邂逅jQuery

刘军 liujun

目录
content

认识jQuery

- **jQuery 读音为：/ˈdʒeɪkwɪəri/ （简称：jQ），是一个快速、小型且功能丰富的 JavaScript 库，官网对jQuery的描述：**
  - 使HTML文档遍历、操作、事件处理、动画和 Ajax 之类的事情变得更加简单。
  - 具有易于使用的 API，可在多种浏览器中使用。
  - jQuery 结合多功能性和可扩展性，改变了数百万人编写 JavaScript 的方式。

- **jQuery官网：https://jquery.com/**

Current Active Support

Desktop

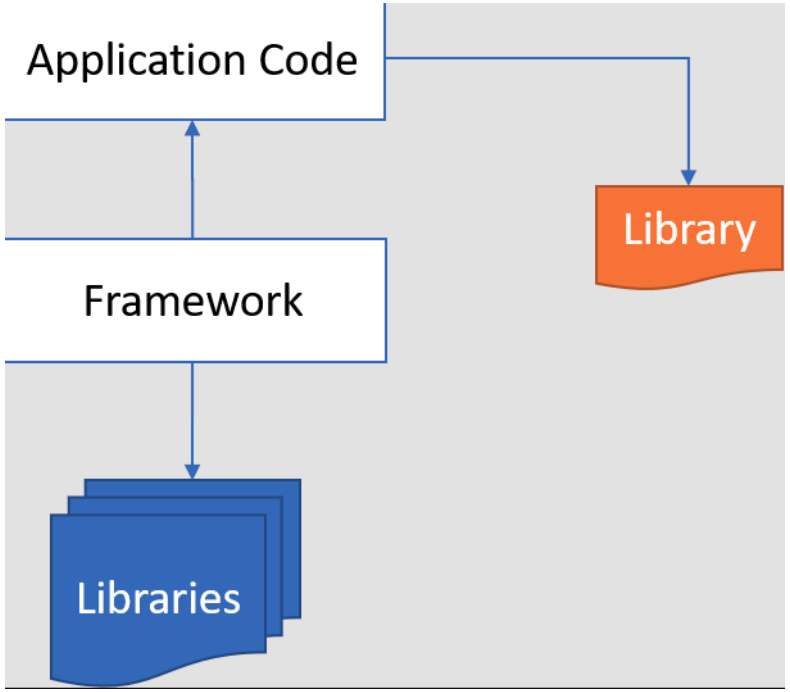
- Chrome: (Current - 1) and Current
- Edge: (Current - 1) and Current
- Firefox: (Current - 1) and Current, ESR
- Internet Explorer: 9+
- Safari: (Current - 1) and Current
- Opera: Current

Mobile

- Stock browser on Android 4.0+[1]
- Safari on iOS 7+

库(library)和框架(framework)的概念

- **随着JavaScript的普及，以及越来越多人使用JavaScript来构建网站和应用程序**
  - JavaScript社区认识到代码中存在非常多相同的逻辑是可复用的。
  - 因此社区就开始对这些相同的逻辑的代码封装到一个JavaScript文件中。
  - 这个封装好的JavaScript文件就可称为JavaScript库或JavaScript框架。



- **库(library)**
  - JavaScript库是一个预先编写好并实现了一些特定功能的代码片段的集合。
  - 一个库中会包含许多的函数、变量等，可根据需求引入到项目中使用。
  - 一些常见的库有jQuery、Day.js、Lodash和React等
- **框架（framework)**
  - JavaScript框架是一个完整的工具集，可帮助塑造和组织您的网站或应用程序。
  - 提供一个结构来构建整个应用程序，开发人员可以在结构的规则内更安全、更高效地工作。
  - 一些更常见的框架有：Bootstrap、Angular、Vue、Next.js等。

jQuery优点与缺点

- **jQuery的优点**
  - 易于学习：相对于其它的前端框架，jQuery 更易于学习，它支持 JavaScript 的编码风格。
  - 少写多做（Write less, do more）
    - ✓ jQuery提供了丰富的功能(DOM操作、过滤器、事件、动画、Ajax等)。
    - ✓ 可以编写更少可读的代码来提高开发人员的工作效率。
  - 优秀的 API 文档：jQuery 提供了优秀的在线 API 文档。
  - 跨浏览器支持：提供出色的跨浏览器支持 (IE9+)，无需编写额外代码。

```
document.addEventListener("DOMContentLoaded", function() {
    paragraphs = document.getElementsByTagName("p")
    for (i = 0; i <= paragraphs.length; i++) {
        paragraphs[i].innerHTML = "This is a Paragraph";
    }
});
```

- **jQuery的缺点：**
  - jQuery代码库一直在增长（自 jQuery 1.5 起超过 200KB）
  - 不支持组件化开发
  - jQuery 更适合DOM操作，当涉及到开发复杂的项目时，jQuery能力有限。

```
$(document).ready(function() {
    $("p").html("This is a Paragraph")
});
```

jQuery起源和历史

- **早在2005年8月22日，John Resig first hints of a JavaScript library to use CSS selectors（Selectors in JavaScript） with a more succinct syntax than existing libraries（Behaviour ）。**
  - John Resig 首次提出支持CSS选择器的JavaScript库，其语法比现有库（例如：Behaviour ）更简洁。
- **在2006年之前，John Resig（一名从事自己项目的Web开发人员）对编写跨浏览器的JavaScript感到非常繁琐。**
  - 2006年1月16日，John Resig在BarCamp的演讲中介绍了他的新库(jQuery)。

> Finally, I announced my second new release of the evening: jQuery: New Wave Javascript. In a nutshell, this code revolutionizes the way you can get Javascript to interact with HTML – it really is an amazing set of code, and I've dumped a lot of time and effort into getting it right. I'm working on the documentation for the site, right now – which should be ready within the next couple days.

最后，我宣布了今晚发布的第二个新版本:jQuery: New Wave Javascript。简而言之，

这段代码彻底改变了Javascript与HTML交互的方式——它确实是一组令人惊叹的代码，

我已经投入了大量的时间和精力来实现它。我现在正在为网站编写文档，应该会在接下来的几天内准备好。

John is a Principal Architect at Khan Academy

  - 之后John Resig又花了 8 个月的时间完善jQuery库，直到2006-8-26才发布了 1.0 版本。
    - ✓ 原本打算使用 JSelect（JavaScript Selectors）命名该库，但域名都已被占用。

Behaviour vs Selectors in JavaScript

Note: In all of these examples, Behaviour uses 'element.onclick' (or similar bindings) in order to attach an event handler – this is generally accepted as an improper way of doing this, since you will only be able to attach one event handler at a time – newer handlers will overwrite older ones.

```
Behaviour.register({
  '#example li': function(e){
    e.onclick = function(){
      this.parentNode.removeChild(this);
    }
  }
});
```

The same as above, done in my code:

```
$('#example li').bind('click',function(){
  this.parentNode.removeChild(this);
});
```

This is the second example made available on the Behaviour web site. It goes to two disparate branches in the DOM Document and attaches two different event handlers.

```
Behaviour.register({
  'b.someclass' : function(e){
    e.onclick = function(){
      alert(this.innerHTML);
    }
  },
  '#someid u' : function(e){
    e.onmouseover = function(){
      this.innerHTML = "BLAH!";
    }
  }
});
```

The same as above, done using my syntax:

```
$('b.someclass').bind('click',function(){
  alert(this.innerHTML);
});

$('#someid u').bind('mouseover',function(){
  this.innerHTML = 'BLAH!';
});
```

jQuery历史版本

Version	Initial release	Latest update	Minified size (KB)	Additional notes
1.0	August 26, 2006			First stable release
1.1	January 14, 2007			
1.2	September 10, 2007	1.2.6	54.5	
1.3	January 14, 2009	1.3.2	55.9	Sizzle Selector Engine introduced into core
1.4	January 14, 2010	1.4.4	76.7	
1.5	January 31, 2011	1.5.2	83.9	Deferred callback management, ajax module rewrite
1.6	May 3, 2011	1.6.4 (September 12, 2011)[32]	89.5	Significant performance improvements to the attr() and val() functions
1.7	November 3, 2011	1.7.2 (March 21, 2012)[33]	92.6	New Event APIs: .on() and .off(), while the old APIs are still supported.
1.8	August 9, 2012	1.8.3 (November 13, 2012)[34]	91.4	Sizzle Selector Engine rewritten, improved animations and $(html, props) flexibility.
1.9	January 15, 2013	1.9.1 (February 4, 2013)[35]	90.5	Removal of deprecated interfaces and code cleanup
1.10	May 24, 2013	1.10.2 (July 3, 2013)[36]	90.9	Incorporated bug fixes and differences reported from both the 1.9 and 2.0 beta cycles
1.11	January 24, 2014	1.11.3 (April 28, 2015)[37]	93.7	
1.12	January 8, 2016	1.12.4 (May 20, 2016)[38]	94.9	
2.0	April 18, 2013	2.0.3 (July 3, 2013)	81.7	Dropped IE 6–8 support for performance improvements and reduction in filesize
2.1	January 24, 2014	2.1.4 (April 28, 2015)	82.4	
2.2	January 8, 2016	2.2.4 (May 20, 2016)	83.6	
3.0	June 9, 2016[39]	3.0.0 (June 9, 2016)	84.3	Promises/A+ support for Deferreds, $.ajax and $.when, .data() HTML5-compatible
3.1	July 7, 2016	3.1.1 (September 23, 2016)	84.7	jQuery.readyException added, ready handler errors are now not silenced
3.2	March 16, 2017[40]	3.2.1 (March 20, 2017)	84.6	Added support for retrieving contents of `<template>` elements, and deprecation of various old methods.
3.3	January 19, 2018[41]	3.3.1 (January 20, 2018)[42]	84.9	Deprecation of old functions, functions that accept classes now also support them in array format.
3.4	April 10, 2019[43]	3.4.1 (May 1, 2019)[44]	86.1	Performance improvements, `nonce` and `nomodule` support, fixes for radio elements, a minor security fix.
3.5	April 10, 2020[1]	3.5.1 (May 4, 2020)[45]	87.4	Security fixes, `.even()` & `.odd()` methods, `jQuery.trim` deprecated
3.6	March 2, 2021	3.6.0 (March 2, 2021)[46]	90.0[47]	Bug fixes, return JSON when there is a JSONP error

为什么学习jQuery

- jQuery是一个非常受欢迎的JavaScript库，被全球约 7000 万个网站使用。它优秀的设计和架构思想非常值得我们去学习。
- jQuery 的座右铭是“Write less , do more”，它易于学习， 非常适合JavaScript 开发人员学习的第一个库。
- 前端JavaScript库非常多，学习jQuery有利于我们学习和理解其它的JavaScript库（例如：Day.js、Lodash.js等）
- 许多大型科技公司，虽然他们现在不会直接使用jQuery来做项目，但在项目中仍然会借鉴很多jQuery设计思想。
- 因此，了解 jQuery 依然是一个好主意。

jQuery的安装

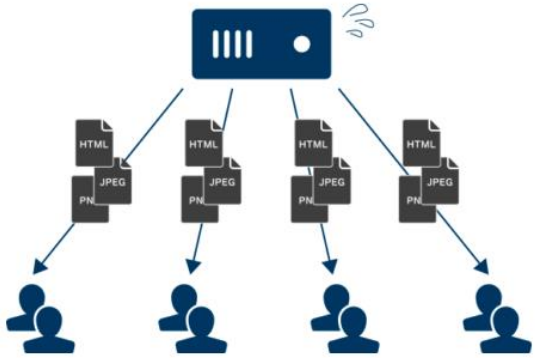
- **jQuery 本质是一个JavaScript 库。**
  - 该库包含了：DOM操作、选择器、事件处理、动画和 Ajax 等核心功能。
  - 现在我们可以简单的理解它就是一个JavaScript文件。
  - 执行该文件中会给window对象添加一个jQuery函数（例如：window.jQuery）。
  - 接着我们就可以调用jQuery函数，或者使用该函数上的类方法。

- 下面我们来看看jQuery安装方式有哪些?
  - 方式一：在页面中，直接通过CDN的方式引入。
  - 方式二：下载jQuery的源文件，并在页面中手动引入。
  - 方式三：使用npm包管理工具安装到项目中（npm在Node基础阶段会讲解）

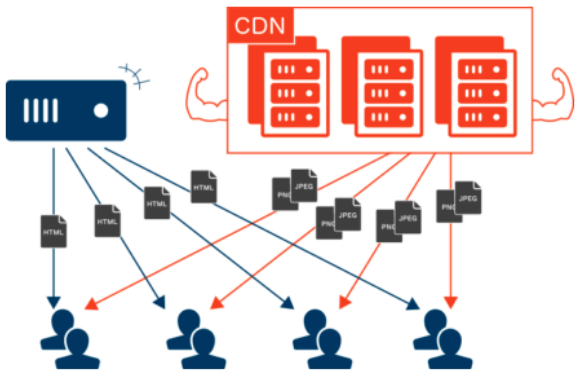
认识CDN

- **什么是CDN呢？CDN称之为内容分发网络（Content Delivery Network或Content Distribution Network，缩写：CDN）**
  - CDN它是一组分布在不同地理位置的服务器相互连接形成的网络系统。
  - 通过这个网络系统，将Web内容存放在距离用户最近的服务器。
  - 可以更快、更可靠地将Web内容(文件、图片、音乐、视频等)发送给用户。
  - CDN不但可以提高资源的访问速度，还可以分担源站的压力。
- **更简单的理解CDN：**
  - CDN会将资源缓存到遍布全球的网站，用户请求获取资源时；
  - 可就近获取CDN上缓存的资源，提高资源访问速度，同时分担源站压力。
- **常用的CDN服务可以大致分为两种：**
  - 自己购买的CDN服务：需要购买开通CDN服务（会分配一个域名）。
    - ✓ 目前阿里、腾讯、亚马逊、Google等都可以购买CDN服务。
  - 开源的CDN服务
    - ✓ 国际上使用比较多的是unpkg、JSDelivr、cdnjs、BootCDN等。

没有 CDN



使用 CDN



方式一：CDN

- jQuery使用CDN方式引入
  - <script src="https://code.jquery.com/jquery-3.6.0.js"></script>
- 下面实现一个 Hello jQuery 的案例：

```
<body>
  <!--
    1.使用CDN方式引入 jQuery 库
    2.当执行完这行代码之后会给window 添加 jQuery函数 和 $ 函数
      例如: window.jQuery = window.$ = function () {}
  -->
  <script src="https://code.jquery.com/jquery-3.6.0.js"></script>

  <script>
    // 3.选中body元素，并给body元素添加: Hello jQuery 文本
    jQuery('body').text('Hello jQuery')
  </script>

</body>
```

方式二：下载源码引入

- 下载jQuery的源码
  - 官网下载：https://jquery.com/download/
  - CDN连接地址下载： https://releases.jquery.com/jquery/
  - GitHub仓库中下载：https://github.com/jquery/jquery
- 下面使用源码的方式引入jQuery：

```
<body>
  <!--
    1.使用源码方式引入 jQuery 库
    2.当执行完这行代码之后会给window 添加 jQuery函数 和 $ 函数
      例如： window.jQuery = window.$ = function () {}
  -->
  <script src="../libs/jQuery.js"></script>

  <script>
    // 3.选中body元素，并给body元素添加：Hello jQuery 文本
    jQuery('body').text('Hello jQuery')
  </script>
</body>
```

方式三：npm安装（了解）

- 使用npm安装jquery到项目中（npm在Node基础阶段会讲解）

```
npm install jquery
```

```
yarn add jquery
```

jQuery初体验-计数器案例

```html
<body>
  <!-- 原生实现 -->
  <button class="sub">-</button>
  <span class="counter">0</span>
  <button class="add">+</button>

  <script>
    var counter = 0;
    let subEl = document.querySelector('.sub')
    let counterEl = document.querySelector('.counter')
    let addEl = document.querySelector('.add')
    subEl.onclick = function() {
      counterEl.textContent = --counter
    }
    addEl.onclick = function() {
      counterEl.textContent = ++counter
    }
  </script>
</body>
```

```html
<body>
  <!-- jQuery实现 -->
  <button class="sub">-</button>
  <span class="counter">0</span>
  <button class="add">+</button>

  <script src="../libs/jQuery.js"></script>
  <script>
    var counter = 0;
    var $couter = $('.counter')
    $('.sub').click(function() {
      $couter.text(--counter)
    })
    $('.add').click(function() {
      $couter.text(++counter)
    })
  </script>
</body>
```

jQuery监听文档加载

- jQuery监听document的DOMContentLoaded事件的四种方案
  - $(document).ready(handler) ： deprecated
  - $("document").ready(handler)： deprecated
  - $().ready(handler)： deprecated
  - **$(handler)：推荐用这种写法，其它可以使用但是不推荐**

```
<body>
  <script src="../libs/jQuery.js"></script>
  <script>
    // $().ready(function() {
    //   var counter = 0;
    //   var $couter = $('.counter')
    //   $('.sub').click(function() {$couter.text(--counter)})
    //   $('.add').click(function() {$couter.text(++counter)})
    // })
    $(function() {
      var counter = 0;
      var $couter = $('.counter')
      $('.sub').click(function() {$couter.text(--counter)})
      $('.add').click(function() {$couter.text(++counter)})
    })
  </script>

  <!-- jQuery实现 -->
  <button class="sub">-</button>
  <span class="counter">0</span>
  <button class="add">+</button>
```

- 监听window的load事件，即网页所有资源（外部连接，图片等）加载完
  - .load(handler)： This API has been removed in jQuery 3.0
  - $(window).on('load', handler) : 推荐写法

注意：这个 API 在 jQuery 3.0 中已经被移除；请使用 `.on( "load", handler )` 代替 `.load( handler )` 和 `.trigger( "load" )` 代替 `.load()`。

```
<!-- jQuery实现 -->
<button class="sub">-</button>
<span class="counter">0</span>
<button class="add">+</button>
<img src="https://api.jquery.com/jquery-wp

<script src="../libs/jQuery.js"></script>
<script>
  $(window).on('load', function() {
    console.log('所有资源加载完毕')
  })
</script>
```

jQuery与其它库的变量名冲突

- **和 jQuery库一样，许多JavaScript库也会使用 $ 作为函数名或变量名。**
  - 在 jQuery 中，$ 是jQuery的别名。
  - 如果我们在使用jQuery库之前，其它库已经使用了 $ 函数或者变量，这时就会出现冲突的情况。
  - 这时我们可以通过调用jQuery中的noConflict函数来解决冲突问题。
  - jQuery在初始化前会先备份一下全局其它库的jQuery和$变量，调用noConflict函数只是恢复之前备份的jQuery和$变量。

```
10852  var
10853
10854      // Map over jQuery in case of overwrite
10855      _jQuery = window.jQuery, // 先备份其它框架的 jQuery 变量
10856
10857      // Map over the $ in case of overwrite
10858      _$ = window.$;  // 先备份其它框架的 $ 变量
10859
10860  jQuery.noConflict = function( deep ) {
10861      if ( window.$ === jQuery ) { // 判断当前的$是否是jQuery
10862          window.$ = _$; // 恢复先前备份其它框架的$
10863      }
10864
10865      if ( deep && window.jQuery === jQuery ) {
10866          window.jQuery = _jQuery; // 恢复先前备份其它框架的jQuery
10867      }
10868
10869      return jQuery; // 返回一个 jQuery 函数
10870  };
```

认识jQuery函数

- **jQuery是一个工厂函数(别名$)，调用该函数，会根据传入参数类型来返回匹配到元素的集合，一般把该集合称为jQuery对象。**
  - 如果传入假值：返回一个空的集合。
  - 如果传入选择器：返回在在documnet中所匹配到元素的集合。
  - 如果传入元素：返回包含该元素的集合。
  - 如果传入HTML字符串，返回包含新创建元素的集合。
  - 如果传入回调函数：返回的是包含document元素集合, 并且当文档加载完成会回调该函数。
  - 因为函数也是对象，所以该函数还包含了很多已封装好的方法。如：jQuery.noConflict、jQuery.rea

```
// Define a local copy of jQuery
jQuery = function( selector, context ) {

 // The jQuery object is actually just the init
 // Need init if jQuery is called (just allow er
 return new jQuery.fn.init( selector, context );
};
```

- jQuery函数的参数：
  - jQuery(selector [, context])：selector 是字符串选择器；context 是匹配元素时的上下文，默认值为 document
    - ✓ jQuery(selector [, context])
    - ✓ jQuery(element)
    - ✓ jQuery(elementArray)
    - ✓ jQuery()
  - jQuery(html [, ownerDocument])
    - ✓ jQuery(html [, ownerDocument])
    - ✓ jQuery(html)
  - jQuery(callback)

```
init = jQuery.fn.init = function( selector, context, root ) {
 var match, elem;

 // HANDLE: $(""), $(null), $(undefined), $(false)
 if ( !selector ) {
  return this;
 }

 // Method init() accepts an alternate rootjQuery
 // so migrate can support jQuery.sub (gh-2101)
 root = root || rootjQuery; //  rootjQuery = jQuery(document)
```

认识jQuery对象

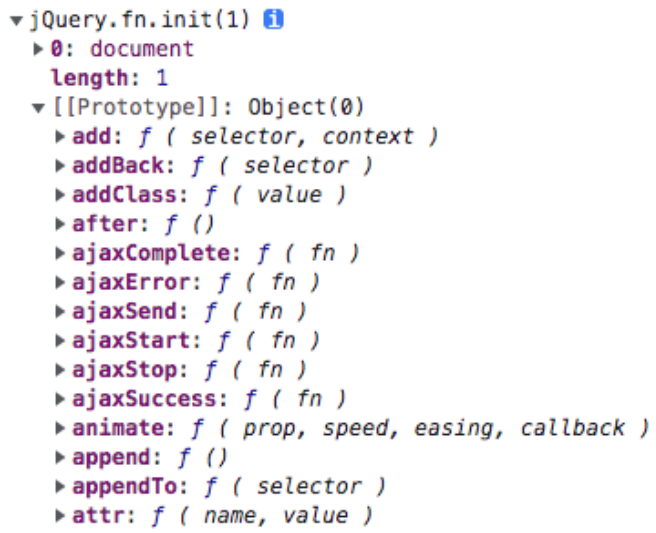
- **jQuery对象是一个包含所匹配到元素的集合，该集合是类数组(array-like)对象。**
  - jQuery对象是通过调用jQuery函数来创建的。
  - jQuery对象中会包含N（>=0）个匹配到的元素。
  - jQuery 对象原型中包含了很多已封装好的方法。例如：DOM操作、事件处理、动画等方法。

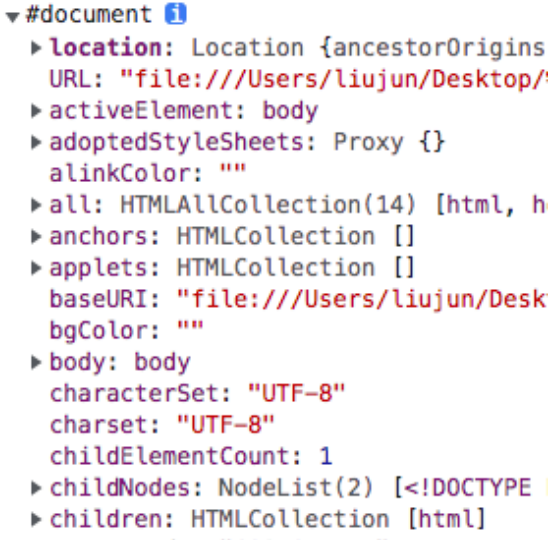
- 下面我们通过调用jQuery函数来新建一个jQuery对象，例如：
  - $() 新建一个空的jQuery对象
  - $(document) 新建一个包含document元素的jQuery对象
  - $('选择器') 新建一个包含所选中DOM元素的jQuery对象

```
▼jQuery.fn.init(1) i
  ▶0: document
    length: 1
  ▼[[Prototype]]: Object(0)
    ▶add: ƒ ( selector, context )
    ▶addBack: ƒ ( selector )
    ▶addClass: ƒ ( value )
    ▶after: ƒ ()
    ▶ajaxComplete: ƒ ( fn )
    ▶ajaxError: ƒ ( fn )
    ▶ajaxSend: ƒ ( fn )
    ▶ajaxStart: ƒ ( fn )
    ▶ajaxStop: ƒ ( fn )
    ▶ajaxSuccess: ƒ ( fn )
    ▶animate: ƒ ( prop, speed, easing, callback )
    ▶append: ƒ ()
    ▶appendTo: ƒ ( selector )
    ▶attr: ƒ ( name, value )
```

jQuery对象 与 DOM Element的区别

- **jQuery对象与DOM Element的区别**
  - 获取的方式不同
    - DOM Element 是通过原生方式获取，例如：document.querySelector()
    - jQuery对象是通过调用jQuery函数获取，例如：jQuery(' ')
  - jQuery对象是一个类数组对象，该对象中会包含所选中的DOM Element的集合。
  - jQuery对象的原型上扩展非常多实用的方法，DOM Element 则是W3C规范中定义的属性和方法。





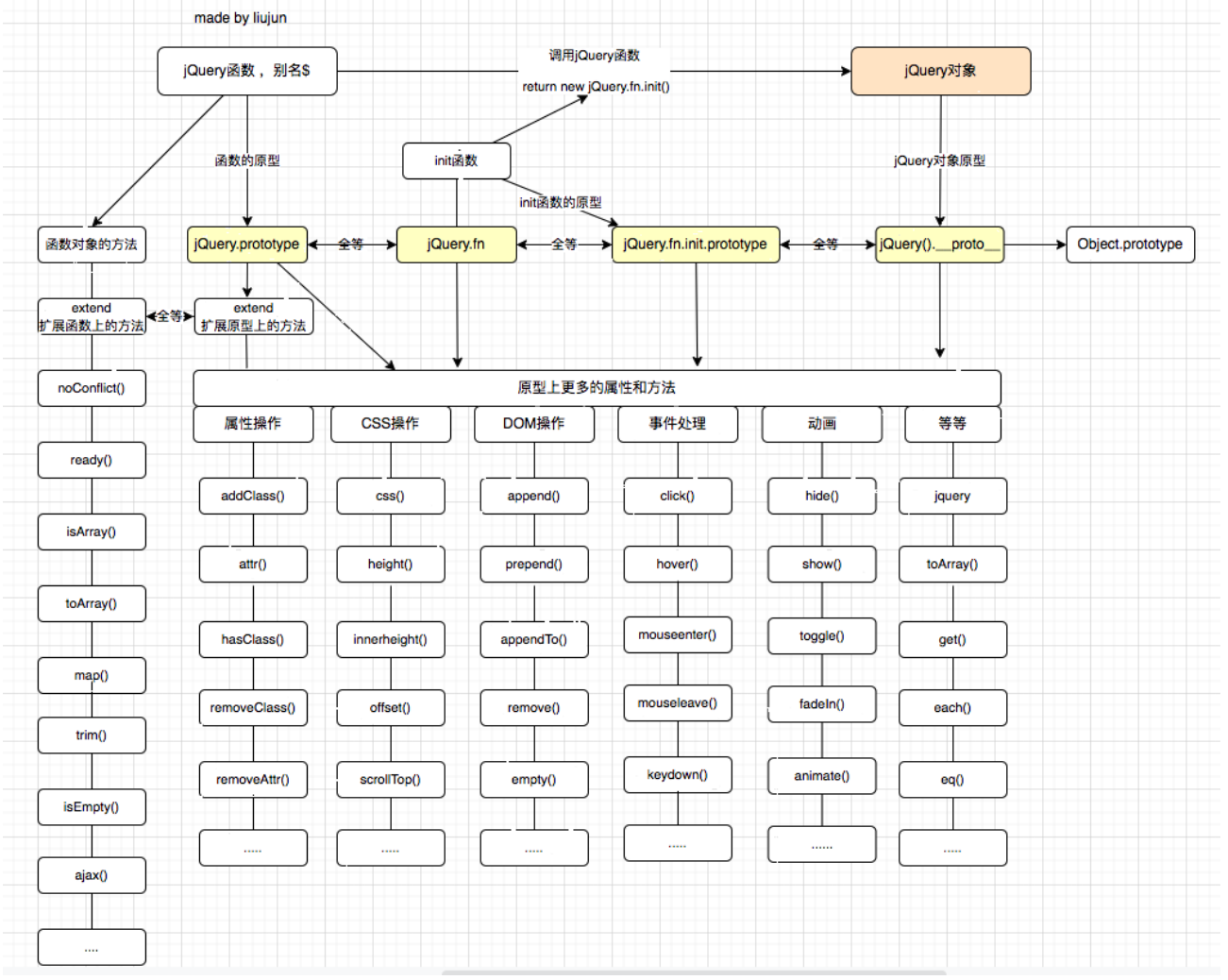
jQuery对象 与 DOM Element 的转换

- **jQuery对象转成DOM Element**
  - .get(index)： 获取 jQuery 对象中某个索引中的 DOM 元素。
    - ✓ index一个从零开始的整数，指示要检索的元素。
    - ✓ 如果index超出范围（小于负数元素或等于或大于元素数），则返回undefined。
  - .get() : 没有参数，将返回jQuery对象中所有DOM元素的数组。

- **DOM Element转成jQuery对象**
  - 调用jQuery函数或者$函数
  - 例如：$(元素)

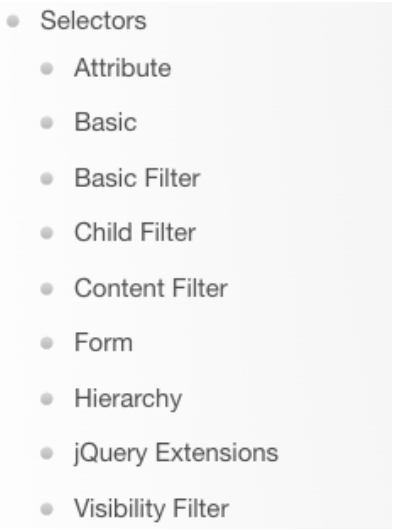
jQuery架构设计图

- 在开始学习jQuery语法之前，我们先来了解一下jQuery的架构设计图。
- jQuery架构设计图如下：



jQuery的选择器(Selectors)

- jQuery函数支持大部分的CSS选择器，语法：jQuery（'字符串格式的选择器'）
  - 1.通用选择器（*）
  - 2.基本选择器（id, class, 元素）
  - 3.属性选择器（ [attr] , [atrr=" value "] ）
  - 4.后代选择器（div > span，div span）
  - 5.兄弟选择器（div + span , div ~ span）
  - 6.交集选择器（div.container）
  - 7.伪类选择器（:nth-child()，:nth-of-type()，:not()， 但不支持状态伪类 :hover, :focus...）
  - 8.内容选择器（:empty，:has(selector)），empty指选中的元素没有子元素或文本； has指选中的元素是否存在某个子元素
  - 9.可见选择器（:visible，:hidden）
  - 10.jQuery扩展选择器： （:eq()，:odd，:even, :first, :last ）
  - .....



VSCode生成代码片段

- 在线生成代码片段地址：https://snippet-generator.app/

jQuery Page | jQuery Page | VSCode | Sublime Text | Atom

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

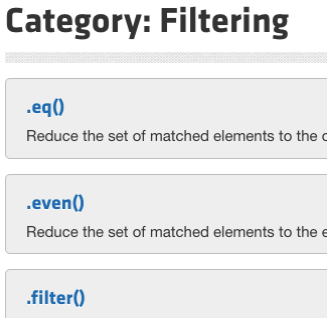
To declare a placeholder (⌘ + i): ${1:example} | More info

```
"jQuery Page": {
  "prefix": "jQuery Page",
  "body": [
    "<!DOCTYPE html>",
    "<html lang=\"en\">",
    "<head>",
    "  <meta charset=\"UTF-8\">",
    "  <meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\">",
    "  <meta name=\"viewport\" content=\"width=device-width, initial-",
    "  <title>Document</title>",
    "</head>",
    "<body>",
    "  ",
    "</body>",
    "</html>"
  ],
  "description": "jQuery Page"
}
```

Copy snippet

jQuery过滤器(Filtering) API

- jQuery过滤器API（即jQuery原型上的方法）
  - 1.eq(index)：从匹配元素的集合中，取索引处的元素， eq全称(equal 等于)，返回jQuery对象。
  - 2.first()：从匹配元素的集合中，取第一个元素，返回jQuery对象。
  - 3.last(): 从匹配元素的集合中，取最后一个元素，返回jQuery对象。
  - 4.not(selector)：从匹配元素的集合中，删除匹配的元素，返回jQuery对象。
  - 5.filter(selector): 从匹配元素的集合中，过滤出匹配的元素，返回jQuery对象。
  - 6.find(selector): 从匹配元素集合中，找到匹配的后代元素，返回jQuery对象。
  - 7.is(selector|element|．): 根据选择器、元素等检查当前匹配到元素的集合。集合中至少有一个与给定参数匹配则返回true。
  - 8.odd() :将匹配到元素的集合减少为集合中的奇数，从零开始编号，返回jQuery对象。
  - 9.even()：将匹配到元素的集合减少到集合中的偶数，从零开始编号，返回jQuery对象。
  - 10.支持链式调用
  - .....

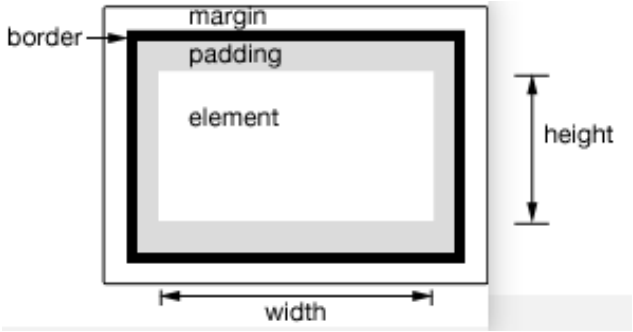


jQuery对文本的操作

- **.text()、.text(text)**
  - 获取匹配到元素集合中每个元素组合的文本内容，包括它们的后代，或设置匹配到元素的文本内容。
  - 相当与原生元素的textContent属性。
- **.html()、html(htmlString)**
  - 获取匹配到元素集合中第一个元素的HTML内容，包括它们的后代，或设置每个匹配元素的 HTML 内容。
  - 相当与原生元素的innerHTML属性。
- **.val()、.val(value)**
  - 获取匹配到元素集合中第一个元素的当前值 或 设置每个匹配到元素的值。
  - 该.val()方法主要用于获取input,select和等表单元素的值。
  - 相当与获取原生元素的value属性。

jQuery对CSS的操作

- **.width()、.width(value)**
  - 获取匹配到元素集合中第一个元素的宽度或设置每个匹配到元素的宽度。
- **.height()、height(value)**
  - 获取匹配到元素集合中第一个元素的高度或设置每个匹配到元素的高度。



- **.css(propertyName)、.css(propertyNames)**
  - 获取匹配到元素集中第一个元素样式属性的值，底层是调用getComputedStyle函数获取。
  - .css("width")和.width()之间的区别:
    - ✓ width()返回一个无单位的像素值（例如，400），而css()返回一个具有完整单位的值（例如，400px）
- **.css(propertyName, value)、.css(properties)**
  - 为每个匹配到元素设置一个 或 多个 CSS 属性。
  - 调用css方法添加样式会直接把样式添加到元素的style属性上。

Class属性的操作

- **.addClass(className)、.addClass(classNames)、.addClass(funcntion)**
  - 将指定的类添加到匹配元素集合中的每个元素，每次都是追加class。
  - 底层调用的是setAttribute("class", finalValue)方法添加class。
- **.hasClass(className)**
  - 是否给任意匹配到的元素分配了该类。
  - 底层是通过getAttribute("class").indexOf()来判断是否存在。
- .removeClass()、.removeClass(className)、.removeClass(classNames)、.removeClass(function)
  - 给匹配元素集中的每个元素删除单个类、多个类或所有类。
  - 底层调用的是setAttribute("class", finalValue)方法。
- .toggleClass()、.toggleClass(className[,state])、.toggleClass(classNames[,state])
  - 根据类的存在或状态参数的值，在匹配到元素的集合中，给每个元素添加或删除一个或多个类。

```
removeClass: function( value ) {
	var classes, elem, cur, curValue, clazz, j, finalValue,
		i = 0;

	if ( isFunction( value ) ) {
		return this.each( function( j ) {
			jQuery( this ).removeClass( value.call( this, j, getClass( this ) ) );
		} );
	}
```

```
toggleClass: function( value, stateVal ) {
	var type = typeof value,
		isValidValue = type === "string" || Array.isArray( value );

	if ( typeof stateVal === "boolean" && isValidValue ) {
		return stateVal ? this.addClass( value ) : this.removeClass( value );
	}

	if ( isFunction( value ) ) {
		return this.each( function( i ) {
			jQuery( this ).toggleClass(
				value.call( this, i, getClass( this ), stateVal ),
				stateVal
			);
		} );
	}
```

attributes和property属性的操作

- **.attr(attributeName)**
  - 获取匹配元素集和中第一个元素的属性值，底层调用了原生的 getAttribute() API
- **.attr(attributeName, value)、.attr(attributes)**
  - 为每个匹配元素设置一个或多个属性，底层调用了原生的 setAttribute() API
- **.removeAttr(attributeName)**
  - 在匹配到元素的集中，给每个元素删除一个属性。
  - 底层调用了原生的 removeAttribute() API
- **.prop(propertyName)**
  - 获取匹配到元素集合中第一个元素的属性值
- **.prop(propertyName，value)、.prop(propertys)**
  - 为每个匹配元素设置一个或多个属性。
- removeProp(**propertyName**)
  - 删除匹配元素集的属性,(只能删除用户自定义添加的prop，不能删除元素本身的属性)。

```
<!--
    标准的attribute: 比如id、class、href等(会在DOM对象上创建对应的prop)
    非标准的attribute: 比如name、age、height等(只会存在attributes属性上)
-->
<div id="container" class="box"
     name="liujun" age="17" height="1.66">
</div>
```

```
▼ div#container.box
    accessKey: ""
    align: ""
    ariaAtomic: null
    ariaAutoComplete: null
    hidden: false
    id: "container"
    inert: false
    innerHTML: "\n  "
    innerText: ""
    inputMode: ""
    isConnected: true
    isContentEditable: false
    lang: ""
  ▶ lastChild: text
    lastElementChild: null
    localName: "div"
    namespaceURI: "http://ww
  ▶ nextElementSibling: scri
  ▶ nextSibling: text
    nodeName: "DIV"
    nodeType: 1
    nodeValue: null
    nonce: ""
    offsetHeight: 0
    offsetLeft: 8
```

自定义data-xx属性的操作

- .data()、.data(key)
  - 获取匹配元素集中第一个元素的自定义属性的值
- .data(key, value) 、.data(obj)
  - 为每个匹配元素设置一个或多个自定义属性
- .removeData([name])
  - 会删除data()函数给匹配元素属性添加的数据 和 data()函数绑定的自定义属性。
  - data函数添加的属性会被移除，但是如果属性同时在签上定义了就不会被移除。

```
<div class="box" data-name="why" data-age="18"></div>

<script>
  var boxEl = document.querySelector(".box")
  console.log(boxEl.dataset.name)
  console.log(boxEl.dataset.age)
</script>
```

```
// 1.拿到元素的自定义属性
console.log($('div').data()) // 拿到所有
console.log($('div').data('age')) // 拿到指定
console.log($('div').data('name')) // 拿到指定
```

jQuery的DOM操作-插入内容

- **.append(content [, content]) 、append(function)**
  - 将参数的内容插入到匹配元素集中每个元素的末尾。
    - ✓ content 的类型: DOM element, text node, array of elements and text nodes, HTML string, or jQuery object

```
// Flatten any nested arrays
args = flat( args );
```

- **.prepend(content [, content]) 、prepend(function)**
  - 将参数的内容插入到匹配元素集中每个元素的开头。

- **.after(content [, content]) 、after(function)**
  - 在匹配元素集中的每个元素之后，插入由参数指定的内容。
- **.before(content [, content])、before(function)**
  - 在匹配元素集中的每个元素之前，插入由参数指定的内容。

```
<ul>
  <li class="li-1">li-1</li>
  <li class="li-2">li-2</li>
  <li class="li-3">li-3</li>
  <li class="li-4">li-4</li>
  <li class="li-5">li-5</li>
</ul>
```

jQuery的DOM操作-插入内容

- .appendTo(target)
  - 将匹配元素集中的每个元素插入到目标元素的末尾。
    - ✓ target的类型：A selector, element, HTML string, array of elements, or jQuery object。
- .prependTo(target)
  - 将匹配元素集中的每个元素插入到目标元素的开头。
- .insertAfter(target)
  - 在目标元素之后，插入匹配元素集中的每个元素。
- .insertBefore(target)
  - 在目标元素之前，插入匹配元素集中的每个元素。

jQuery的DOM操作-移除/替换/克隆

- **.empty(): 删除匹配元素集的所有子节点，自身不会删除。**
- **.remove( ) 、.remove([selector])**
  - 删除匹配的元素集，自身也会删除。
    - ✓ selector参数：字符串类型选择器。筛选匹配元素集的元素来删除

```
clone: function( dataAndEvents, deepDataAndEvents ) {
    dataAndEvents = dataAndEvents == null ? false : dataAndEvents;
    deepDataAndEvents = deepDataAndEvents == null ? dataAndEvents : deepDataAndEvents;

    return this.map( function() {
        return jQuery.clone( this, dataAndEvents, deepDataAndEvents );
    } );
},
```

- **.replaceAll(target):** 用匹配到的元素集替换每个目标元素。
- **.replaceWidth(newContent)、.replaceWidth(function)**
  - 用新内容替换匹配元素集中的每个元素，并返回被移除的元素集。
    - ✓ newConten参数的类型： HTML string, DOM element, array of DOM elements, or jQuery object

```
clone: function( elem, dataAndEvents, deepDataAndEvents ) {
    var i, l, srcElements, destElements,
        clone = elem.cloneNode( true ),
        inPage = isAttached( elem );
```

- **.clone()、.clone(withDataAndEvents)**
  - 对匹配的元素集执行深度复制，底层是调用了elem.cloneNode(true)来复制元素。
    - ✓ withDataAndEvents参数：布尔值，是否复制该元素的事件处理程序和数据，默认值为false。