = 0;
```

北京大学

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, World!</title>
    <script>
        function SayHello() {
            document.getElementById("myMessage").innerHTML = "Hello, World!";
        }
    </script>
</head>
<body>
    <h1>第一个 JavaScript 程序</h1>
    <p id="myMessage">试试点击下方的按钮...</p>
    <button type="button" onClick="SayHello()">点我</button>
</body>
</html>
```

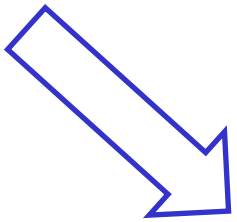
程序运行效果

localhost:63342/web/04_JavScript实验

第一个 JavaScript 程序

试试点击下方的按钮...

点我



localhost:63342/web/04_JavScript实验/01_Jav

第一个 JavaScript 程序

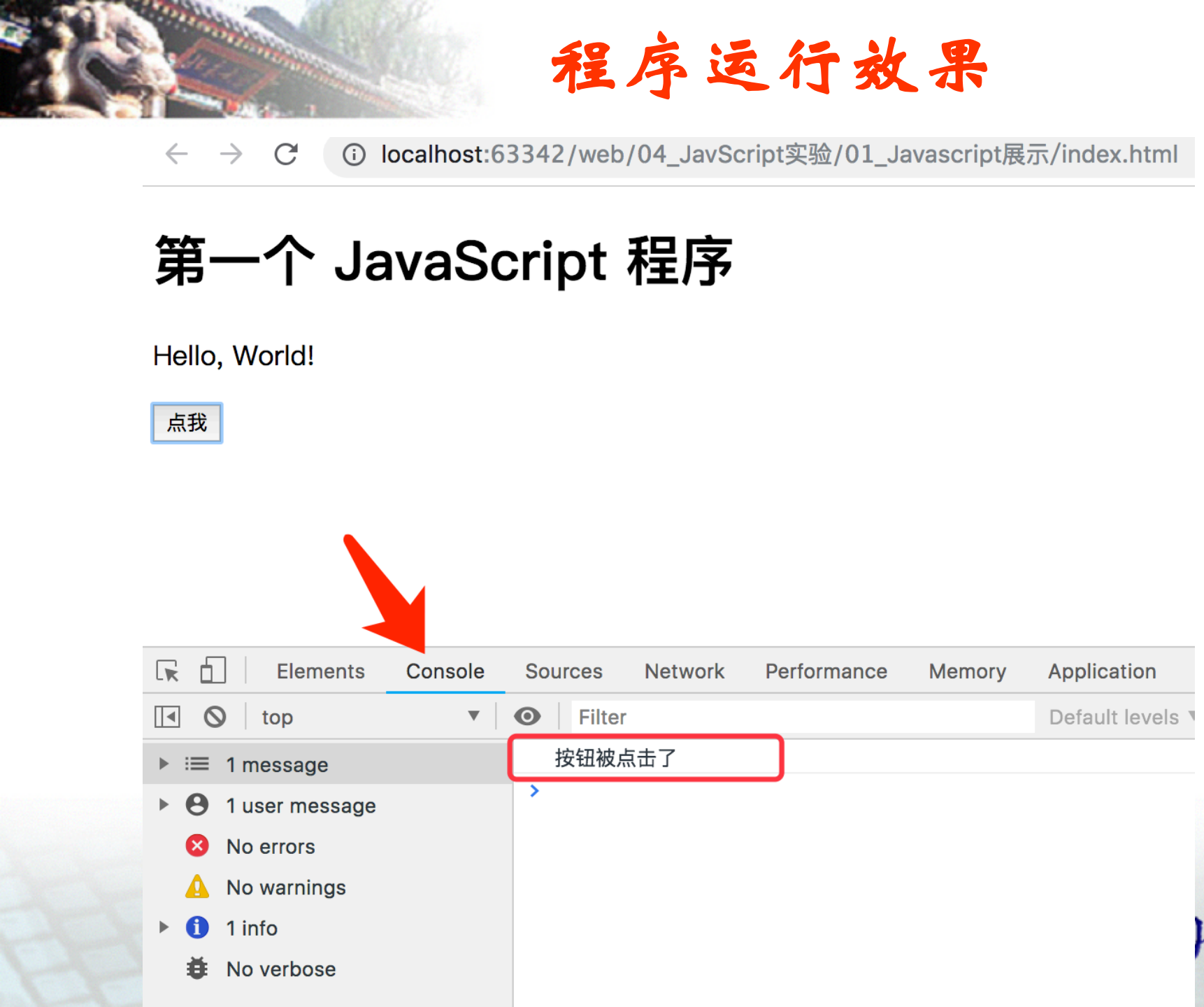
Hello, World!

点我

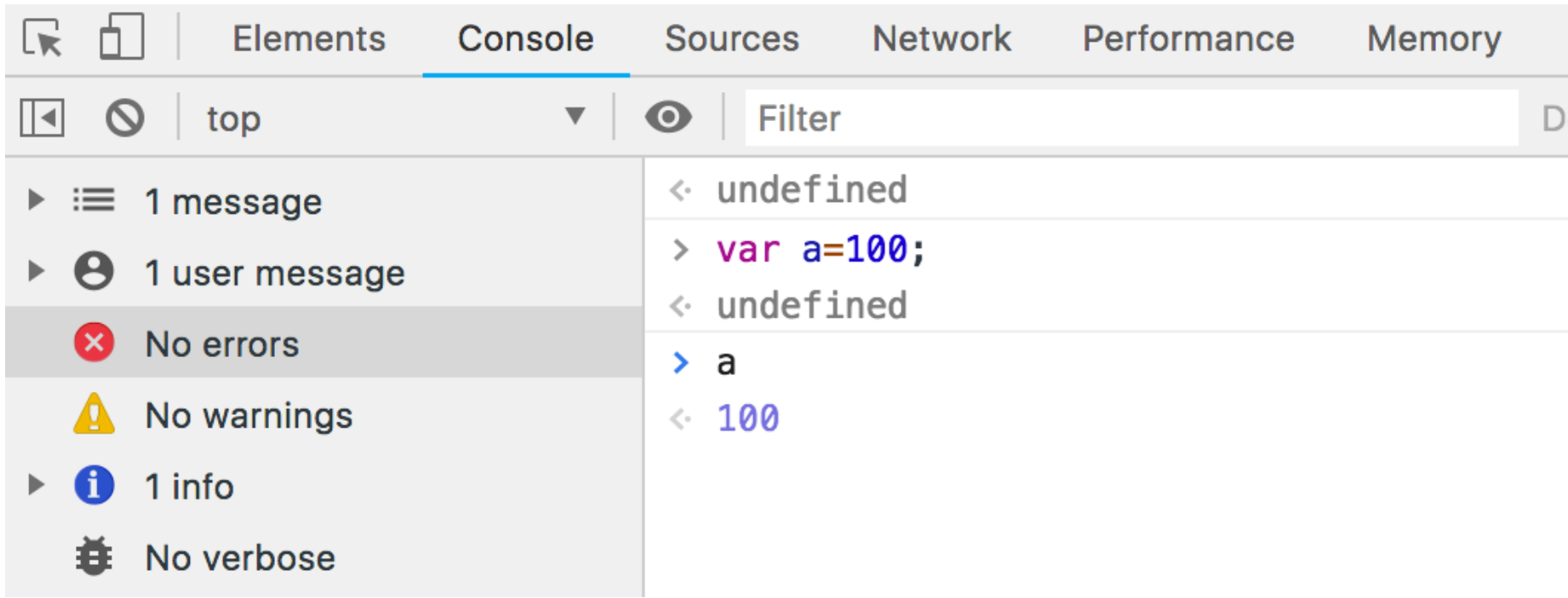
```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, World!</title>
    <script>
        function SayHello() {
            document.getElementById("myMessage").innerHTML = "Hello, World!";
            console.log("按钮被点击了");
        }
    </script>
</head>
<body>
    <h1>第一个 JavaScript 程序</h1>
    <p id="myMessage">试试点击下方的按钮...</p>
    <button type="button" onClick="SayHello()">点我</button>
</body>
</html>
```

在什么地方显示？

程序运行效果


localhost:63342/web/04_JavScript实验/01_Javascript展示/index.html
第一个 JavaScript 程序
Hello, World!
点我
Elements
Console
Sources
Network
Performance
Memory
Application
top
Filter
Default levels
1 message
1 user message
No errors
No warnings
1 info
No verbose
按钮被点击了

直接在控制台执行脚本


Elements
Console
Sources
Network
Performance
Memory
top
Filter
1 message
1 user message
No errors
No warnings
1 info
No verbose
undefined
var a=100;
undefined
a
100

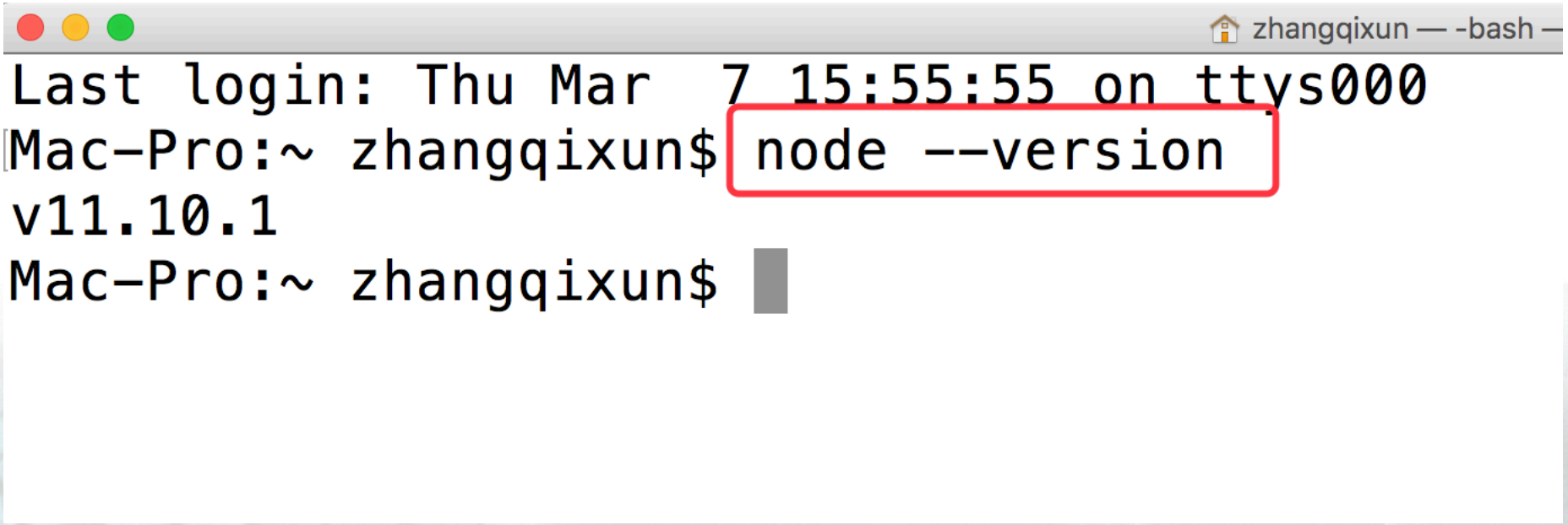
Node

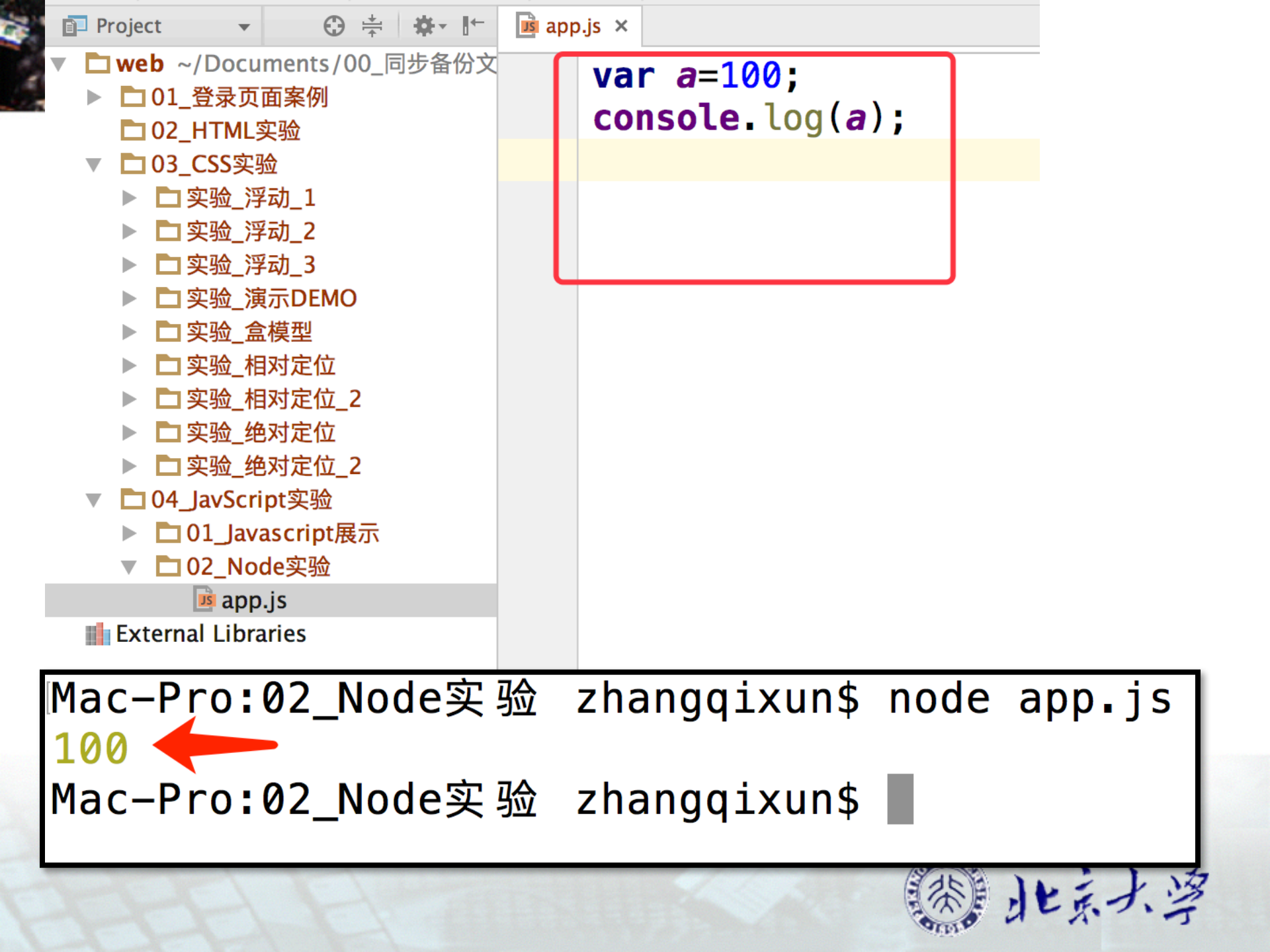
nodejs_百度百科

简介：Node 是一个让 JavaScript 运行在服务端的开发平台，它让 JavaScript 成为与PHP、Python、Perl、Ruby 等服务端语言平起平坐的脚本语言。发布于2009年5月，由Ryan Dahl开发，实质是对 Chrome V8引擎进行了封装。Node对一些特殊用例进行优化，提供替代的API，使得V8在非浏览器环境下运行得更好。V8引擎执...

发展史 特性 模块 安装 集成开发环境 应用方向 更多>>

https://baike.baidu.com/




Project
web ~/Documents/00_同步备份文
01_登录页面案例
02_HTML实验
03_CSS实验
实验_浮动_1
实验_浮动_2
实验_浮动_3
实验_演示DEMO
实验_盒模型
实验_相对定位
实验_相对定位_2
实验_绝对定位
实验_绝对定位_2
04_JavScript实验
01_Javascript展示
02_Node实验
app.js
External Libraries
app.js
var a=100;
console.log(a);
Mac-Pro:02_Node实验 zhangqixun$ node app.js
100
Mac-Pro:02_Node实验 zhangqixun$

JavaScript基本语法

定义变量

```
var  a;
a = 100;
或
var   a = 100;
```

```
var  a;
a = 100;
a = “abc”;
```

Javascript是一种松散类型的语言，在脚本中不需要指定一个变量的数据类型。解析器会自动推算出某个变量的正确数据类型。

标识符命名规则

- 区分大小写
- 可以包含字母、数字、下划线、美元符号、以及Unicode中的语言字符（包含中文），不能以数字开头。
- 变量名不能有空格
- 不能使用 **JavaScript** 中的关键字做变量名

从规范的角度来讲，应该是以字母开头，可以包含数字、下划线。

app.js

```
var 你好=200;
console.log(你好);
console.log(typeof 你好);
```

```
Mac-Pro:02_Node实验 zhangqixun$ node app.js
200
number
Mac-Pro:02_Node实验 zhangqixun$
```

“use strict” --严格模式

- "use strict"是JavaScript中一个非常好的特性，而且非常容易使用。

```
var a = 10;
b = 100;
console.log('a=',a);
console.log('b=',b);
```

```
"use strict"
var a = 10;
b = 100;
console.log('a=',a);
console.log('b=',b);
```

```
b = 100;
  ^

ReferenceError: b is not defined
```

程序示例

```
1  var a = 10;
2  var b;
3  b = 100;
4
5  console.log('a=',a);
6  console.log('b=',b);
7  console.log(typeof(a));
8
9  a = "abc";
10 console.log('a=',a);
11 console.log(typeof(a));
```

```
a= 10
b= 100
number
a= abc
string
```

基本数据类型

- 字符串 (String)
- 数值 (Number)
- 布尔 (Boolean)
- 空 (Null)
- 未定义 (Undefined)

String类型

- String类型用于表示由零或多个16位Unicode字符组成的字符序列，即字符串。字符串可以由单引号(')或双引号(")表示。

```
var str1 = "Hello1";
var str2 = 'Hello2';
```

- 任何字符串的长度都可以通过访问其length属性取得

```
(str1.length);        //输出6
```

程序示例

```
1  "use strict"
2
3  var str1 = "Hello1";
4  var str2 = 'Hello2';
5  console.log(str1);
6  console.log(str1.length);
7
8  console.log(str2);
9  console.log(str2.length);
```

```
Hello1
6
Hello2
6
```

值转换为字符串

- 要把一个值转换为一个字符串有两种方式。
  - 使用几乎每个值都有的**toString()方法**。数值、布尔值、对象和字符串值都有**toString()**方法。但**null和undefined值没有这个方法**。
  - 在不知道要转换的值是不是**null**或**undefined**的情况下，还可以使用转型**函数String()**，这个函数能够将任何类型的值转换为字符串。

程序示例

```
"use strict"
var value1 = 10;
var value2 = true;
var value3 = null;
var value4;

console.log(value1.toString());
console.log(value2.toString());
//console.log(value3.toString());
//console.log(value4.toString());

console.log(String(value1));
console.log(String(value2));
console.log(String(value3));
console.log(String(value4));
```

```
10
true
10
true
null
undefined
```

Number类型

- 用来表示整数和浮点数值
- 还有一种特殊的数值，即NaN（非数值 Not a Number）
- 有3个函数可以把非数值转换为数值：Number()、parseInt()和parseFloat()

程序示例

```
1  "use strict"
2  var value1 = 10;
3  var value2 = 10.11;
4  var value3;
5  console.log(value1);
6  console.log(value2);
7
8  value3 = Number("000011");
9  console.log(value3);
10
11 value3 = Number("Hello World");
12 console.log(value3);
13 console.log(isNaN(value3));
14
15 var value4 = parseFloat("22.5");
16 console.log(value4);
17
18 var value5 = parseInt("070");
19 console.log(value5);
```

```
10
10.11
11
NaN
true
22.5
70
```

Boolean类型

- 该类型只有两个字面值：true和false。
- 要将一个值转换为其对应的Boolean值， 可以调用类型转换函数Boolean()， 例如：

```
var message = 'Hello World';
var messageAsBoolean = Boolean(message);
```

数据类型	转换为true的值	转换为false的值
Boolean	true	false
String	任何非空字符串	“”（空字符串）
Number	任何非零数字值（包括无穷大）	0和NaN
Object	任何对象	null
Undefined	n/a（不适用）	undefined

程序示例

```
1  "use strict"
2  var message = 'Hello World';
3  var num = 100;
4  var messageAsBoolean = Boolean(message);
5  var numAsBoolean = Boolean(num);
6
7  console.log(messageAsBoolean);
8  console.log(numAsBoolean);
9
10 num = 0;
11 numAsBoolean = Boolean(num);
12 console.log(numAsBoolean);
```

```
true
true
false
```

Undefined类型

- **Undefined类型只有一个值，即特殊的undefined。在使用var声明变量但未对其加以初始化时，这个变量的值就是undefined**

```
1  "use strict"
2  var value1 ;
3  console.log(value1);
```

```
undefined
```

Null类型

- Null类型也是只有一个值的数据类型，这个特殊的值是null。

```
1 "use strict"
2 var value1 ;
3 var value2 = null;
4 console.log(value1);
5 console.log(value2);
6 console.log(typeof(value1));
7 console.log(typeof(value2));
```

```
undefined
null
undefined
object
```

程序示例

```
1  "use strict"
2  var value1 ;
3  var value2 = null;
4  console.log(value1);
5  console.log(value2);
6  console.log(typeof(value1));
7  console.log(typeof(value2));
8
9  console.log(value1 == value2);
10 console.log(value1 === value2);
```

```
undefined
null
undefined
object
true
false
```

北京大学

typeof操作符

- 对一个值使用typeof操作符可能返回下列某个字符串：
  - "undefined"——如果这个值未定义；
  - "boolean"——如果这个值是布尔值；
  - "string"——如果这个值是字符串；
  - "number"——如果这个值是数值；
  - "object"——如果这个值是对象或null；
  - "function"——如果这个值是函数；

算术运算符

- JavaScript中的算术运算符有单目运算符和双目运算符。
- 双目运算符
  +（加）、-（减）、*（乘）、/（除）、%（取模）、|（按位或）、&(按位与)、<<（左移）、>>（右移）、>>>（右移，零填充）。

- 单目运算符
  ++（递加1）、--（递减1）。

比较运算符

- 比较运算符它的基本操作过程是，首先对它的操作数进行比较，再返回一个true或False值，有８个比较运算符：

  <（小于）

  >（大于）

  <=（小于等于）

  >=（大于等于）

  ==（等于，只是值相等）

  ===（等于，值与类型均相等）

  !=（不等于，只是值）

  !==（不等于，值与类型）

逻辑运算符

- 在JavaScript中可使用以下逻辑运算符:

  ！（取反)

  &&（与）

  ||（或）

if语句

```
if (条件 1)
{
   当条件 1 为 true 时执行的代码;
}else if (条件 2)
{
   当条件 2 为 true 时执行的代码;
}else
{
   当条件 1 和 条件 2 都不为 true 时执行的代码;
}
```

switch语句

```
switch(k)
{
   case k1:
      执行代码块 1 ;
      break;

   case k2:
      执行代码块 2 ;
      break;

   default:
      默认执行（k 值没有在 case 中找到匹配时）;
}
```

while语句

```
while (条件)
{
   需要执行的代码;
}
```

```
do
{
   需要执行的代码;
}
while (条件);
```

for语句

```
for(变量 = 初始值 ; 循环条件 ; 变量累加方法)
{
   循环语句;
}
```

for/in语句

```
var person={fname:"John",lname:"Doe",age:25};

for (x in person)
{
  txt=txt + person[x];
}
```

for/in 语句循环遍历对象的属性

break、continue

- **Break 语句**

  break 语句可用于跳出循环。

  break 语句跳出循环后，会继续执行该循环之后的代码

- **Continue 语句**

  continue 语句中断循环中的迭代，然后继续循环中的下一个迭代。

函数

函数由关键词 **function** 定义， 函数可以有多个参数。基本格式为：

```
function 函数名 (参数1， 参数2)
{
   函数体;

   return 返回值;
}
```

程序示例

temp.js

```
1  "use strict"
2
3  function add(name, email, age){
4      var student = [];
5      student[0] = name;
6      student[1] = email;
7      student[2] = age;
8      return student;
9  }
10
11 var name = "zhang";
12 var email = "zhang@ss.pku.edu.cn";
13 var age = 30;
14
15 var student = add(name , email , age);
16
17 console.log(student);
18
```

```
/usr/local/bin/node temp.js
[ 'zhang', 'zhang@ss.pku.edu.cn', 30 ]
```

匿名函数

03_匿名函数/app.js

```
"use strict"
var func=function(webName){
    console.log(webName);
};
func("你好");
```

```
Mac-Pro:03_匿名函数 zhangqixun$ node app.js
你好
Mac-Pro:03_匿名函数 zhangqixun$
```

注意“=”右边的函数就是一个匿名函数，创造完毕函数后，又将该函数赋给了变量func

03_匿名函数/app.js

```
"use strict"
var func=function(webName){
    console.log(webName);
};
func("你好");

(function(x, y){
    console.log(x + y);
})(2, 3);
```

```
Mac-Pro:03_匿名函数 zhangqixun$ node app.js
你好
5
Mac-Pro:03_匿名函数 zhangqixun$
```

回调函数

函数作为函数参数进行传递。

04_回调函数/app.js

```
"use strict"

function output(name,email,age) {
    console.log(name+" | "+email+" | "+age);
}

function read(callback, name , email , age) {
    console.log("read function");
    callback(name , email , age);
}

read(output, "zhang" ,"zhang@ss.pku.edu.cn", 40);
```

```
Mac-Pro:04_回调函数 zhangqixun$ node app.js
read function
zhang | zhang@ss.pku.edu.cn | 40
Mac-Pro:04_回调函数 zhangqixun$
```

变量作用域

- 全局版本的变量与局部版本的变量

```
1  "use strict"
2  var num = 10;
3  function funA(){
4      var num = 20;
5      console.log(num);
6      funB();
7  }
8
9  function funB(){
10     console.log(num);
11 }
12 funA();
```

```
20
10
```

闭包

- 闭包（closure）是Javascript语言的一个难点，也是它的特色，很多高级应用都要依靠闭包实现。

- 闭包：是指有权访问另外一个函数作用域中的变量的函数。

- 创建闭包的常见方式就是在一个函数内部创建另外一个函数。

函数嵌套定义

05_函数嵌套定义/app.js

```
"use strict"

function Student(n,e,a) {
    var s_name, s_email, s_age;
    function set(name, email, age) {
        s_name = name;
        s_email = email;
        s_age = age;
    }
    set(n,e,a);
    return [s_name,s_email,s_age];
}

var student = Student("zhang" ,"zhang@ss.pku.edu.cn", 40);

console.log(student);
```

```
Mac-Pro:05_函数嵌套定义 zhangqixun$ node app.js
[ 'zhang', 'zhang@ss.pku.edu.cn', 40 ]
Mac-Pro:05_函数嵌套定义 zhangqixun$
```

程序示例

05_函数嵌套定义/app.js

```
"use strict"

function Student(n,e,a) {
    var s_name, s_email, s_age;
    function set(name, email, age) {
        s_name = name;
        s_email = email;
        s_age = age;
    }
    set(n,e,a);
    console.log(name);
    return [s_name,s_email,s_age];
}

var student = Student("zhang" ,"zhang@ss.pku.edu.cn", 40);
console.log(student);
```

错误，无法访问函数内部的数据

- set函数可以访问上一级函数的变量
- Student函数不能访问set函数的变量

- 出于种种原因，有时候需要得到函数内的局部变量。
- 但是，正常情况下，这是办不到的，只有通过变通方法才能实现。
- 在函数的内部，再定义一个函数，然后将这个返回。

程序示例

06_访问函数内部变量/app.js

```
"use strict"
function add() {
    var counter = 0;
    counter += 1;
    console.log(counter);
    return counter;
}

add();
add();
add();
```

怎么才能让计数器数值不断增加？

```
Mac-Pro:06_访问函数内部变量 zhangqixun$ node app.js
1
1
1
Mac-Pro:06_访问函数内部变量 zhangqixun$
```

06_访问函数内部变量/app.js × app2.js ×

```
"use strict"

var counter = 0;
function add() {

    counter += 1;
    console.log(counter);
    return counter;
}

add();
add();
add();
```

```
Mac-Pro:06_访问函数内部变量 zhangqixun$ node app2.js
1
2
3
Mac-Pro:06_访问函数内部变量 zhangqixun$
```

06_访问函数内部变量/app.js × app2.js × app3.js ×

```
"use strict"


function add() {
    var counter = 0;
    // counter += 1;
    function plus() {counter += 1;}
    plus();

    console.log(counter);
    return counter;
}

add();
add();
add();
```

```
Mac-Pro:06_访问函数内部变量 zhangqixun$ node app3.js
1
1
1
Mac-Pro:06_访问函数内部变量 zhangqixun$
```

06_访问函数内部变量/app.js × | app2.js × | app3.js × | app4.js × | 07_闭包/app.js ×

```
"use strict"

function add() {
    var counter = 0;
    return function () {
        counter += 1;
        console.log(counter);
        return counter;
    }
};

var add_counter = add();

add_counter();
add_counter();
add_counter();
```

```
Mac-Pro:07_闭包 zhangqixun$ node app.js
1
2
3
Mac-Pro:07_闭包 zhangqixun$
```

闭包可以用在许多地方。它的最大用处有两个

- 可以读取函数内部的变量
- 函数内部变量的值始终保持在内存中

程序示例

temp.js

```
1  "use strict"
2
3  function Student(n, e, a){
4      var s_name, s_email, s_age;
5      var a=0;
6      s_name = n;
7      s_email = e;
8      s_age = a;
9      function get_n(){
10         a++;
11         console.log("a=",a);
12         return s_name;
13     }
14     return get_n;
15 }
16
17 var get_name = Student("zhang", "zhang@ss.pku.edu.cn", 30);
18 var name = get_name();
19 get_name();
20 console.log(name);
```

程序中调用了两次get_name，而局部变量a在第一次调用后，并没有释放。

```
/usr/local/bin/node temp.js
a= 1
a= 2
zhang
```

Q&A