* J+ UI Library

一个即学即用的组件库

主页: <http://www.jplusui.com>

目录

[目录 2](#_Toc321342766)

[概述 4](#_Toc321342767)

[一、 J+ UI Library介绍 4](#_Toc321342768)

[二、 开始体验 – 第一个实例 4](#_Toc321342769)

[(一) 坏境需求 4](#_Toc321342770)

[(二) Hello World 5](#_Toc321342771)

[概念 6](#_Toc321342772)

[一、 组件和组件库 6](#_Toc321342773)

[(一) 组件 6](#_Toc321342774)

[(二) 组件库 6](#_Toc321342775)

[(三) 组件名字空间 6](#_Toc321342776)

[二、 引用组件 6](#_Toc321342777)

[(一) 最简单的引入方法 6](#_Toc321342778)

[(二) 组件的合并 7](#_Toc321342779)

[(三) 组件的直接引用 7](#_Toc321342780)

[三、 组件机制 7](#_Toc321342781)

[(一) 组件工作原理 7](#_Toc321342782)

[(二) 组件生命周期 8](#_Toc321342783)

[四、 代码设计 9](#_Toc321342784)

[(一) 面向对象思想 9](#_Toc321342785)

[(二) 类 9](#_Toc321342786)

[(三) 构造函数 10](#_Toc321342787)

[(四) 继承 11](#_Toc321342788)

[(五) 事件 12](#_Toc321342789)

[(六) 接口 13](#_Toc321342790)

[(七) 调试API 13](#_Toc321342791)

[(八) 系统API 14](#_Toc321342792)

[第五章 节点操作 19](#_Toc321342793)

[第一节 事件 19](#_Toc321342794)

[第二节 属性 19](#_Toc321342795)

[第三节 坐标 20](#_Toc321342796)

[第四节 节点 20](#_Toc321342797)

[第六节 获取节点 21](#_Toc321342798)

[第六章 Ajax 23](#_Toc321342799)

[*文章导读* 23](#_Toc321342800)

[第七章 控件 25](#_Toc321342801)

# 概述

## J+ UI Library介绍

J+ UI Library是一个Javascript组件库。它主要用于三个目的:

* 快速开发普通前端页面。
* 开发跨平台的RIA应用。
* 方便用户二次开发，创造属于自己的组件库。

J+ 的主要特性有:

* 轻量，但功能丰富，核心部分普通压缩后为20K。
* 虽然类库自带底层，但也可以使用 jQuery 作为底层支持。
* 所有UI组件都是轻量级的封装，更符合前端开发人员的习惯，也方便自定义样式。
* 作为国产类库，对国内的特有需求提供支持。如城市选择器、解决中文乱码问题。
* 使用HTML5，但对低版本浏览器有降级处理，部分支持移动平台。

## 开始体验 – 第一个实例

### 坏境需求

#### 浏览器

J+ 支持所有主流浏览器，但在最新版浏览器能获得最佳体验。对于开发者人员，建议使用下列浏览器之一:

* [FireFox](http://www.mozilla.com/en-US/firefox/fx/) 4 + [Firebug](http://getfirebug.com/) (推荐)
* [Google Chrome](http://www.google.com/chrome/) 10+
* IE9+
* [Safari](http://www.apple.com/safari/download/) 5+

#### 服务器

J+ 的正常运行不需要服务器坏境。但是我们强烈建议在服务器坏境下调试开发，因为J+开发版会自动载入被引入的代码模块，而在[file:///](http://en.wikipedia.org/wiki/File_URI_scheme)协议下，这个功能将不能使用。服务器可以是任意的HTTP服务器，如 [Apache](http://httpd.apache.org/docs/current/platform/windows.html) 。

#### 下载 J+ UI 开发包(SDK)

从 <http://www.jplusui.com/download/> 下载最新版 J+ SDK。解压 jplus.zip 到web服务器跟目录下的 jplus 文件夹，然后访问 <http://localhost/jplus/index.html>。如果浏览器中出现了欢迎界面，说明一切准备工作都已完成。

### Hello World

创建一个新的 HTML 文件，内容如下:

<html>

<head>

<title>Hello Ext</title>

<link rel="stylesheet" type="text/css" href="jplus/build/ styles/milk.css">

<script type="text/javascript" src="jplus/build/ scripts/ milk.js"></script>

<script type="text/javascript">

function btn\_click (){

Dom.get("btn").setText("我被点了");

}

</script>

</head>

<body>

<button id="btn" type="button" class="x-button" onclick="btn\_click()">点我</button>

</body>

</html>

其中， milk.css 和 milk.js 分别包含了 milk 风格样式的全部组件的CSS和JS代码。在浏览器中打开这个文件，您可以看到页面上显示了一个按钮，点击按钮后，按钮的文字出现了变化。

以上示例演示了如何使用 button 组件。更多的示例可要翻看API文档和演示，它们位于下载的 SDK 里的 docs 目录。

# 概念

## 组件和组件库

### 组件

J+ 的所有功能都是通过组件的形式提供的。比如需要在网页上显示一个Button(按钮)，则需要引入Button 组件。一个组件可以只有css代码，也可以既有css代码，又有 js代码。

一个网页总是需要用到很多组件。您可以将库里需要用到的所有组件合成一个文件，然后在项目中引入这个文件。具体合成方法见第四章 自定义合成。

### 组件库

不同的组件库下的组件其主题风格也是不同的。比如 milk组件库下的组件都是以灰色为主色调的。一般一个项目只选择某一个组件库。J+ 提供了完整的组件库开发坏境，它可要将多套不同风格的组件库放在一起开发。您也可以根据已有的组件库创建出自己风格的组件库。具体的组件开发见 第五章 添加自己的组件。

### 组件名字空间

为了区分不同组件，J+为不同的组件定义了独立的名字空间。

名字空间由三部分组成: 组件库名.分类名.组件名。比如 milk 组件库下的 form 分类的 textbox 组件，其名字空间为 Milk.Form.TextBox 。如果需要更精确地定位组件，应该使用组件的名字空间来表示这个组件。

通过一个组件的名字空间，我们可以方便地找到一个组件对应的css和js文件。比如 Milk.Form.TextBox 组件对应的文件有:

* Css源码: jplus/milk/form/assets/styles/textbox.css
* Js源码: jplus/milk/form/assets/scripts/textbox.js
* 组件演示网页: jplus/milk/form/textbox.html

注意这里的名字空间仅仅是一个组件的叫法，和JS代码无关。

## 引用组件

### 最简单的引入方法

最简单的引用方式就是在html中引用 jplus/build/styles/milk.css 和 jplus/build/ scripts/ milk.js 文件。这2个文件包含了全部组件的代码。只需引入一次即可立即体验所有组件。

如果需要将文件夹拷贝到其它目录，建议复制并重命名 build 文件夹。

### 组件的合并

每个组件开发的时候是独立的。实际使用的时候，一般将用到的组件合并为一个代码。

合成的时候，组件相关的js被合为最终的js文件，相关的css被合为最终的css文件，相关的图片资源也会被自动拷贝出来。

当然，除了合并组件，您也可以使用模块加载系统动态加载组件。

### 组件的直接引用

如果希望直接引入单个组件，以 Milk.Button.Button 为例，需要引入下列文件:

* Jplus/mlik/button/assets/styles/button.css
* Jplus/mlik/button/assets/scripts/button.js
* 依赖的其它组件相关的文件。

如果是在服务器坏境下使用J+，则只需引入 button.js ,这个文件会自动引入相关的js和css文件。

## 组件机制

### 组件工作原理

即使是再复杂的组件，其本质上还是由HTML+CSS+JS 组成的。J+提供了一个强大的组件机制，保证所有组件都可以使用HTML代码或者纯JS代码创建。

#### 使用纯HTML代码创建组件

以 Milk.Core.Button 为例: 只需在页面上书写如下 HTML 代码，就能显示一个按钮。

<button class=”x-button”>按钮</button>

每个组件都有不同的HTML，具体参见组件的演示页。使用纯HTML代码创建组件特别适合普通的网站开发。

自然，并不是所有组件都能使用纯HTML代码创建的。

#### 使用纯JS代码创建组件

以 Milk.Core.Button 为例: 只需在页面上书写如下 JS 代码，就能显示一个按钮。

new Button().setText(‘按钮’).appendTo()

如果您更习惯于类似 jQuery UI 的options模式，也可以写成:

new button({text: ‘按钮’, appendTo: true});

所有组件都能使用纯JS创建，这样的形式更适合在 RIA 使用。

#### 使用HTML + JS代码创建组件

以 Milk.Core.Button 为例: 只需在页面上书写如下 HTML 代码，就能显示一个按钮。

<button id=”btn” class=”x-button”>按钮</button>

new Button(“btn”);

HTML 保证了界面需求，然后使用JS来操作HTML。这是比较广泛的用法。

#### 使用纯HTML + JS Parser代码创建组件

JS Parser 允许您不需写任何CSS和JS代码，只需通过HTML和HTML属性即可定义一个组件，如:

<button id=”btn” class=”x-button” icon=”home”>按钮</button>

### 组件生命周期

一个组件从创建到删除经历了四个步骤:

其中创建不一定是使用JS创建，也可以是从页面上获取已存在的元素。

初始化主要是绑定一些默认事件等。

渲染则是将节点添加到DOM树。

移除则是将节点从 DOM 树删除。

## 代码设计

J+ 没有创造很多概念，但由于面向对象思想，所以很多概念对于Jser来说肯能是新词。

### 面向对象思想

假如需要实现修改页面的一个文本框的值的内容。首先要知道，页面的所有 HTML 标签都是一个 Element 类实例， 先必须获得这个文本框类的实例。往往我们通过 var input = document.getElementById("id") 获取这个文本框， 这里 input 就是 Element 类实例， 要想获取这个框的内容， 需要调用 Element 类的动态成员value ， 因此 最后的结果是: input.value 。

要想实现一个功能，只要查阅实现这个功能的类，然后研究这个类的成员即可。

### 类

Javascript 跟我们说: 一切都是对象。类就是对象。所谓的对象就是函数或变量的集合。一个对象的具体化叫这个对象的实例。 比如有个书本类， 那所有的书本都是这个类的实例。 在 Javascript 中， String 是一个类， "aaa" 是 String 类的实例。

J+ 约定所有类名大写，如 MyBook， Object。

一般的类使用 new 实例化， 如 var a = new Date(); 这里的 a 就是 Date 类的实例。

一个类下有2种成员----静态成员和动态成员。 成员可以是变量(字段)、函数(方法)或属性。

类的静态成员属于类。 如 String.fromCharCode(3) ， 这里 fromCharCode 被 String 对象所含，所以称为 String 类的静态成员。

而类的动态成员属于类的实例。 如 "aaa".substr(1) 中 substr 是String 类实例的成员，所以是 String 类的动态成员。

比如有一个类 A ，它含动态成员 a() 、静态成员 b() ， 那要想调用2个函数，分别应如下调用:

|  |
| --- |
| A.b(); // 静态成员直接这样使用    var instance = new A(); // 创建这个类实例  instance.a(); // 动态成员对实例操作 |

根据以上的理解，可以得出:

Math.sin -> sin 是 Math 类的静态成员

Array.concat -> concat 是 Array 类的静态成员

[].concat -> concat 是 Array 类的动态成员

document.getElementById(‘id’).className -> className 是 Element 类 动态成员

可以在类的 prototype 属性找到类动态成员，如: [].concat 就是 Array.prototype.concat。

在J+ 中，使用如下代码可创建类，这个类拥有动态成员version = 2 ，动态成员 buy, 静态成员 type = “book” :

|  |
| --- |
| var MyBook = new Class({ // new 可以不写  // 在这里写的都是 动态成员  version: 2,    buy: function(){  alert(‘买了’);  }  });  MyBook.type = “book”; // 静态成员可简单赋值。  // 下面是使用这个类  var book = new MyBook();  Book.buy(); // 输出 ‘买了’  alert(MyBook.type); // 输出 “book” |

在 Javascript 中， 一切都是变量， 类本身也是变量。如

|  |
| --- |
| var a = Date;  var d = new a(); |

在 Javascript 中， 对象和函数的地位一样， 故类也是函数。这肯能导致误解， 因此 J+ 把类和函数分开。

类名大写，函数小写;

一个类在使用时需要加 new ，而函数不加。 如 new Date(); -> Date 类 myFun(); -> myFun 函数。

### 构造函数

类初始化所执行的函数叫构造函数，即在 new MyBook() 时被执行的函数便是 MyBook 类的构造函数。J+规定类的成员中名字是 constructor 的成员就是构造函数，如果没有定义构造函数，将会用默认的构造函数创建。

下面代码演示了如何利用构造函数:

|  |
| --- |
| var MyBook = new Class({ // new 可以不写  version: 2,    // 构造函数  constructor: function(){  alert( this.version ); // 输出当前实例的 version  },    buy: function(){  alert(‘买了’);  }  });  // 下面是使用这个类  var book = new MyBook(); // 输出 2 |

### 继承

比如有一个类 A， 类 B， 如果 B 继承于 A，那B 可拥有 A 的全部动态成员，其中 B 叫 A 的子类、 A 叫 B 的父类。子类可以覆盖父类的同名的动态成员。

所有的类都继承 Object 类。Object 的动态成员有 2 个: toString() 和 valueOf() ， 因此这个成员也是所有类的动态成员 -- 因为全部变量(类实例)都可使用 .toString() 。

子类如果不存在自己的构造函数，则默认调用父类的构造函数。

下面代码演示如何继承已有类:

|  |
| --- |
| var MyBook = new Class({ // new 可以不写  version: 2,    // 构造函数  constructor: function(){    },    buy: function(){  alert(‘买了’);  }  });  var MyNotePad = MyBook.extend({  buy: function(){  alert(this.version); // 输出当前 version， 这个属性继承于 MyBook 类。  }  });  // 下面是使用这个类  var book = new MyNotePad ();  book.buy(); // 输出 2 |

如果需在子类调用父类的被覆盖函数(如上例，子类的buy 需调用父类的 buy )，可使用 this.baseCall(‘buy’, arguments); 或 this.baseCall(‘buy’, 参数1, 参数2…)

如果子类需调用父类的构造函数，使用: this.base(参数1，参数2,…)

### 事件

任何类都有事件机制。

|  |
| --- |
| var Cup = new Class();  // 此时 Cup 有 on, un 等事件函数。  var cup = new Cup();  cup.on(“事件名”, function(){}) // 绑定事件  cup.un(“事件名”, function(){}) // 去除事件  cup.one(“事件名”, function(){}) // 绑定单次事件(仅运行1次)  cup.trigger(“事件名”) // 触发此事件 |

当然，也可以自定义事件。一个事件分4部分:

1. add 绑定事件
2. remove 删除事件
3. trigger 执行事件
4. setup 安装事件

|  |
| --- |
| var Cup = new Class();  Cup. addEvents({    bought: { // bought 是事件名  // 下面的这些成员不一定都要写，因为有默认值  add: function(elem, type, fn){ // elem 是绑定事件的目标， type 是绑定的事件名， fn 是绑定的函数  alert(“事件被绑定”);  },  remove: function(elem, type, fn){  alert(“事件被移除”);  },  trigger: function(e){ // e 是由 createEventArgs 创建的参数， this 指向绑定事件目标。  alert(“事件被触发”);  }  }  });  // 此时 Cup 有 on, un 等事件函数。  var cup = new Cup();  cup.on(“bought”, function(){}) // 输出 事件被绑定  cup.un(“bought”, function(){}) // 输出 事件被移除  cup.trigger(“bought”) // 输出 事件参数生成 和 事件被触发 |

### 接口

如果一个变量仅仅为了扩充类而存在，这个对象称为接口。任何实现这个接口的类都拥有其功能。约定接口名以 I 开头。

### 调试API

#### 调试输出 (trace)

调试一个 Javascript 代码是辛苦的(相对高级编程语言) ，很多开发者还在使用 alert 输出变量来调试。在 PyJs 内核中， 提供 trace 函数来调试输出。比如

|  |
| --- |
| trace("一个变量"); // 输出变量 |

这个函数会先检测当前有无调试平台，如 Firebug ， 如存在则直接调用这些平台的 API 。否则使用 trace.alert 输出 (默认trace.alert 直接使用 alert )。

trace 函数还有字符串格式化的功能:

|  |
| --- |
| trace("{0} + {1}", 2, 3); // 输出 2 + 3  trace("({x}, {y})", {x:1, y: 1}); // 输出 (1, 1); |

trace函数在编译后将会被删除，因此程序功能设计不能依赖trace函数。

下面具体说明:

|  |  |
| --- | --- |
| **函数名** | **说明** |
| trace(“x”); | 完全可以替换大多数jser用的alert和摩登jser的console（因为有时浏览器不支持console） |
| trace(“{0}可以被替换”,4); | 格式化输出 |
| trace.log(变量A,变量B,变量C…); | 支持同时多变量的输出 |
| trace.info(“提示”); | 输出 提示 |
| trace.error(“错误”); | 输出 错误 |
| trace.warm(“警告”); | 输出 警告 |
| trace.dir({a:1}); | 遍历一个对象。 |
| trace.clear(); | 清除控制台，无控制台时无用 |
| trace.ifNot(条件,语句); | 如果条件不满足（!条件 === true），输出语句 |
| trace.tryExecute(函数); | 试着运行一个函数，返回错误信息。 |
| trace.runTime(函数,[参数,运行次数（默认1000）]); | 返回一个函数运行时间 |
| trace.inspect({}); | 遍历一个对象，并作为字符串形式返回。 |



可以试试： trace.dir(trace); 将输出trace的全部方法

#### 调试确认 (assert)

assert 是调试时用于确认变量是否符合要求的函数。

|  |  |
| --- | --- |
| **函数名** | **说明** |
| assert(a>1); | 确保a>1 否则产生错误。 |
| assert(a>1,”{0}必须大于一定值”,a); | 可以输出错误信息。 |
| assert.notNull(不为null,”信息”); | 确保不空 |
| assert.notEmpty(不为空,”信息”); | 确保不空 |
| assert.notStatic(不为值变量,”信息”); | 保证变量是静态的 |
| assert.instanceOf(“”, ”string”, ”类型”); | 保证变量属于一个类型 |
| assert.instanceOf (“”, [”string”,”object”], ”类型”); | 保证变量属于几个类型的一个 |

### 系统API

J+ 中的函数都是以类成员的方式提供的。

#### 类

函数 Class为创建一个类，语法：

|  |
| --- |
| var Cup = new Class(参数);  //new 可省略 |

此时的参数可以为：

1. 类的构造函数

2. 类的动态成员

3. 空

#### 名字空间

名字空间帮助避免不同代码之间的冲突。

|  |
| --- |
| JPlus.namespace("Q.W.E", {  T: 2  // 可同时加其它内容  }); |

这句将产生 1 个对象:

Q = { W : { E : { T : 2}} }

#### 类对象

任何类都含这些成员:

|  |  |
| --- | --- |
| **函数名** | **说明** |
| JPlus.Object. extend (成员) | 返回继承当前的类的类。 |
| JPlus.Object. implement(成员) | 扩冲当前类的动态成员。 |
| JPlus.Object. implementIf(成员) | 如果当前类的动态成员不存在，扩充。 |
| JPlus.Object. addEvents (事件) | 为当前类增加事件支持。 |

用法说明：

|  |
| --- |
| var Cup = new Class();  Cup. implement({  size : 9  });  // 此时 Cup 有 size 的动态成员。 |

#### 和变量自身有关的方法 Object

|  |  |
| --- | --- |
| **函数名** | **说明** |
| Object.clone(对象) | 复制对象(深拷贝)。 |
| Object.type(变量) | 返回变量类型。肯能返回 string number boolean undefined null function element class array date regexp object |
| Object.extend(对象，内容) | 扩展对象。(相当于常见的apply函数)  如 a ={}; b = {c:2}; Object.extend(a, b); // -> a = {c:2}; |
| Object.extendIf(对象，内容) | 如果不存在则扩展对象。 |
| Object.each(对象，函数) | 遍历对象。 |
| Object.update(对象，函数，目标) | 用函数更新数组。 这个函数可以作为 类似map的函数的替代品。当没有指定目标时，将更新到自己。 |
| Object.isObject(变量) | 判断是否是引用变量。(除了null且 typeof 结果等于“object”) |
| Object.isArray(变量) | 判断数组 |
| Object.isFunction(变量) | 判断函数 |
| Object.value(字符串) | 不用eval 获得如 ”obj.a” 的值。 |

#### 和数组有关的方法 Array

|  |  |
| --- | --- |
| **函数名** | **说明** |
| Array.create(迭代对象，开始数，长度) | 生成一个数组，一般参数是 arguments 。这里的迭代对象即含length成员的对象。 |
| Array.plain([数组]) | 连接多个数组（部分浏览器原生支持） |
| [].indexOf(内容) | 返回内容位置。（部分浏览器原生支持） |
| [].remove(内容) | 删除内容。 |
| [].contains (内容) | 检查是否存在指定参数。 |
| [].filter(函数) | 根据函数过滤数组的内容。 |
| [].each (函数) | 对每个元素执行。支持 false 停止。 |
| [].forEach (函数) | 对每个元素执行。（部分浏览器原生支持） |
| [].include (内容) | 包含一个内容，不重复。 |
| [].select (函数) | 选择数组中指定元素，返回一个数组副本。 |
| [].unique() | 剔除数组中重复元素。 |

#### 和函数有关的方法 Function

|  |  |
| --- | --- |
| **函数名** | **说明** |
| Function.empty | 空函数。 |
| Function. from | 返回返回自己的函数。 |
| Function.bind | 返回绑定域的函数。 |
| Function.returnTrue | 始终得到 true 的函数。 |
| Function.returnFalse | 始终得到 false 的函数。 |

#### 和字符有关的方法 String

|  |  |
| --- | --- |
| **函数名** | **说明** |
| String.format(格式化字符串,格式化变量) | String.format(“{0} == {1}”, 1 , “3”); // 1 == 3  String.format(“{a}为变量”,{a:”aa”}); // aa为变量 |
| String.map(字符串， 执行， 目标) | 根据字符串分隔后的值对函数执行存入目标 |
| String.toArray(字符串, 分隔符) | 分隔字符串。 |
| "".trim() | 删除收尾空白字符串。 |
| "".capitalize () | 首字母大写。 |
| "".toCamelCase () | 返回骆驼规则字符串。 |

#### 和浏览器有关的方法 navigator

|  |  |
| --- | --- |
| **函数名** | **说明** |
| navigator. browser | 浏览器。 |
| navigator. version | 浏览器版本。 |
| navigator.isQuirks | 如果是IE6/7，返回 true |
| navigator.isIE | 如果是这个浏览器， 返回 true |
| navigator.isIE6 | 如果是这个浏览器， 返回 true |
| navigator.isIE7 | 如果是这个浏览器， 返回 true |
| navigator.isIE8 | 如果是这个浏览器， 返回 true |
| navigator.isIE9 | 如果是这个浏览器， 返回 true |
| navigator.isFirefox | 如果是这个浏览器， 返回 true |
| navigator.isFirefox2 | 如果是这个浏览器， 返回 true |
| navigator.isFirefox3 | 如果是这个浏览器， 返回 true |
| navigator.isOpera | 如果是这个浏览器， 返回 true |
| navigator.isOpera9 | 如果是这个浏览器， 返回 true |
| navigator.isOpera10 | 如果是这个浏览器， 返回 true |
| navigator.isChrome | 如果是这个浏览器， 返回 true |
| navigator.isSafari | 如果是这个浏览器， 返回 true |
| navigator.isUnkown | 如果是这个浏览器， 返回 true |

#### 和请求有关的方法 XMLHttpRequest

|  |  |
| --- | --- |
| **函数名** | **说明** |
| XMLHttpRequest 函数 | 创建一个 XMLHttpRequest 对象。(全部浏览器都用此函数) |

#### 其它函数 document 和脚本本身有关的方法 JPlus

|  |  |
| --- | --- |
| **函数名** | **说明** |
| document.id(id字符串或节点) | 返回节点。 |
| Document.getDom() | 获取跟节点。 |
| document.addEventListener(类型，事件) | 增加元素监听器。 |
| document.removeEventListener (类型，事件) | 移除元素监听器。 |
| Py.$(id字符串或 节点) | 根据id或节点返回 Element 对象 |
| Py.ready(函数) | 指明一个函数在页面载入时被使用。 |
| Py.eval(脚本) | 运行一个脚本，全局。 |
| Py.loadText(地址，回调) | 同步载入文件文字。 |
| Py.addCss(样式) | 添加css。 |
| Py.addEventListener(类型，事件) | 增加元素监听器。 |
| Py.removeEventListener(类型，事件) | 移除元素监听器。 |
| Py.rootPath | 获取或设置当前py.js所在文件夹地址 (这是using载入用的跟位置) |
| Py.data(变量，数据名) | 获取某个对象的数据。 |
| Py.dataIf(变量，数据名) | 如果存在获取数据。 |
| Py.setData(变量，数据名， 内容) | 设置对象的数据。 |
| Class.extend(成员) | 扩展类。 |
| Class.addCallback(变量，名字，函数) | 增加类成员回调。 |
| Py.Native(函数) | 本地化类。 |
| Py.Class(成员) | 创建类。 |
| JPlus.checkStatusCode(请求实例) 判断一个实例状态是否正常。 | 连接多个字符串（部分浏览器原生支持）。这个函数比加号效率高。 |
| Py.namespace(名字空间) | 声明名字空间。 |
| Py.using(名字空间) | 载入资源。 |

第五章 节点操作

本章函数全在 core/system/dom/element.js 文件提供。

第一节 事件

|  |  |
| --- | --- |
| **函数名** | **说明** |
| elem.on(“click”, log); | 绑定事件 |
| elem.un(“click”); | 去除事件 |
| elem.trigger (“click”); | 触发此事件 |
| elem.one(“click”, log); | 绑定只执行1个的事件 |

第二节 属性

|  |  |
| --- | --- |
| **函数名** | **说明** |
| a.setStyle('font-size', 14); | 设置元素 |
| a.getStyle('left'); | 获取样式 |
| a.setAttr('innerText', '12121'); | 设置属性 |
| a.getAttr('innerText'); | 获取属性 |
| a.set({'innerHTML': 'cv'}); | 快速设置元素各个属性。 |
| a.hasClass('light'); | 检查是否拥有某个类。 |
| a.addClass('light'); | 添加指定的类名。 |
| a.removeClass('light'); | 删除全部或者指定的类。 |
| a.toggleClass('light', switch); | switch参数为true则加上对应的class，否则就删除。 |
| a.getText(); | 获取内容。 |
| a.setText('sssas'); | 设置内容。 |
| a.setOpacity(0.8); | 获取透明度。 透明度 0 - 1 |
| a.getOpacity(); | 设置透明度。透明度 0 - 1 |
| a.show(); | 显示元素。 |
| a.hide(); | 隐藏元素。 |
| a.animate({height: 2}); | 渐变。 |
| a.setUnselectable(); | 设置元素无法选。 |
| a.bringToFront(被遮罩的节点); | 将元素显示在最前。 |
| a.toggle(); | 切换显示。 |

第三节 坐标

|  |  |
| --- | --- |
| **函数名** | **说明** |
| a.setScroll(2,4); | 设置滚动条。 |
| a.setSize(1,3); | 设置大小。 |
| a.getSize(); | 获取大小。大小是 {x:0, y:0} |
| a.getOffsetParent(); | 获取偏移父元素。 |
| a.getOffsets(); | 获取偏移。 |
| a.getPosition(); | 获取绝对位置。 |
| a.setPosition(2,4); | 设置绝对位置。 |
| a.getScroll(); | 获取滚动条。 |
| a.getScrollSize(); | 获取滚动范围。 |
| document.setScroll(2,4); | 设置滚动条。 |
| document.setSize(1,3); | 设置大小。 |
| document.getOffsetParent(); | 获取偏移父元素。 |
| document.getOffsets(); | 获取偏移。 |
| document.getPosition() | 获取绝对位置。 |
| document.getWindowSize() | 获取浏览器窗口的大小。 |
| Document.setWindowSize(200, 400) | 设置浏览器窗口的大小。 |
| a.getOuterSize(); | 获取外围大小。 |
| a.getOffset(); | 获取左上偏移。 |
| a.getHeight(); | 获取当前高。 |
| a.getWidth(); | 获取当前宽。 |
| a.setOffset({x: 3, y: 4}); | 设置左上偏移。 |
| a.getHeight(6); | 设置当前高。 |
| a.getWidth(5); | 设置当前宽。 |

第四节 节点

|  |  |
| --- | --- |
| **函数名** | **说明** |
| a.remove(ab); | 删除节点或自己。 |
| a.empty(); | 删除节点全部内容。 |
| a.dispose(); | 释放节点。 |
| a.getElementsByClassName('d'); | 获取指定类名的。 |
| a.getElementsByAttribute('id', 'ab'); | 获取指定属性名的。 |
| 1. insert ('beforeEnd', 'asd');   a.insert ('beforeEnd', 节点); | 插入一个节点，位置可以为: beforeEnd AfterEnd beforeBegin afterBegin |
| a.append ('asdasdasdasda');  a.append (节点); | 末尾添加节点。 |
| a.clone(); | 克隆节点。 |
| a.get('previous'); | 获取指定节点。 |
| a.get('next'); | 获取指定节点。 |
| a.get('first'); | 获取指定节点。 |
| a.get('last'); | 获取指定节点。 |
| a.get('child', 0); | 获取指定节点。 |
| a.get('parent'); | 获取指定节点。 |
| a.get('siblings'); | 获取指定节点。 |
| a.get('previouses'); | 获取指定节点。 |
| a.get('nexts'); | 获取指定节点。 |
| a.get('parents'); | 获取指定节点。 |
| a.getChildren(); | 获取指定节点。 |
| aa.getChild('div'); | 获取指定节点。 |
| a.wrapWith("<i></i>"); | 用指定的节点包装节点。 |
| aa.replaceWith(ab); | 用指定的节点更新节点。 |
| a.renderTo(document.body); | 节点插入到指定节点。 |
| a.contains(ab); | 判断节点是否包含指定节点。 |
| a.find(“.a”); | 执行简单选择器，返回第一个节点。 |
| a.findAll(“.a”); | 执行简单选择器，返回列表 |

第六节 获取节点

|  |  |
| --- | --- |
| **函数名** | **说明** |
| document.getElement | 浏览器。 |
| Py.$ | 获取节点。 |
| document.getElements | 获取多个节点。 |

第六章 Ajax

本章函数全在 core/system/ajax/ajax.js 文件提供。

|  |  |
| --- | --- |
| **函数名** | **说明** |
| Ajax.get | 发送 get 请求。 |
| Ajax.post | 发送 post 请求。 |
| Ajax.prototype.send | 发送 send 请求。 |

### 文章导读

Javascript始终是一个易学的语言，但为不同的浏览器而重复写同一个功能的代码却使人烦恼。因此Javascript比其它计算机语言更需要框架。使用一个框架意味着肯能要多很多冗余代码，且效率不高，而PyJs就为解决这些问题而产生。比如库里的组件可以被单独使用，而不需要额外地引入其它内容。

要进入 PyJs ，必须做以下准备:

1. 从本书一开始处的链接下载 PyJs 最新版。

2. 准备用来测试的常用浏览器及调试程序。如 Firebug， Py.Debug in Javascript

3. 准备好 Javascript 开发工具。如 Aptana 或 Dreamweaver 。

4. 挪出时间。做任何事都需要一心一意。

下载的文件中，内容说明:

demo/ 演示文件夹，存在用于演示效果的一些例子

core/ 全部源码的编译之后的代码

source/ 全部源码

help/ 帮助文件

license/ 开源协议

tools/ 工具

第七章 控件

本章函数全在 core/system/dom/element.js 文件提供。

控件即组件。一个控件就是一个功能体，比如按钮控件，文本框控件。PyJs内一个控件就是一个继承 Py.Control 的类。

一个控件从本质上说就是封装一个 HTML，CSS，Javascript的对象。比如按钮控件，内部同样是用<input />创建，但控件封装操作这个元素。

所有控件使用同一个控件周期:

比如一个按钮控件 Button , 如下方法使用:

|  |
| --- |
| var button = new Button(); // 内部会创建和初始化  button.renderTo(document.body); // 显示 |

如果希望自定义控件，则应继承Control并重写上面四个函数:

|  |
| --- |
| var MyContrpl = Contol.extend({    // 默认的 create() 函数会使用 this.tpl 创建节点。  tpl: ‘<input type=”button” />’,  // 为方便控件自动检索生成，加上这个属性。  xType: ‘mycontrol’,    // 初始化, 重写Control类的同名函数  init: function(){  this.dom.value = “aaa”; // this.dom 就是当前控件的节点  this.on(‘click’, function(){ // 最终是对 this.dom 绑定  alert(‘点了’);  }  }  });  // Control 类包含 Element 类所有的函数。  // 这个控件其它函数使用父类的成员。 |

Control类的成员函数和Element的成员函数相同。

Control 类一个很重要的功能就是配置扩展。

|  |
| --- |
| var button = new Button({  value: ‘内容’,  onclick: function(){  alert(‘点’);  }  });  button.renderTo(document.body); // 显示  // 可以发现此时这个按钮的内容已经是指定内容了，且点击后执行onclick设置的值。 |

如上例，传给一个控件构造函数的对象称为控件配置。一个控件也可以自己有默认配置，一个控件的 options 成员就是它的默认的配置。

控件配置中的每一项都可以为下列内容(如果有重复，按顺序最上面优先):

比如对于配置项 disabled: true

1. 如果当前控件有disabled函数: 则执行 disabled(true);
2. 如果当前控件有 setDisabled 函数: 则执行 setDisabled(true);
3. 如果当前控件有 disabled.set 函数: 则执行 disabled.set(true);
4. 如果当前控件有 set 函数: 则执行 set(“disabled”, true); 以下过程皆为 set 内部处理
5. 如果disbaled 是特殊属性: 则执行 setDisabled(true)
6. 如果 disabled 是 on 开头的: 则执行 dom.on(“disabled”, true);
7. 如果 disabled 是 CSS 属性: 则执行 setStyle(“disabled”, true);
8. 如果 disabled 是 节点属性: 则执行 dom[“disabled”] = true;
9. 最后，执行: this[“disabled”] = true;

这些配置的检测虽然复杂，但提供了强大的配置功能，使得许多控件只需一句话使用，比如:

|  |
| --- |
| new Button({  renderTo: true, // renderTo(内容)  value: ‘内容’, // dom.value = 内容;  height: 300, // setHeight(内容);  color:’white’, // setStyle  onclick: function(){ // on(‘click’, 内容);  alert(‘点’);  }  }); |