

# JavaScript 进阶

**主讲人： 和凌志**

# Node.js 概述

Node.js 的核心技术在于，它让JavaScript 从浏览器中分离出来，从而让 JavaScript得以在服务器上运行。

# Node.js 安装

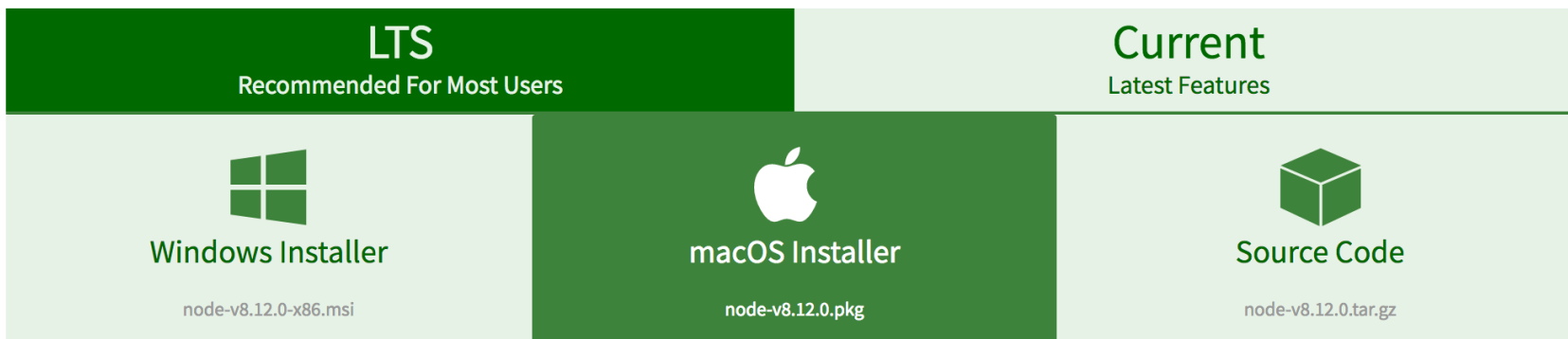
<https://nodejs.org/en/download/>



## Downloads

Latest LTS Version: **8.12.0** (includes npm 6.4.1)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.



# 如何查看所安装的 node.js 的版本？

在终端窗口，执行：

```
$ node -v
```

## node.js 的验证

在终端窗口，执行： `$ node`

在 node.js 提示符下，执行：

```
> console.log('Hello World')
```

## 在 node.js 环境，执行 JavaScript 代码

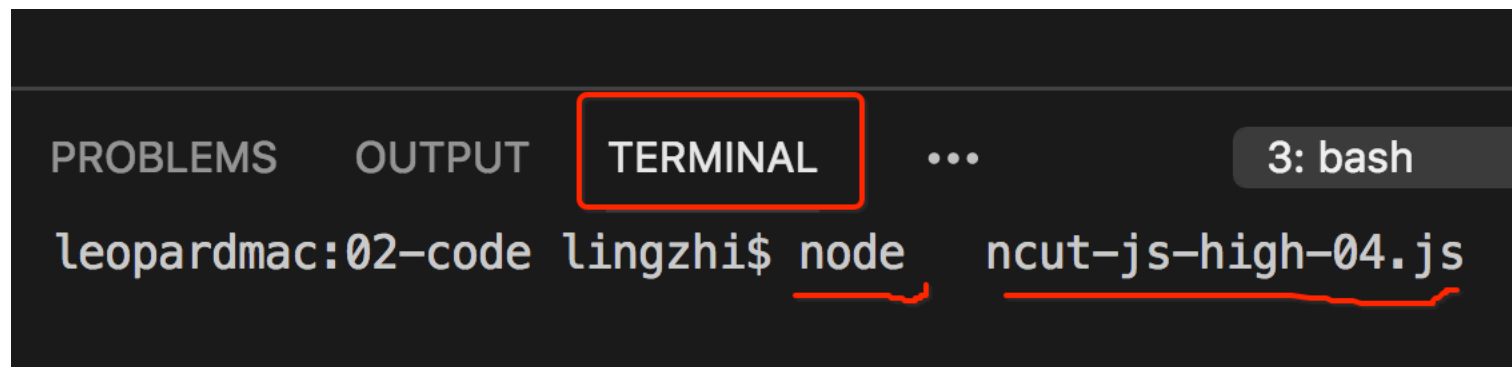
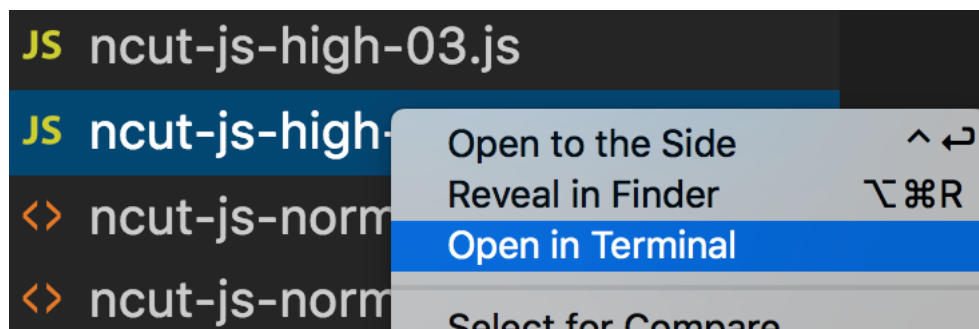
- 在终端窗口，执行：

```
$ node    hello.js
```

- 注意：

- 创建的文件类型必须以 js 为后缀
- js 文件所在的路径
- 也可以在 VS code 直接运行 js 文件

在 VS code 中，鼠标右键，选中要运行的 js 文件，打开它的 终端窗口，





# JavaScript 进阶实例



# JavaScript 中的 == 与 === 的区别

```
var a = "hello";

var b = new String("hello");

console.log(typeof a);

console.log(typeof b);

if (a == b) {
    console.log("两个等号情况下, a等于b");
}

if (a === b) {
    console.log("三个等号情况下, a等于b");
}
```

考虑到此差异，建议通过直观的方式进行定义，避免使用 new

在 JavaScript 中，函数也是对象，函数可以作为参数来传递

```
var a = function() {  
  
};  
  
console.log(a);
```

输出结果: [Function: a]

函数可以存储在变量中，随后可以像其他对象一样，进行传递

## 函数的参数数量

```
var myFunction = function (a,b,c) {  
  
};  
  
console.log (myFunction.length) ;
```

JavaScript 函数有一个很有意思的属性——参数数量，该函数指明函数声明时可接收的参数数量。在JavaScript中，该属性名为 `length`。Node.js 框架就是通过判断不同参数的个数提供不同的功能。

# 函数声明、函数表达式、匿名函数

- **函数声明** : `function fnName () {...};`

使用`function`关键字声明一个函数，再指定一个函数名，叫函数声明。

- **函数表达式** `var fnName = function () {...};`

使用`function`关键字声明一个函数，但未给函数命名，最后将匿名函数赋予一个变量，叫函数表达式，这是最常见的函数表达式语法形式。

- **匿名函数** : `function () {};`

使用`function`关键字声明一个函数，但未给函数命名，所以叫匿名函数，匿名函数属于函数表达式，匿名函数有很多作用，赋予一个变量则创建函数，赋予一个事件则成为事件处理程序或创建 **闭包 ( block )** 等等。

# 自执行函数（立即执行函数）

## Immediately Invoked Function Expression (IIFE)

```
var a = 3;

(function () {
    var a = 5;

    console.log(a);
})();

console.log(a);
```

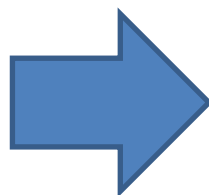
自执行函数是一种机制，通过这种机制声明和调用一个匿名函数，能够达到仅定义一个新作用域的作用。

自执行函数对声明私有变量是很有用的，这样可以让私有变量不被其他代码访问。

自执行函数（立即执行函数） 换一种写法：

Immediately Invoked Function Expression (IIFE)

```
(function() {  
    var a = "block";  
})();
```



```
function something() {  
    var a = "block";  
}  
something();
```

如果不用自执行函数，怎么处理？

```
var a = 3;

function test() {
    var a = 5;
    console.log(a);
};

test();

console.log(a);
```

先声明一个函数，再调用该函数

## 闭包： 函数嵌套函数

- 闭包，简单说来就是函数嵌套函数，或者说定义在一个函数内部的函数，它是将函数内部和函数外部连接起来的一座桥梁。
- 闭包可以用在很多地方，它的最大用处有两个，一个是可以读取函数内部的变量，另一个是让这些变量的值始终保持在内存中。



# JavaScript语言特有的“链式作用域”

```
function t1() {  
    var a = 100;  
    console.log("第一个 a 值是: ", a);  
  
    function t2() {  
        var a = 200;  
        console.log("第二个 a 值是: ", a);  
    }  
    return t2();  
}  
  
t1();
```

- 在函数 t1 内部声明的变量 a 本来是一个局部变量，为什么在调用 t2 函数能打印出 a 变量的值呢？
- 原因：函数 t2 被包括在函数 t1 内部，这时 t1 内部的所有局部变量，对 t2 都是可见的。但是反之不然，t2 内部的局部变量，对 t1 是不可见的。这就是 Javascript 语言特有的“链式作用域”结构（chain scope）。
- 子对象会一级一级地向上寻找所有父对象的变量。所以，父对象的所有变量，对子对象都是可见的，反之则不成立。



# JavaScript 常见面试题

ES5 JavaScript , only have two scopes

*Block level scope*  
*global scope*

*function level scope*  
*function scope*

```
{  
  var a = "block";  
}  
  
console.log(a);
```

以上代码，可以正常运行吗？



Block scope: 块范围，是指只在它的 { } 内起作用

JavaScript 的 *no block level scope* 一开始就成为  
JavaScript 工程师的“灾难”



The variable **a**, as we've declared it above, exists in *global scope*

so this means it's visible from *everywhere* in our application.

# 如果把变量声明在函数内部，将会怎样？

In ES5 apart from global scope, the only other scope is *function scope*

```
function hello() {  
    var a = "function";  
}  
hello();  
console.log(a);
```

报错: ReferenceError: a is not defined

报错原因:

This is because the **a** variable is declared *inside a function* and is therefore only visible inside *that* function

trying to access it *outside the function* results in an error.

如果在 function 内部添加一个 for 循环，将会怎样？

```
function hello() {  
  var a = "function";  
  for (var i = 0; i < 10; i++) {  
    var a = "block";  
  }  
  console.log(a);  
}  
hello();
```

输出结果: block



理解变量的生命周期， global scope 和 function scope

What gets printed out here is *block* not *function* despite the fact we are outside the for loop, that's because the **body** of the **for** loop **is not its own scope**.

本质上讲， for 循环体内， 无法形成自己的scope

通过立即执行函数，改下代码如下

```
function hello() {  
    var a = "function";  
    for (var i=0; i<5; i++) {  
        (function() {  
            var a = "block";  
        })();  
    }  
    console.log(a);  
}  
hello();
```

It's a function that we **call immediately** after defining it.

输出结果: function

立即执行函数里面的变量，在它的函数之外是不可见的！

- Since functions have their own scope, using an **IIFE** has the same effect as if we had block level scope
- the variable `a` inside the IIFE isn't visible *outside* the IIFE.



# 如何解决 ES5 (Javascript) var 之痛?

ES6 has the new **let** keyword

we use it in place of the **var** keyword and it creates  
**a variable with block level scope**

用 let 代替 var ， 将会怎样？

```
function hello() {  
  var a = "function";  
  for (var i = 0; i < 5; i++) {  
    let a = "block";  
  }  
  console.log(a);  
}  
hello();
```

a declared in the **for**  
loop body only exists  
between the { and },

输出结果： **function**

以下代码，将输出怎样的结果？

```
var funcs = [ ];  
for (var i = 0; i < 5; i += 1) {  
    var y = i;  
    funcs.push(function () {  
        console.log("y=",y);  
    })  
}
```

```
funcs.forEach(function (func) {  
    func()  
});
```

```
console.log("i=" , i);
```

```
y= 4  
y= 4  
y= 4  
y= 4  
i= 5
```

for 循环中，用 let 代替 var，将输出怎样的结果？

```
var funcs = [ ];  
for (var i = 0; i < 5; i += 1) {  
    let y = i;  
    funcs.push(function () {  
        console.log("y=", y);  
    })  
}  
  
funcs.forEach(function (func) {  
    func()  
});  
  
console.log("i=" , i);
```

```
y= 0  
y= 1  
y= 2  
y= 3  
y= 4  
i= 5
```

for 循环中，用 let 代替 var，将输出怎样的结果？

```
var funcs = [ ];  
for (var i = 0; i < 5; i += 1) {  
    let y = i;  
    funcs.push(function () {  
        console.log("y=", y);  
    })  
}
```

```
funcs.forEach(function (func) {  
    func()  
});
```

```
console.log("i=" , i);
```

```
y= 0  
y= 1  
y= 2  
y= 3  
y= 4  
i= 5
```

注意：

**var** 与 **let** 定义变量，其作用域不同



ES6 的 let 拥有属于自己的 scope , 从此不再 “迷茫” !

```
var funcs = [ ];
for (let i = 0; i < 5; i += 1) {
  let y = i;
  funcs.push(function () {
    console.log("y=", y);
  })
}


funcs.forEach(function (func) {
  func()
});

console.log("i=" , i);
```

ReferenceError: i is not  
defined

# 小结

- `let` is a very important addition the javascript language in ES6.
- It's `not a replacement` to `var`, `var` can still be used even in ES6 and has the same semantics as ES5.
- However unless you have a specific reason to use `var` , I would expect all variables you define from now on to use `let`.



TypeScript 应运而生，  
并受到前端开发者的热捧！



Q & A