4. 深入理解jQuery

- jQuery凭借简洁的语法和跨平台的兼容性，极大地简化了JavaScript开发人员遍历HTML文档、操作DOM、处理事件、执行动画和开发Ajax的操作。

- 其独特而又优雅的代码风格改变了JavaScript程序员的设计思路和编写程序的方式。

学习jQuery源码

- 学习先进的设计理念
- 学习行业公认最佳实践
- 学习各种实现技巧
- 巩固JavaScript基础
- 深入理解jQuery

jQuery与作用域

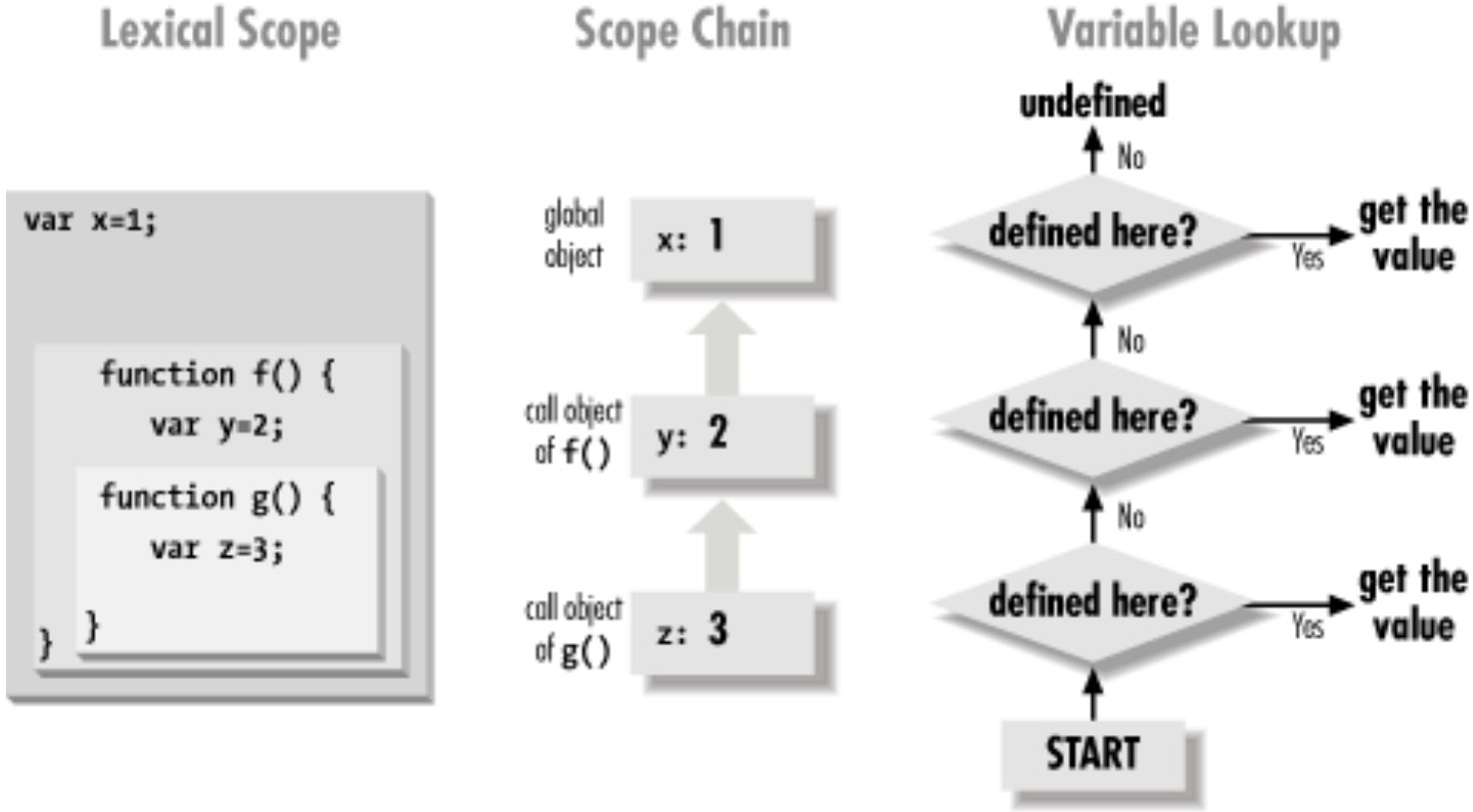
- 执行环境（execution context）定义了变量或者函数有权访问的其他数据。

- 每个执行环境中都有一个与之关联的变量对象（variable object），环境中定义的所有变量和函数都保持在这个对象中。——举例：window.undefined === undefined // true

- 全局执行环境是最外围的一个执行环境。在Web浏览器中，这个环境是window对象。

- 某个执行环境中得所有代码执行完毕后，该环境被销毁，保存在其中的所有变量和函数定义也随之销毁。

- 每个函数都有自己的执行环境。当执行流进入一个函数时，函数的环境就会被推入一个环境栈中。



全局变量

```
function add(num1, num2){
  sum = num1 + num2;
  return sum;
}

var result = add(10, 20);
// 30
alert(sum); // 30

// 不要这样写!
```

局部变量

```
function add(num1, num2){
  var sum = num1 + num2;
  return sum;
}

var result = add(10, 20);
// 30
alert(sum); // undefined
```

变量查询的性能

- 访问局部变量比访问全局变量更快，因为不用向上搜索作用域链。

jQuery选择器

- jQuery(selector [, context])
- jQuery(selector [, context])
- jQuery(element)
- jQuery(elementArray)
- jQuery(object)
- jQuery(selection)
- jQuery()

jQuery对象

- 基于HTML字符串创建，或者从文档中选择
- 是一个类数组对象，包含一系列DOM对象的集合，拥有大量的jQuery方法
- 也叫做“匹配元素”
- 使用jQuery()方法来创建（别名为$()）
- jQuery对象的很多方法返回jQuery对象，所以支持链式语法
- 如果使用“破坏性”的jQuery方法改变了jQuery对象，则可以使用.end()方法来返回被改变之前的jQuery对象
- 没有选择任何对象的jQuery对象也可以进行各种操作

.noConflict()

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
  // Code that uses jQuery's $ can follow here.
});
// Code that uses other library's $ can follow here.
```

jQuery与构造函数

创建对象

```
var person = new Object();

person.name = 'Nicholas';
person.age = 29;
person.job = 'front-end engineer';
person.sayName = function(){
 alert(this.name);
}
```

对象字面量

```
var person = {
 name : 'Nicholas',
 age : 29,
 job : 'front-end engineer',

 sayName : function(){
  alert(this.name);
 }
}
```

工厂模式

```
function createPerson(name, age, job){
  var o = new Object();
  o.name = name;
  o.age = age;
  o.job = job;

  o.sayName = function(){
    alert(this.name);
  }
  return o;
}
var person1 = createPerson('peter', 20, 'software engineer');
var person2 = createPerson('jim', 24, 'doctor');
```

构造函数模式

```
function Person(name, age, job){
 this.name = name;
 this.age = age;
 this.job = job;

 this.sayName = function(){
  alert(this.name);
 }
}

var person1 = new Person('peter',20, 'software engineer');
var person2 = new Person('jim', 24, 'doctor');

person1 instanceof Object; // true
person1 instanceof Person; // true
person2 instanceof Object; // true
person2 instanceof Person; // true
```

函数转移到构造函数外部

```
function Person(name, age, job){
 this.name = name;
 this.age = age;
 this.job = job;
 this.sayName = sayName;
}

function sayName(){
 alert(this.name);
}

var person1 = new Person('peter', 20, 'doctor');
var person2 = new Person('jin', 24, 'software engineer');

person1.sayName === person2.sayName; // true
```

原型模式

- 每个函数都有一个prototype属性
- prototype属性是一个指针，指向一个对象
- 对象包含特定类型的所有实例共享的属性和方法

```
function Person(){}

Person.prototype.name = 'Pete';
Person.prototype.age = 24;
Person.prototype.job = 'Doctor';
Person.prototype.sayName = function(){
 alert(this.name);
}

var person1 = new Person();
person1.sayName(); // 'Pete'

var person2 = new Person();
person2.sayName(); // 'Pete'

person1.sayName === person2.sayName; // true
```

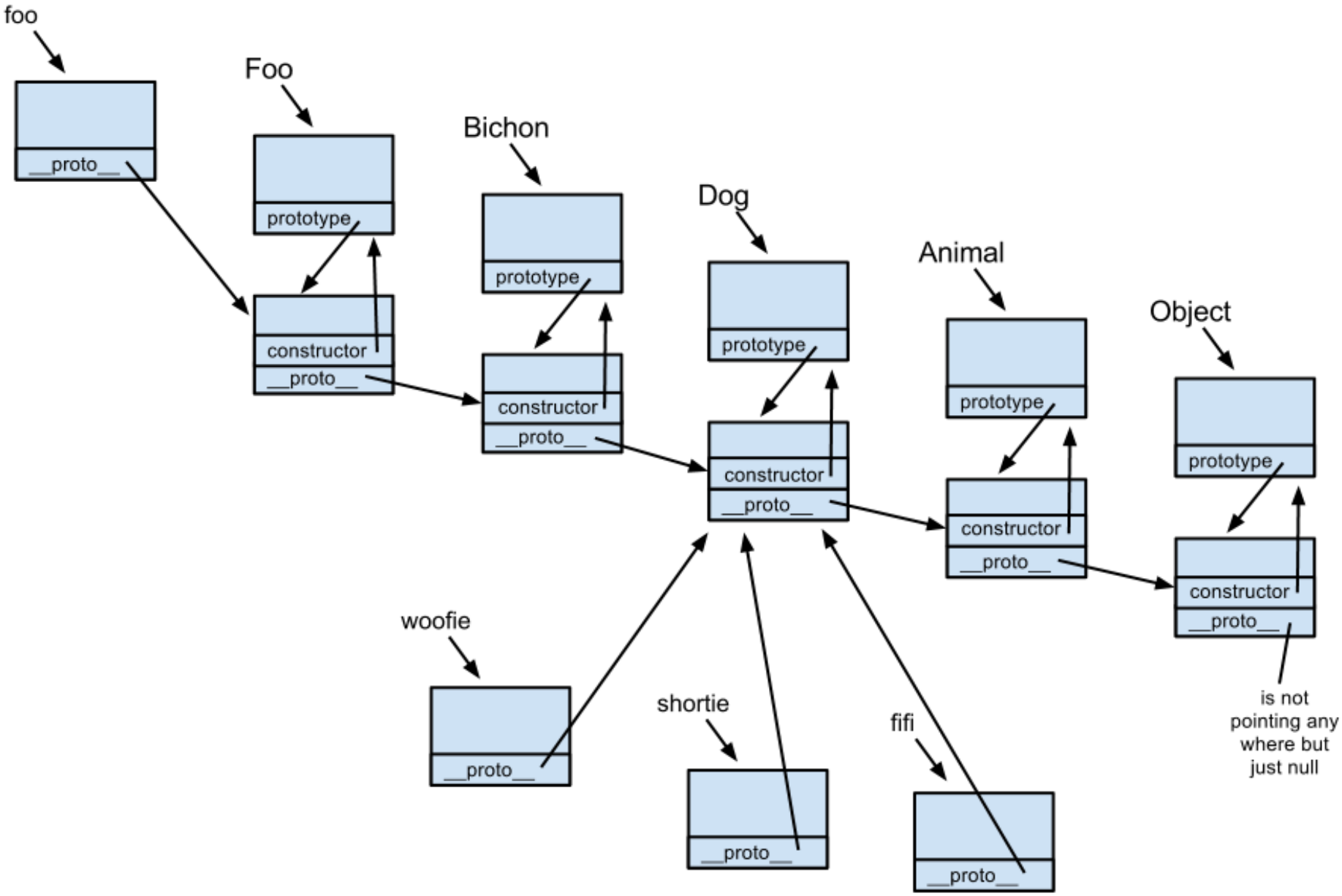
原型链

```
function Person(){}

Person.prototype.name = 'Pete';
Person.prototype.age = 24;
Person.prototype.job = 'Doctor';
Person.prototype.sayName = function(){
  alert(this.name);
}

var person1 = new Person();
var person2 = new Person();

person1.name = 'John';
alert(person1.name); // John
alert(person2.name); // Pete
```


foo
__proto__
Foo
prototype
constructor
__proto__
Bichon
prototype
constructor
__proto__
Dog
prototype
constructor
__proto__
Animal
prototype
constructor
__proto__
Object
prototype
constructor
__proto__
is not pointing any where but just null
woofie
__proto__
shortie
__proto__
fifi
__proto__

- 子类中的变量定义会“屏蔽”父类中的变量定义

constructor

- Person.prototype.constructor //指向Person

jQuery扩展

```
//构造jQuery对象
var jQuery = (function(){
    var jQuery = function(selector,context){
            return new jQuery.fn.init(selector,context,rootjQuery);
        },
        //一堆局部变量声明
    jQuery.fn = jQuery.prototype = {
        constructor: jQuery,
        init: function(selector,context,rootjQuery){ ...},
        //一堆原型属性和方法
    };
    jQuery.fn.init.prototype = jQuery.fn;
    jQuery.extend = jQuery.fn.extend = function(){ ...};
    jQuery.extend({
        //一堆静态属性和方法
    });
    return jQuery;
})();
```

```
jQuery.fn.extend({
  check: function() {
    return this.each(function() {
      this.checked = true;
    });
  },
  uncheck: function() {
    return this.each(function() {
      this.checked = false;
    });
  }
});
// jQuery.fn是jQuery的prototype
```

- jQuery.extend()和jQuery.fn.extend()是一样的，用于把一个或者多个对象的属性合并到jQuery对象
- 除了核心和选择器之外，jQuery的其他功能都使用extend扩展，跟第三方插件是一样的

jQuery.fn.init(selector, context, rootjQuery)

- (“#id”)
- (“.class”)
- (“img”)

```
   <a href="example.html" hreflang="en">Some text</a>
   <a href="example.html" hreflang="en-UK">Some other text</
a>
   <a href="example.html" hreflang="english">will not be
outlined</a>

   <script>
   $( "a[hreflang|='en']" ).css( "border", "3px dotted
green" );
   </script>

   [name=”value”]
   [name!=”value”]
   [name|=”value”]
   [name*=”value”]
   [name~=”value”]
   [name^=”value”]
   [name$=”value”]
```

	selector		context	示　例
1	可以转换为 false		—	$()
2	DOM 元素		—	$(document.body)
3	字符串	“body”	—	$('body')
4		单独标签	—	$('<div>') $(' ',{'class': 'test'})
5		复杂 HTML 代码	—	$(' abc ')
6		“#id”	undefined	$('#id')
7		选择器表达式	undefined	$('div p')
8		选择器表达式	jQuery 对象	$('div p', $('#id'))
9		选择器表达式	DOM 元素	$('div.foo').click(function() { $('span', this).addClass('bar'); });
10	函数		—	$(function(){ ... })
11	jQuery 对象		—	$($('div p'))
12	其他任意类型的值		—	$({ abc: 123 }) $([1, 2, 3])

```
// Handle $(DOMElement)
if (selector.nodeType){
    this.context = this[0]= selector;
    this.length = 1;
    return this;
}
```

jQuery 2.1源码

jQuery文件结构

- src/:源码全部在src目录下，全部是AMD规范的，入口文件是src/jquery.js，这个文件声明了全部的依赖，打包压缩就是从这个文件开始的。
- test/: 单元测试
- build/: jQuery的grunt插件，包括 build, diet, pre-uglify, post-uglify等, 还有一个r.js是requirejs用来合并压缩代码的脚本
- dist/:最后生成的jquery.js和jquery.min.js都放在这里
- bower_components:/ 用bower管理的三个依赖：sizzle.js,require.js,quint.js，执行 bower install 后会被安装到 这里
- Gruntfile.js：grunt 的配置文件
- package.json:npm配置文件
- bower.json:bower配置文件
- ……

Build: space between curly and paren is optional ···

timmywil authored 5 days ago

latest commit `63a577aa0b`

build	Build: space between curly and paren is optional	2 days ago
external	Build: update Sizzle	2 months ago
src	Core: Switch from modules to just window.setTimeout etc.	4 days ago
test	Offset: add tests for hidden elements + scroll	5 days ago
.editorconfig	Misc: Need for speed removed by `9ad6e7e`	8 months ago
.gitattributes	Build: change .gitattributes; use system line ends for non-JS files	a year ago
.gitignore	Core: Make jQuery objects iterable	7 days ago
.jscsrc	Core: Make jQuery objects iterable	7 days ago
.jshintignore	Core: Make jQuery objects iterable	7 days ago
.jshintrc	Build: remove deprecated JSHint options	4 months ago
.mailmap	Authors: Update AUTHORS.TXT and .mailmap	6 months ago
.npmignore	Build: ignore test dependencies for npm install	a year ago
.travis.yml	Tests: add the current version of node and iojs to the travis config	3 months ago
AUTHORS.txt	Release: update AUTHORS.txt	5 months ago
CONTRIBUTING.md	CONTRIBUTING: Condense info and add directions to other resources	7 months ago
Gruntfile.js	Core: Make jQuery objects iterable	7 days ago

src/jquery.js

```
define([
 "./core",
 "./selector",
 "./traversing",
 "./callbacks"//以及其他模块……
], function( jQuery ) {

return (window.jQuery = window.$ = jQuery);

});
```