JavaScript高级编程

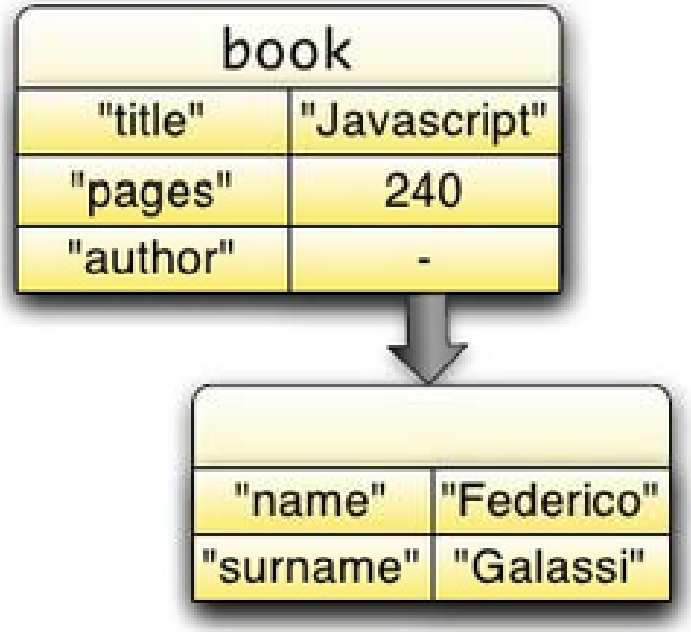
窦连军 @八月虎baidu
北京乐美无限科技有限公司

高级编程内容

1. 原型继承
2. 函数闭包与作用域
3. 异常

对象Object

- Containers of key/value pairs （键值对）
  - keys are strings
  - values are anything




```
// Creation with literal
var book = {
    title: "Javascript",
    pages: 240,
    author: {
        name: "Federico",
        surname: "Galassi"
    }
}
```

对象的动态性

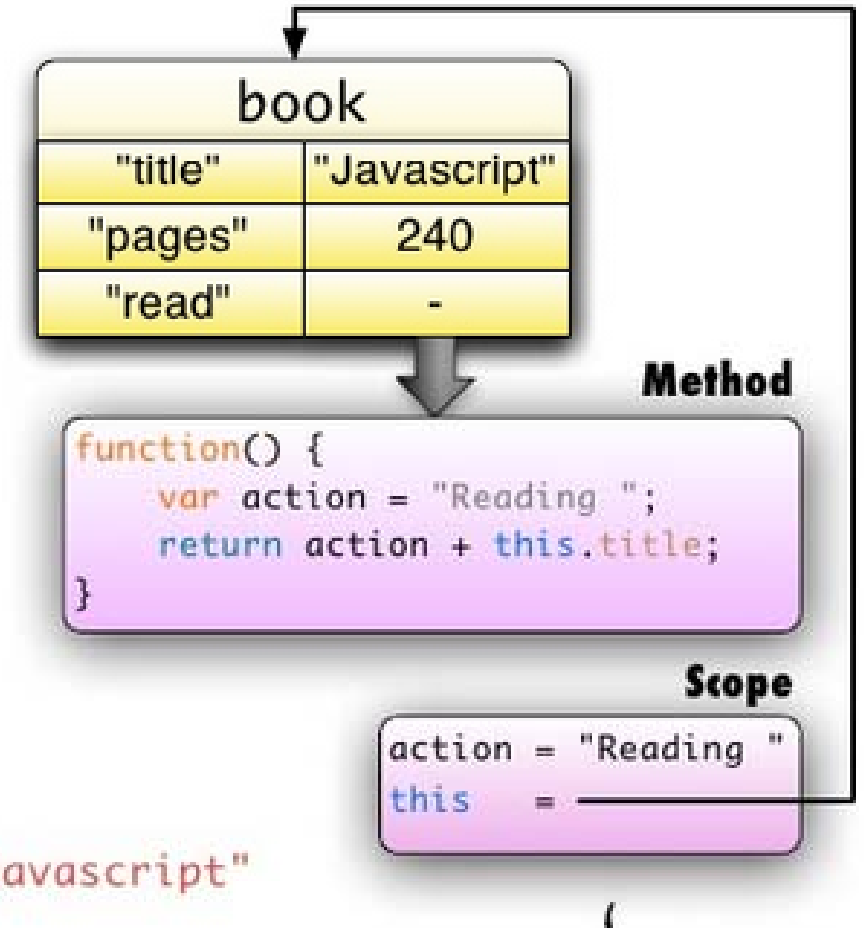
- Objects are dynamic
  - Properties can be added and removed at runtime
  - No class constraints

```
// Get a property
book["title"]              // returns "Javascript"
book.title                 // same as book["title"]
book.propertyNotThere      // returns undefined

// Set or update a property
book.cover = "butterfly.jpg"
book.title = "Javascript the good parts"

// Delete a property
delete book.title          // now book.title is undefined
```

对象的方法

- Methods are function valued properties
- Inside methods **this** is bound to object “on the left”

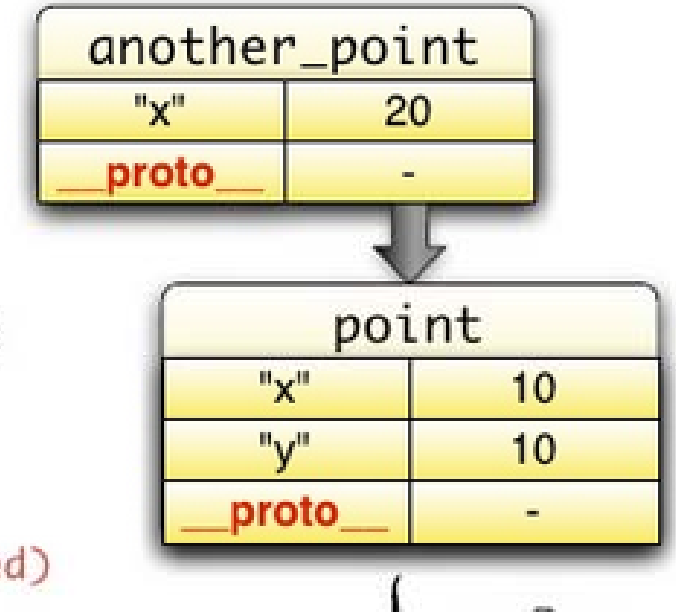
```
book.read = function() {
    var action = "Reading ";
    return action + this.title;
}

book.read(); // returns "Reading Javascript"
```



对象属性的继承机制

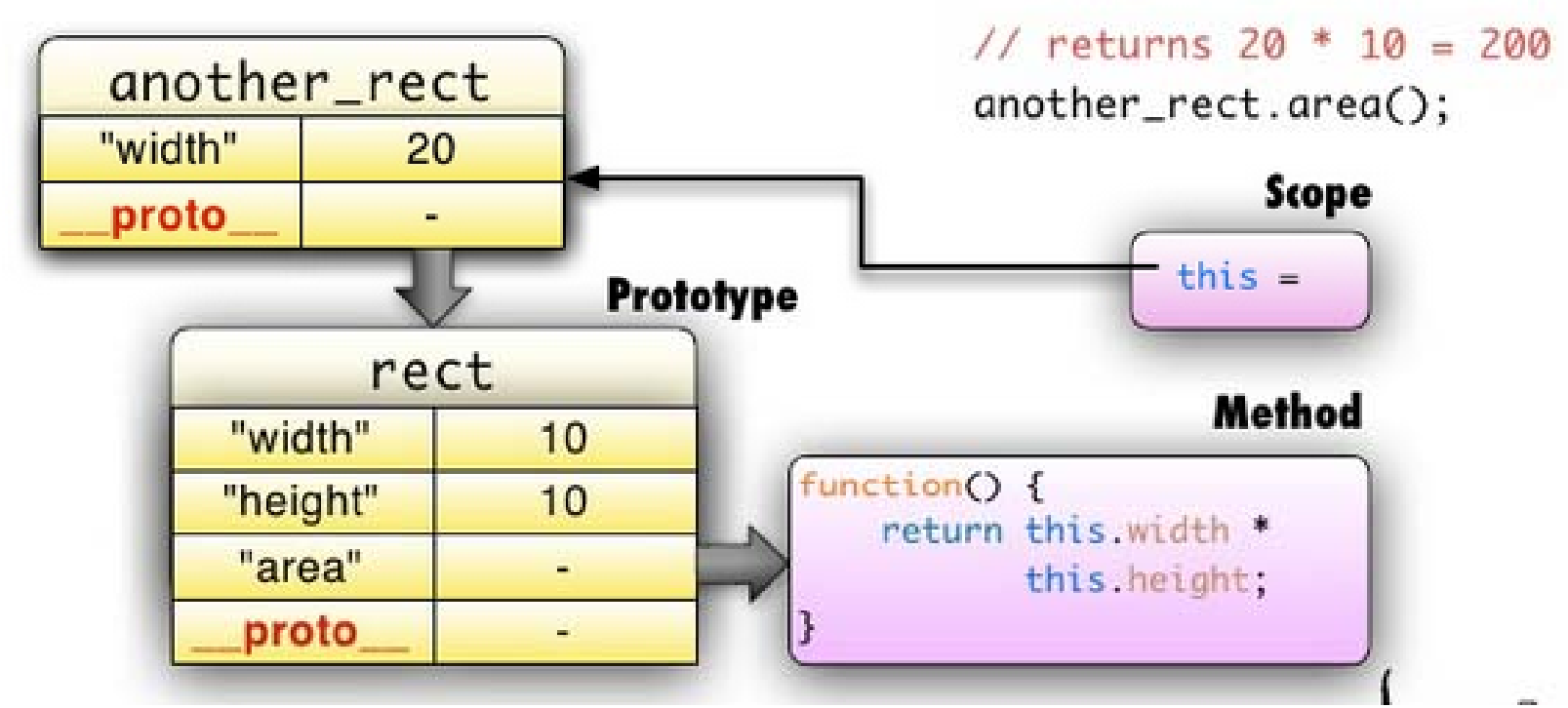
- Every object can be linked to another object through the special “prototype” property
- If a property does not exist in the object, request is delegated to its prototype

```
var point = {
    x: 10,
    y: 10
};
var another_point = Object.create(point);
another_point.x = 20;

another_point.x;  // returns 20
another_point.y;  // returns 10 (delegated)
```



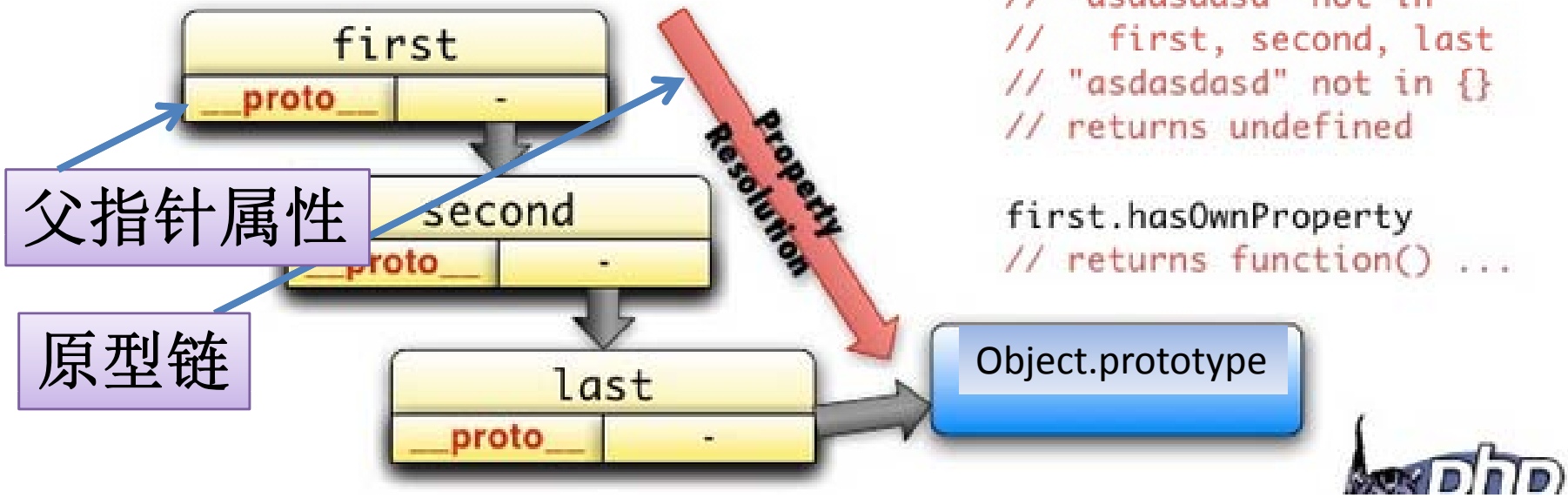
对象属性的继承机制

- Delegation works for methods too
- **this** is always bound to the “first object”



对象属性的继承机制

- Properties are resolved by following the Prototype Chain
- Prototype Chains always ends with Object
- Beyond there's undefined




```
first.asdasdasd;
// "asdasdasd" not in
//   first, second, last
// "asdasdasd" not in {}
// returns undefined

first.hasOwnProperty
// returns function() ...
```

原型继承

- Prototypes are javascript way to share
  - Data
  - Behavior

原型继承

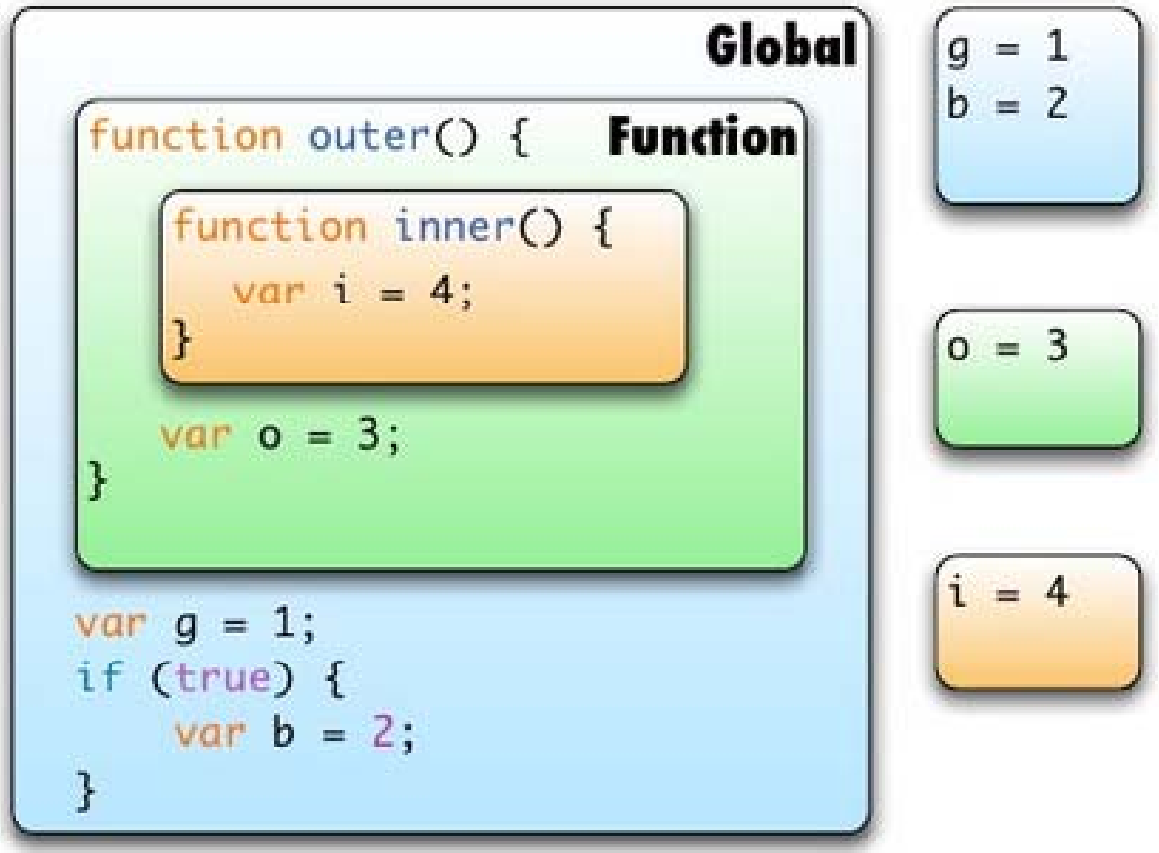
- Prototypal Inheritance
  - Vs Classical Inheritance
  - Simpler
    - No classes and objects, only objects
  - Easier
    - Work by examples, not abstractions
  - Powerful !!
    - Can simulate classical
    - Reverse not true
  - Shhhh, Don't tell anyone
    - Easier to write spaghetti code

原型继承

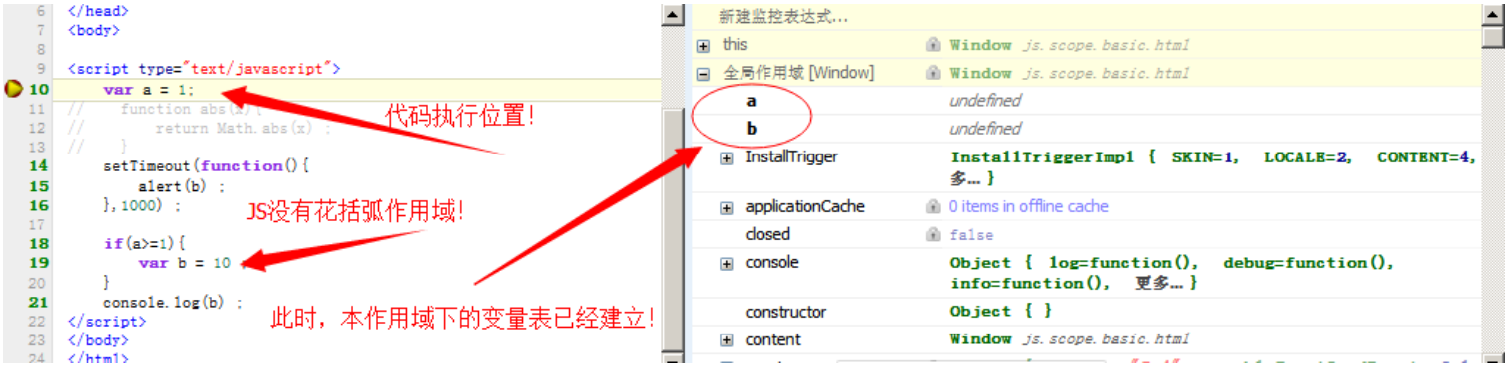
- Ok, I cheated
  - __proto__ available in mozilla only
  - Object.create coming in next revision of language
- Javascript is schizophrenic
  - Prototypal nature
  - Wannabe classical

Scope（作用域）

- Scopes
  - Global
  - Function
- No block-level

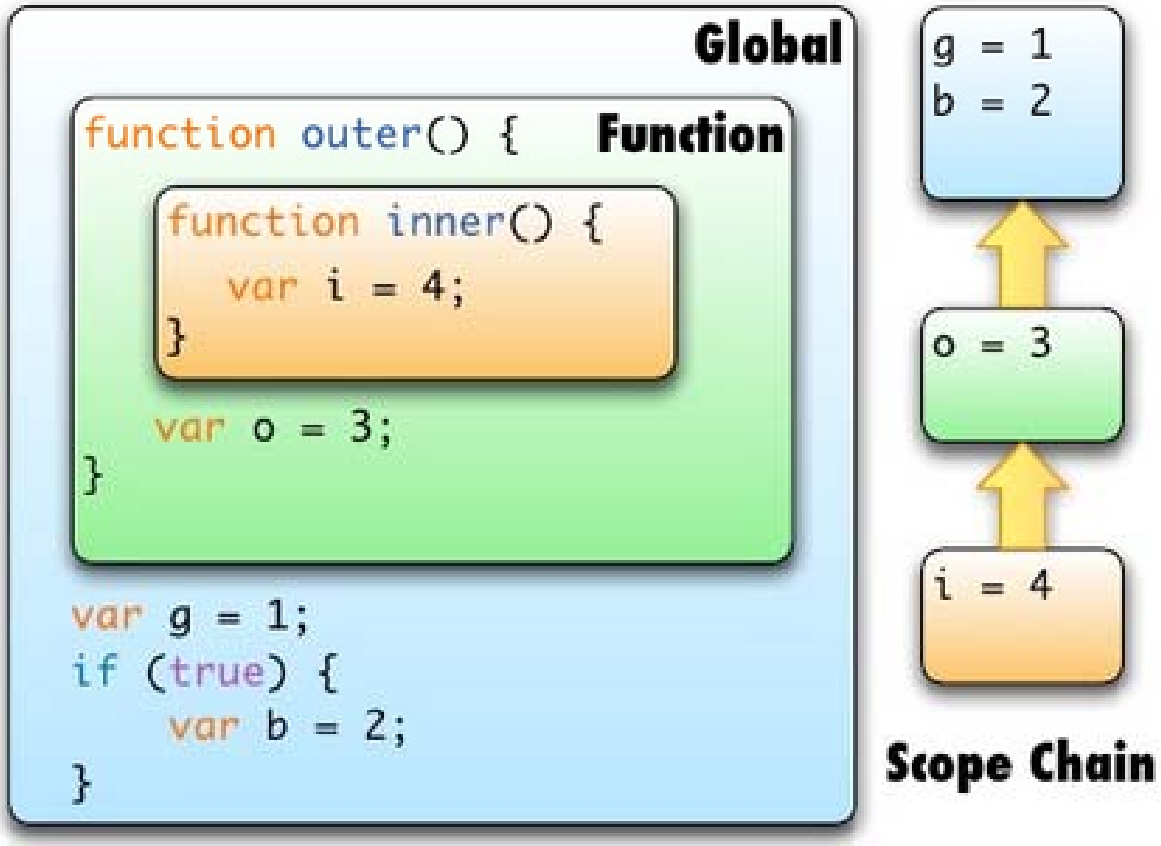


实例演示


6 </head>
7 <body>
8
9 <script type="text/javascript">
10 var a = 1;
11 // function abs(x) {
12 // return Math.abs(x);
13 // }
14 setTimeout(function(){
15 alert(b);
16 },1000);
17
18 if(a>=1){
19 var b = 10;
20 }
21 console.log(b);
22 </script>
23 </body>
24 </html>
代码执行位置!
JS没有花括弧作用域!
此时，本作用域下的变量表已经建立!
新建监控表达式...
this Window js.scope.basic.html
全局作用域 [Window] Window js.scope.basic.html
a undefined
b undefined
InstallTrigger InstallTriggerImpl { SKIN=1, LOCALE=2, CONTENT=4, 多... }
applicationCache 0 items in offline cache
closed false
console Object { log=function(), debug=function(), info=function(), 更多... }
constructor Object { }
content Window js.scope.basic.html

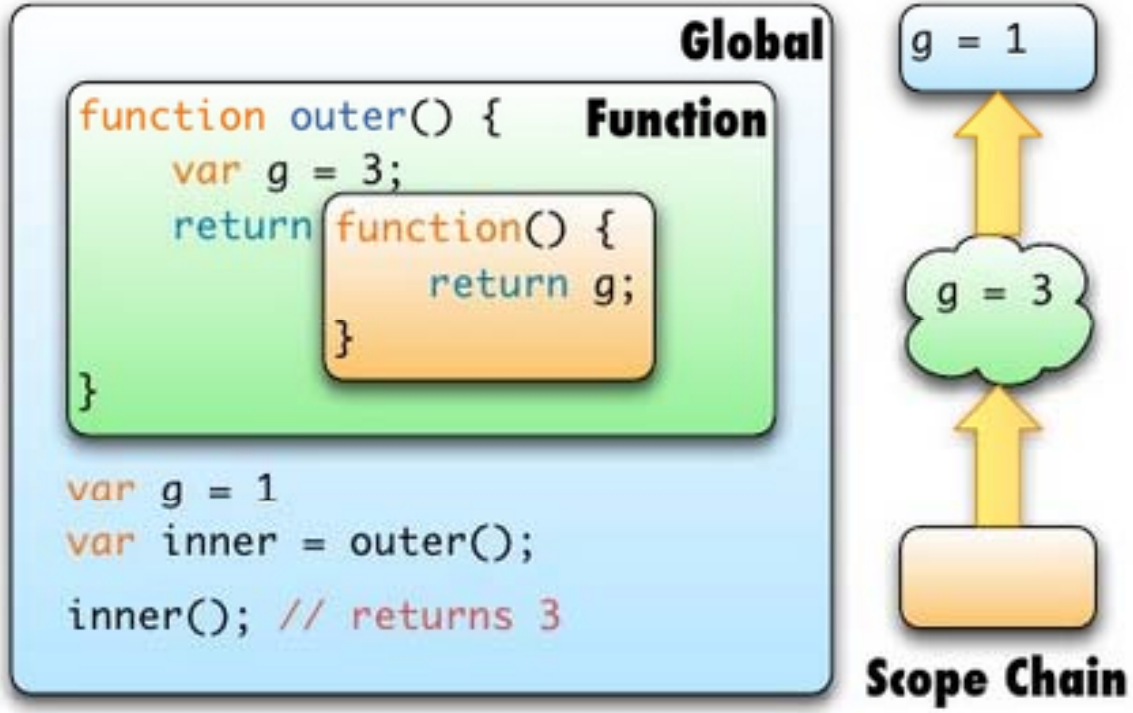
Scope Chain（作用域链）

- Scope Chain
- Each scope "inherits" from the "previous" one



闭包Closure

- Closure:
  Functions "remember" their scope chain



闭包例子1

- 利用闭包实现Timer定时器：

```
var count = 0;

var timer = setInterval(function(){
  if ( count < 5 ) {
    log( "Timer call: ", count );
    count++;
  } else {
    assert( count == 5, "Count came via a closure, accessed each step." );
    assert( timer, "The timer reference is also via a closure." );
    clearInterval( timer );
  }
}, 100);
```

闭包例子2

- 利用闭包实现DOM事件监听器：

```
var count = 1;
var elem = document.createElement("li");
elem.innerHTML = "Click me!";
elem.onclick = function(){
  log( "Click #", count++ );
};
document.getElementById("results").appendChild( elem );
assert( elem.parentNode, "Clickable element appended." );
```

闭包例子3

- 利用闭包实现类的私有属性：

```
function Ninja(){
  var slices = 0;

  this.getSlices = function(){
    return slices;
  };
  this.slice = function(){
    slices++;
  };
}

var ninja = new Ninja();
ninja.slice();
```

对象构建

- Constructor Functions
  - Function is a class constructor
  - Function prototype is class behavior
  - new makes new objects
- Why?
  - feels classical
  - feels familiar

```
function Rectangle(w, h) {
    this.w = w;
    this.h = h;
}
Rectangle.prototype.higher =
function() { this.h += 1 };

var rect = new Rectangle(5,10);
```

模块技术

- 按模块封装代码块：

```
(function(){
  var myLib = window.myLib = function(){
    // Initialize
  };

  // ...
})();
```

```
var myLib = (function(){
  function myLib(){
    // Initialize
  }

  // ...

  return myLib;
})();
```

模块技术

```
(function(){
  var count = 0;

  var timer = setInterval(function(){
    if ( count < 5 ) {
      log( "Timer call: ", count );
      count++;
    } else {
      assert( count == 5, "Count came via a closure, accessed each step." );
      assert( timer, "The timer reference is also via a closure." );
      clearInterval( timer );
    }
  }, 100);
})();

assert( typeof count == "undefined", "count doesn't exist outside the wrapper" );
assert( typeof timer == "undefined", "neither does timer" );
```