YUI3入门

尚春@meituan

我们团队的基础库正在由YUI2迁移到YUI3。YUI3是一次革命，加入了很多新的特性，很多东西我们要重新认识，重新学习。这次分享，我就自己对YUI3的一些认识做一下分享。经验不够，欢迎随时打断，随意提问。

Outline

- YUI3因何而来
- 新的全局对象
- 模块，改变一切
- Loader & Combo
- Node & Event
- YUI3在美团

这次主要将以下几个话题。首先，要回答一个问题：YUI3因何而来，有果必有因，这样我们才能抓住重点。接下来，主要讲解一些YUI3的特性，第一个是新的全局对象。然后，也是最重要的核心概念——模块，我个人认为模块改变了一切，将YUI3提升到一个很高的高度，模块也是基础库的未来。然后简单介绍下Loader和Combo的作用，Node、Event这些核心模块的简单应用。最后，结合我们自身的情况，讲解下YUI3的实际应用。

YUI3 因何而来

- 时代对js框架的召唤
- 统一高效的api
- js/css文件的自动加载
- 多人协作
- 可持续开发

YUI2不够用了吗？为什么要推出YUI3呢？我觉得这几个原因是比较主要的：1，这是时代对js框架的召唤。互联网飞速发展，前端的开发复杂程度越来越高。仅仅依靠基础库屏蔽一些浏览器差异，简化一些操作远远不够，我们需要一个更高效的框架来为我们撑起一片天。YUI3应运而生；2，YUI2中大部分api都是一个静态方法，不符合DOM的语言习惯，我们需要selector，chaining。而一些基础库在这方面做的非常出色，例如jQuery。YUI3借鉴了这些优点，为我们提供了更统一高效的api；3，我们常常在为页面添加script时纠结不堪，要注意顺序，要考虑哪些文件需要哪些不需要。YUI3可以帮我们搞定这一切；4，前端的春天已经到来，原来的单兵作战变成团队协作，如何处理合作时的冲突？YUI3为我们提供了更好的解决方案：沙箱；5，现在很多功能开发周期比较长，不断有新的需求，不断需要改进或添加新功能，需要一种可持续开发方式。YUI3提供了一些这方面的支持，例如版本管理和单元测试（YUI

YUI3的演变

- 2008-8-13 YUI3 Preview 1
- 2008-12-9 YUI3 Preview 2
- 2009-6-24 YUI3 Beta 1
- 2009-9-19 YUI3 3.0.0 GA
- 2011-1-12 YUI3 3.3.0 (meituan)
- 2011-6-2 YUI3 3.4.0 Preview 1

2009年9月19日，YUI3正式版发布（GA generally available）

我们现在用的是最新的正式版本：3.3.0

YUI3的亮点

- Modules
- Auto-completing
- Sandboxing
- Selector-driven
- Chaining
- Custom Event ++

1，无可争议的，模块是最大的亮点；2，在简单做一些模块的依赖关系配置后，我们不必关心调用的模块依赖哪些，直接调用即可，YUI3会自动去计算和排序，并拿到所有模块。3，沙箱在多人协同时的作用不容小觑。后续我们会进行介绍。4，作为先进生产力，选择器已经融入到每一个角落，和getElementsByTagName，getElementsByClassName说拜拜吧。5，链式调用更符合DOM语言规范，YUI3做到了。6，强大的自定义事件，支持默认动作，冒泡和停止传播，并提供了AOP的支持。

新的全局对象YUI

	YUI2	YUI3
全局对象	YAHOO	YUI
类型	Object	Function
作用	全局命名空间	全局命名空间 沙箱工厂 注册模块 还有...

YUI3引入了新的全局对象YUI，这样保证了向前兼容。更重要的，YUI不再仅仅是一个全局命名空间，它是function类型的，而且是一个构造器，充当着沙箱工厂、注册模块的重任。YUI还有另外一个含义，就是...

日本音乐人YUI！漂亮吗？

YUI对象的简化版本

```
var YUI = function() {
    var Y = this;
    if (!(Y instanceof YUI)) {
        Y = new YUI();
    } else {
        Y._init();
    }
    return Y;
};
```

我很欣赏一句话：know why things work the way they work，就是说要知其然也要知其所以然。所以接下来会展示一些经过简化过的代码，让大家更加了解YUI对象到底是什么。YUI构造器是一个Scope-free constructor，用到的方法主要是_init

YUI对象的简化版本

```
YUI.Env = { mods:{} };
YUI.prototype._init = function() {
    var Y = this,  G_Env = YUI.Env,  Env = Y.Env;
    if (!Env) {
        Env = Y.Env = { mods:{}, _used:{}, _attached: {} };
    }
    Y.constructor = YUI;
};
```

YUI有一个静态属性Env，用来存储一些环境变量，其中比较重要的就是模块信息mods

_init方法主要用来为实例初始化实例级别的环境变量Y.Env

现在，我们这样组织代码

```
<script src="http://s1.meituan.net/js/yui/2.8.1/build/utilities/utilities.js?v=1"></script>

<script src="http://s0.meituan.net/js/core.js?v=24bdbb06"></script>

<script src="http://s1.meituan.net/js/app-deal.js?v=52f6ca90"></script>

<script src="http://s1.meituan.net/js/app-biz.js?v=cbb773ea"></script>

<script src="http://s1.meituan.net/js/w-tab.js?v=0e894e77"></script>

<script type="text/javascript">

YAHOO.util.Event.onDOMReady(function(){

    MT.app.Deal.refresh("ios5", false);
    MT.app.Deal.toggleOtherShare();
    MT.app.Deal.removeGuide();
    ...

});

</script>
```

现在，我们是这样组织代码的：在页面里插入一个个需要用到的script，时刻提心吊胆的担心是不是顺序有问题。然后在页面底部调用方法。当页面越来越复杂，开发时间越来越长，维护人员不断变化时，你知道为什么javascript中出现fuck的频率最高了吧。

<script>

- 加载 + 执行
- 顺序进行（默认）
- 共享全局环境
- 缺少相互关系信息

手动添加script标签的情况下，1，加载与执行是紧密耦合的。2，多个script之间默认是顺序执行的，使用defer属性可以延迟。3，所有的js是共享全局环境的，非常容易出现相互影响的情况，例如覆盖。4，我们必须人工的去判断每个js用到了哪些其它js的方法，需要排在哪些js之后。排序？最小复杂度是O(nlogn)。

模块，改变这一切

添加模块	调用模块

YUI.add('test', function(Y) {
  Y.sayHello = function() {
  alert('Hello, world!');
  };
});
	var Y = YUI();
	Y.use('test', function(Y) {
	  Y.sayHello(); // alert
	});
	YUI().use(function(Y) {
	  Y.sayHello(); // error
	});

YUI3引入了模块的概念，并依此进行了彻底的重构。使用模块后，这一些烦恼将会得到改变。

将所有的功能模块化，然后通过模块包含的一些依赖关系将全部模块系统的组织起来，这就是一个框架的基本思想。YUI2中的DOM、Event等等都可以模块化，我们自己开发的一些功能也可以模块化。

添加模块代码非常简单，使用YUI的add方法，注意这里不是在用实例的add方法。调用模块需要首先得到一个YUI的实例，然后使用这个实例的use方法，指明调用的模块，然后在回调函数中就可以使用调用模块的api了。还记得YUI是一个沙箱工厂吗？它的实例能且只能使用调用模块以及调用模块依赖的模块们的api，所以没有调用test模块的化，调用test模块的方法会报错。

add的简化版本

```
YUI.prototype.add = function(name, fn,  details) {
    // register on the global object YUI
    YUI.Env.mods[name] = {
        name: name,
        fn: fn,
        details: details || {}
    };
    return this;
};
YUI.add = YUI.prototype.add;
```

add方法的简化版本。通过add方法注册的模块信息保存在YUI.Env中，是全局范围的，也就是说，在一个模块注册之后，所有YUI生成的实例都可以调用该模块。add方法的第二个参数用来给YUI实例添加方法。details是一些模块信息，例如requires，use等。另外，add方法可以链式调用。

use的简化版本

```
YUI.prototype.use = function() {
    var Y = this, r = [],
        args = Array.prototype.slice.call(arguments, 0),
        callback = args[args.length - 1],
        process = function(names) {  ...  };
    if (typeof callback === 'function') { args.pop(); }
    else { callback = null; }
    process(args);  // calculate sorted full module list r
    Y._attach(r);  // add the apis to Y
    if (callback) { callback(Y); }
    return this;
};
```

YUI的use方法复杂一些，这里是一个非常简化的版本，我们只看重点。

use方法主要处理流程为：计算调用模块集合和回调函数，通过处理调用模块集合得出排序后的所有模块，然后将实例传递到这些模块的方法中，依次添加api。添加完成后执行回调函数。

几个要点：1，实例通过调用模块，将这些模块以及它们依赖的模块的api绑定在自身，然后再将自己传入回调函数中。也就是说，回调函数中的形参也就是实例自身。2，实际上，在有模块没有加载完成的情况下，需要动态加载这些模块，完成后才会调用回调函数。

Module vs <script>

- ~~加载 + 执行~~　　注册 + 执行
- ~~顺序进行（默认）~~　　异步或同步执行
- ~~共享全局环境~~　　沙箱机制
- ~~缺少相互关系信息~~　　添加模块元信息

与直接嵌入script标签相比，模块的特点有：1，注册模块与调用模块分离，两者不是紧密耦合关系。2，所有模块都加载并绑定后才会执行回调函数，多个use间是异步关系。但，当所有模块都具备的情况下，多个use又是同步关系。3，通过实例绑定调用模块的api，实现了一种简便的沙箱机制。这一机制有利于协同开发。4，在模块定义时的一些元信息将所有模块有序组织起来，形成一个系统。

use不依赖add

```
var Y = YUI({
    modules: { jquery: { path: 'vendor/js/jquery.min.js' } }
});
Y.use('jquery', function(Y) {
    $('#header').html('<strong>Hello, world!</strong>');
});
```

YUI3 是一个彻头彻尾的**Javascript框架**

use调用的模块不必是已经add的模块。我们可以为YUI提供适当的信息直接定义模块。在这个例子中，我们定义了一个名称为jquery的模块，并给出了它的路径，然后调用该模块，在加载并绑定完成后就会执行回调函数中的代码。要点：这种方式定义的模块不会给YUI的实例添加任何api，而且会共享全局环境，在下载并执行jquery文件后，$和jQuery也会成为全局对象。它的缺点是破坏了沙箱机制。
从本例可以看出，YUI3可以轻松引入其它第三房库，可以看出，YUI3是一个彻头彻尾的JS框架。

use不仅仅限于js

```
var  Y = YUI({ modules: {
    mycss: { type: 'css', path: 'www/css/my.css' },
    myjs: { path: 'www/js/my.js', requires: ['mycss'] }
}});
Y.use('myjs', function(Y) {
    // my.css file has loaded
});
```

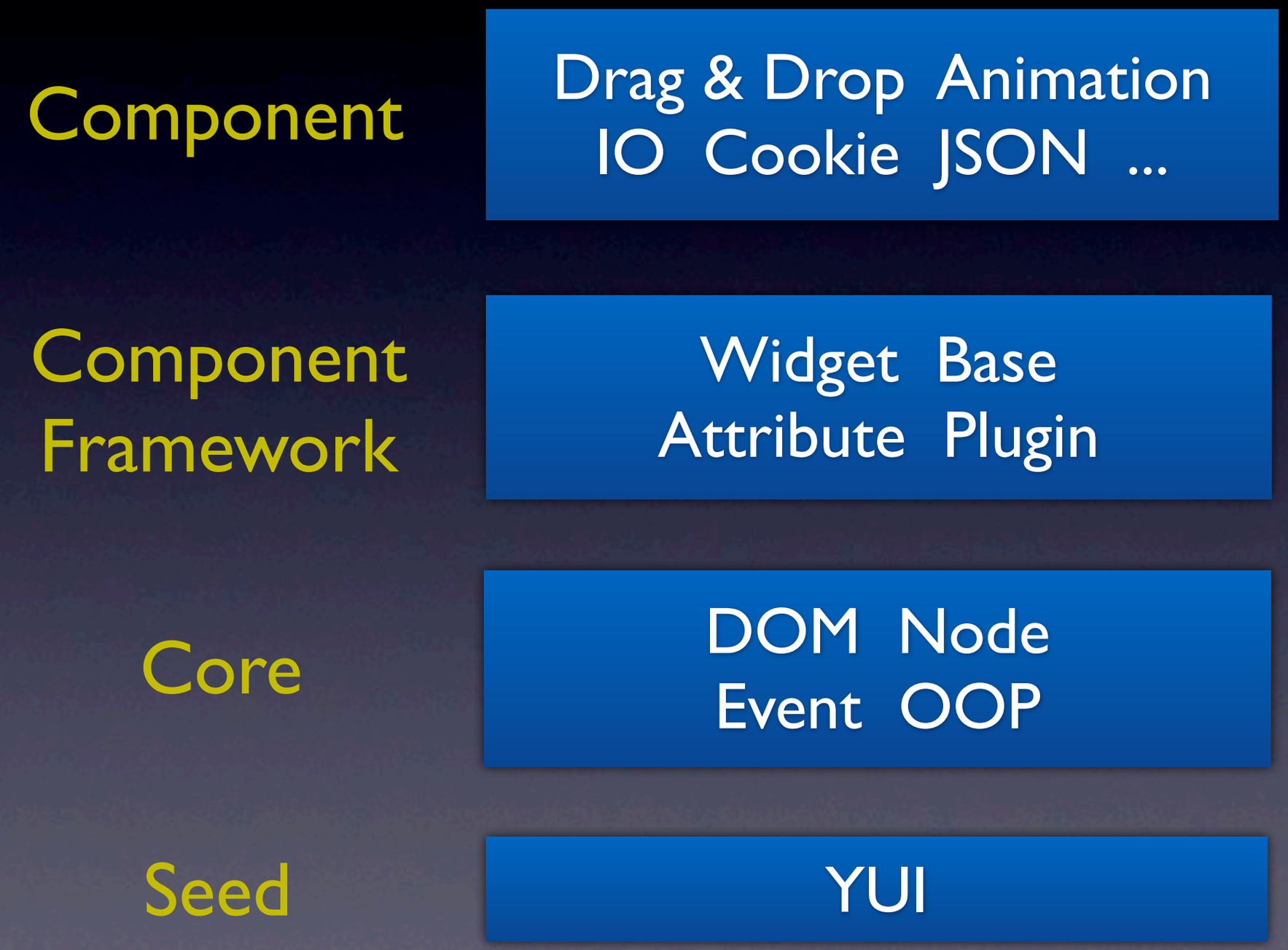
use不仅限于js文件，也可以调用css文件，真正实现按需加载。它的好处是显而易见的，我们以后的开发中可以按照模块将css拆解，一个完整的功能包含若干相关的模块，以及一系列css、image文件。

use还可以这样用

```
var  Y = YUI();
Y.use('module_A', function(Y) {
    // apis of module_A are available
    Y.use('module_B', function(Y) {
        // apis of both module_A and  module_B are available
    });
});
```

use还可以嵌套使用，重复使用。

YUI3 架构



YUI3的所有模块分为四个层级，它们构成了YUI3的架构。

1，最底层是种子文件YUI，它定义了YUI构造器以及yui模块，主要包括Lang，UA，封装的Array，Object，以及Get等等。

2，第二层为核心工具层，主要是一些工具模块，例如DOM，Node，Event

3，第三层为组件框架层，主要是为构建组件提供丰富灵活的机制

4，第四层为组件层，包括了IO，JSON，Animation等重要模块

Loader & Combo

```
<script type=”text/javascript” src=”yui/build/3.3.0/yui/yui-min.js”></script>
YUI({ modules: {
    module_A: { path: ‘www/js/module_A.js’, requires: [‘node’] },
    module_B: { path: ‘www/js/module_B.js’, requires: [‘module_A’] }
}}).use(‘module_B’, function(Y) {
    do something...
});
```

Loader计算需要加载的模块，并按依赖顺序排列，请求加载

```
<script type="text/javascript" src="http://xxx.com/combo?3.3.0/build/oop/oop-min.js&3.3.0/build/dom/dom-base-min.js&...&3.3.0/build/node/node-min.js&3.3.0/build/event/event-base-ie-min.js&3.3.0/build/event/event-delegate-min.js"></script>
```

Combo服务器将所有js压缩合并后返回一个大文件

简单介绍下Loader和Combo。

在页面中，我们只需加入YUI的种子文件，然后在实例化YUI时传入自定义模块的元信息后就可以使用这些模块了。实际的过程是，Loader根据自己保存的YUI3所有模块元信息以及我们传入的自定义模块元信息计算出排序后的加载模块列表，然后生成一个指向combo服务器的请求动态加载这些模块。url很长很长。combo服务器会根据请求中文件的顺序，将其压缩后，有序的合并成一个大文件返回。

Node

- 核心模块之一
- 对DOM节点的封装
- 统一，DOM化的api
- 优秀的扩展性

YUI3 core发生了一点变化：Node取代了DOM。Node是对原生DOM节点的封装，提供了更加统一，DOM化的api，支持链式调用。另外，Node作为一个基础对象，具备优秀的扩展性。我们可以添加自己的方法、属性等等，非常方便。

YUI2 Dom到Node的Api映射

YAHOO.util.Dom.get(‘element_id’)	Y.one(‘#element_id’)
$D.getElementsByClassName(‘more’)	Y.all(‘.more’)
elAuthor.href = ‘meituan‘	ndAuthor.set(‘href’, ‘meituan’)
$D.getAttribute(el, ‘data-params’)	nd.getAttribute(el, ‘data-params’)
$D.getNextSibling(el)	nd.next()
$D.addClass(el, ‘hover’)	nd.addClass(‘hover’)
$D.getRegion(el)	nd.get(‘region’)
el.parentNode.removeChild(el)	nd.remove()
...	...

列举一些YUI2 DOM和YUI3 Node api的映射。可以看出，YUI3 Node更加简洁、高效。

创建元素

YUI2

```
var tip = 'pelease type here...';
var elNew = document.createElement('div');
$D.addClass(elNew, 'tip');
elNew.innerHTML = tip;
var elHeader = $D.get('header');
var elSearch = $D.getElementsByClassName('search', '*', elHeader)[0];
elSearch.appendChild(elNew);
```

YUI3

```
var tip = 'pelease type here...';
var ndNew = Y.Node.create('<div class="tip">' + tip + '</div>');
Y.one('#header .search').appendChild(ndNew);
```

没感觉Node的高效？那么看看这个创建元素的例子。上面是用YUI2来实现，下面用YUI3，代码量相差一倍多。

Node与选择器

```
var nd = Y.one('#demo');
var ndFirstPar = nd.one('p');    // first paragraph in nd
var ndTip = nd.next('div.tip');   // next sibling with tag div and class tip
if (ndTip) {
    nd.ancestor('div').addClass('hidden');
};
nd.all('p').even().addClass('alt');   // add alt class to even paragraphs
if (node.test('.foo.bar')) {   // check whether nd has both foo and bar
    node.removeClass('bar');
}
```

css中的选择器十分精准，哪为何我们还要用一坨坨js苦苦的找到目标节点呢？jQuery引入选择器后被大肆追捧，而且现在一些现代浏览器也加入了querySelector和querySelectorAll两个选择器API。选择器是先进生产力，YUI3进行了深度整合。在这些例子中，我们可以看到这一点。代码更加赏心悦目。

Node与事件

Api: on / delegate / purge / simulate

```
var nd = Y.one('#demo');
// add Listener
nd.on('click', function(e) {
    this.set('innerHTML', '我被电击了， T_T');
});
// remove Listener
nd.purge(true, 'click');
```

Node提供了与事件有关的一些api。on：绑定事件， delegate：事件代理， purge：移除事件，simulate：模拟事件。这种语法更加符合DOM语言习惯， 节点成为中心。

扩展性

Api: addMethod / ATTRS / plug / unplug

```
Y.Node.addMethod('commonAncestor', function(me, other) {
    var ancestors = this.ancestors(), common = null;
    Y.one(other).ancestors.some(function(node) {
        if (ancestors.indexOf(node) > -1) {
            common = node;
            return true;
        }
    });
    return common;
});
```

Node具有优秀的扩展性。可以方便的添加属性，设置属性的存取方法，还可以添加新的方法，还可以加减插件。这里有个不错的小例子，我们给Node添加了一个寻找公共父节点的方法。注意，addMethod第二个参数function中的this指向调用这个方法的Node实例，而它的第一个参数为Node实例对应的原生节点，第二、三个参数若为Node实例的话都会被转化为原生节点。

Node Tips

- Node.getDOMNode 获得HTMLElement
- NodeList.getDOMNodes 获得HTMLElement数组
- refresh 重新获取节点数组
- setData / getData / clearData 在node上存储信息
- 少用DOM，多用Node
- 不要滥用链式调用

一些Node方面的tip。1，我们可以用Node.getDOMNode方法获得原生节点，NodeList.getDOMNodes获得原生节点的数组，这是两个静态方法。2，另外，NodeList保存的是一份静态节点，并不是实时的，所以refresh方法可能会很有用，它可以重新执行query。3，我们可以使用setData/getData/clearData在Node实例上存储信息，取代我们之前直接在原生节点上添加属性的方式。注意，这些实例不会存在原生节点上，它们的生存周期决定于它们依附的Node实例。4，我们要适应以Node为中心的一种写法，这样更加直观。5，不要滥用链式调用，要保持代码的易读性和可维护性。

Event

- 强大的自定义事件
- 一切事件都是自定义
- 简洁、DOM化的api
- 模拟事件
- AOP

YUI3 Core中还包括Event模块，它具有以下几个特点：1，自定义事件可以定义默认行为，进行事件冒泡，也可以被终止传播。2，真相是，一切事件都是自定义事件，原生的click、focus等等都被包装过。原来是DOM包含Event，现在情况被逆转。3，同Node一样，Event提供了更简洁、DOM化的api。4，增加了模拟事件的支持。5，添加了对AOP的支持，我们可以在不侵入原有事件处理流程的前提下，在合适的时机加入一些处理方法。

添加事件监听

```
var nd = Y.one('#demo');
nd.on('click', function(e) {
    this.set('innerHTML', '我被电击了， T_T');
});

// another method
Y.on('click', function(e) {
    this.set('innerHTML', '我被电击了， T_T');
}, '#demo');
```

两种添加事件监听的方法。推荐第一种。

移除事件监听

```
var nd = Y.one('#demo');
nd.on('click',  handleClick);
nd.purge();
```

```
var handle = Y.on('click',  handleClick, '#demo');
// first
handle.detach();
// second
Y.detach(handle);
// third
Y.detach('click', handleClick, '#demo');
```

移除事件的方式非常多。推荐第一种。

事件代理

```
var nd = Y.one('#demo');
nd.delegate('click', function(e) {
    this.set('innerHTML', '我被电击了， T_T');
}, 'li');
```

- 充分利用事件冒泡
- 代码优雅
- 节省内存
- 尤其适合内部元素经常更新的情况

事件代理的api也很简洁，第三个参数是被代理元素的query。回调函数中的this指向触发事件的被代理节点的Node实例。

事件代理是一种非常优秀的技术：1，它充分利用了事件冒泡机制，在更高的层级截获事件，然后可以通过事件目标得知触发事件的元素，非常适合批量节点监听。2，代码优雅，多么赏心悦目。3，节省内存，只需要绑定一次，只有一个回调函数。4，在我们的开发中，往往遇到节点内容更新的情况，例如calendar，例如autoComplete等等，都可以采用事件代理，将事件绑定在包含那些频繁更新的节点的父节点上，省时省力又省心。

键盘事件

```
var nd = Y.one('input[name="deal"]');
// press enter key
nd.on('key', function(e) {
    this.ancestor('form').submit();
}, 'down:13+ctrl');

// another method
Y.on('key', function(e) {
    this.ancestor('form').submit();
}, 'input[name="deal"]', 'down:13+ctrl');
```

Event模块提供了键盘事件的api。非常简单，与其它事件基本一致，还是在用Node的on方法，第一个参数固定为key，第二个仍为回调，第三个参数则是需要监听的key事件信息，格式为：'键盘事件:keycode[+ ctrl + alt + shift + meta]'。和绑定事件一致，还有第二种方法，不做推荐。

模拟事件

- keydown
- keypress
- keyup
- click
- dblclick
- mousedown
- mouseup
- mouseover
- mouseout
- mousemove
- blur
- change
- focus
- resize
- scroll
- select

```
Y.one('body').simulate('click', function(e) {
    this.ancestor('form').submit();
}, { shiftKey:true });
```

YUI3增加了对模拟事件的支持，js越来越强大了。

简化的自定义事件

```
// instance scope
Y.on('Foo.update', function(data) {
    do something
});
Y.fire('Foo.update', { id:nd.get('id'), data:item });

// global scope
Y.Global.on('LiveSearch.select', function(data) {
    if (data.id === nd.get('id')) { do something }
});
Y.Global.fire('LiveSearch.select', { id:nd.get('id'),
```

YUI3的自定义时间太过强大，我们这次只介绍一种简化的版本。

实例范围内的自定义事件使用Y.on/Y.fire，全局范围的使用Y.Global.on/Y.Global.fire，通过这种观察者模式，我们可以方便的进行信息的传递，非常利于模块化开发。第二种自定义事件在我们的widget中被广泛应用。

Event Tips

- NodeList.on的事件监听函数中，this指向NodeList对象
- YUI封装的focus/blur事件可以冒泡，可用事件代理
- 支持渲染周期事件available，contentready和domready
- 绑定事件推荐用nd.on，不要用Y.on

介绍一些tip：1，NodeList实例绑定事件时，回调中this指向的不是触发事件的target，而是NodeList实例。2，focus/blur事件被封装过，支持冒泡，可以在form节点上使用事件代理监听。3，YUI3支持三种渲染周期事件available，contentready和domready。4，为节点绑定事件时，推荐用nd.on，使用Y.on不够直观，而且还容易和自定义事件混淆。

YUI3 在美团

- 正在由YUI2迁移到YUI3，我们在重建新秩序
- 出现很多错误，但不妨碍我们对YUI3的追逐
- 在行进中开火
- 做事不是做菜，要等材料齐了才下锅

压轴内容：YUI3在美团。1，我们现在正在由YUI2迁移到YUI3，由于YUI3的革命性，我们需要建立一套新的秩序。2，YUI3的门槛并不低，迁移的过程中，我犯过很多错误，但这些都挡不住YUI3对我们的吸引力。3，我们是一个快公司，要学会在行进中开火，每天都有一些进步，尽快发挥战斗力。4，一种做事的思想，我们需要先开始做，然后积累经验不断改进，而不是一开始要把所有情况都考虑到，设计一个perfect的功能后再去实践，时不我待。

挑战

- 划分、组织模块
- 快速开发
- Combo
- loading策略
- 版本管理

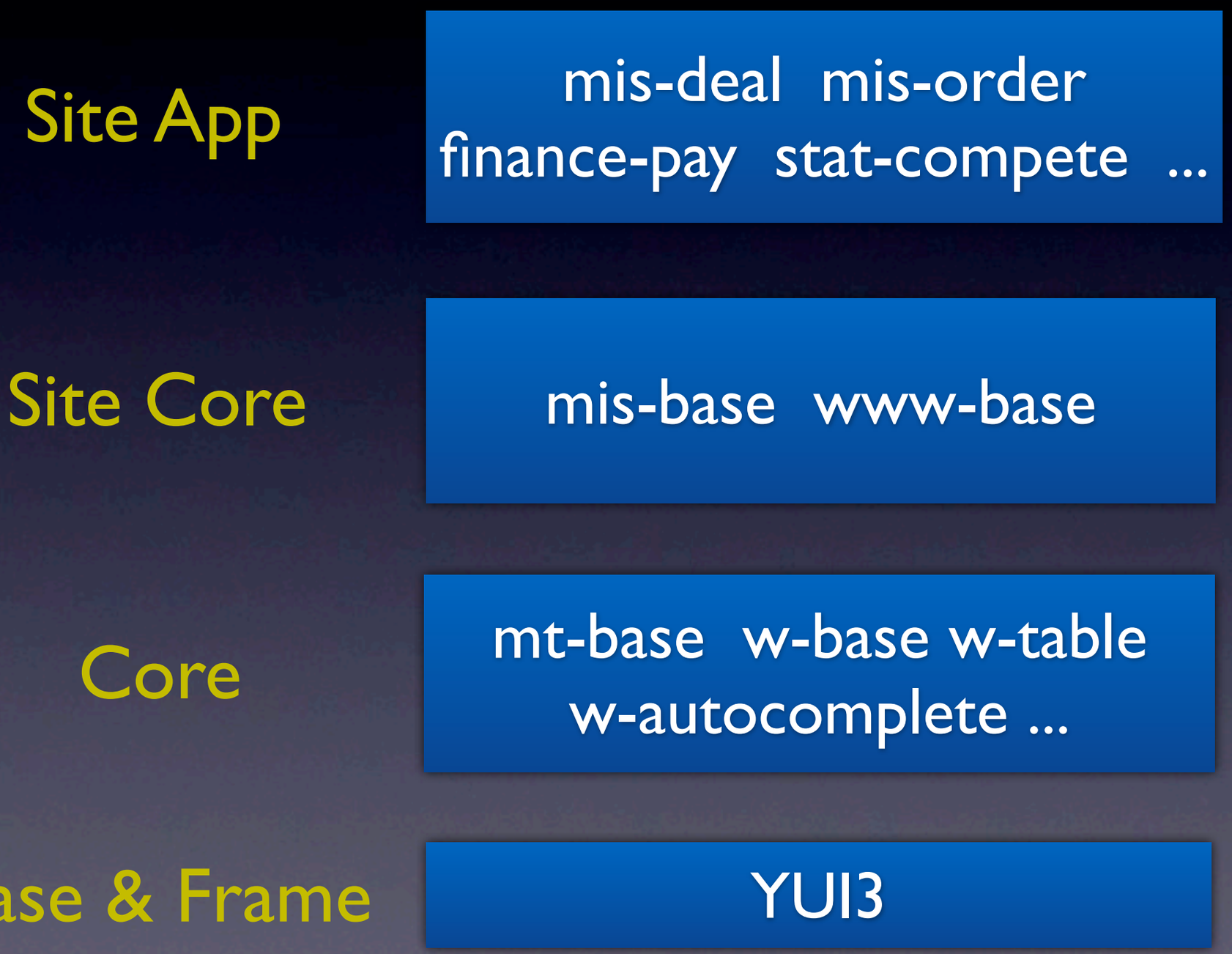
我们遇到很多挑战：1，如何将原来的功能划分为模块，如何将所有模块组织起来，这是最大的挑战。2，Combo服务的实现。yahoo的combo只能提供YUI自身模块，而且速度在国内并不占优势。我们采用了minify，进行了一些修改和配置，能够在线上线下提供combo的支持。3，模块加载策略的研究。目前我们采用的方式是：预加载通用基础模块，例如yui，node，event，io等，然后是站点级别的一些操作，例如mis系统的widget初始化、消息系统初始化等，最后是页面级别的一个特定应用入口。4，版本管理的问题。原来我们采用了query string的方式进行版本管理，现在则是用embedding的方式。

划分模块的原则

- 抽象与应用脱离
- 职责单一
- 粒度得当
- 不容忍非主流

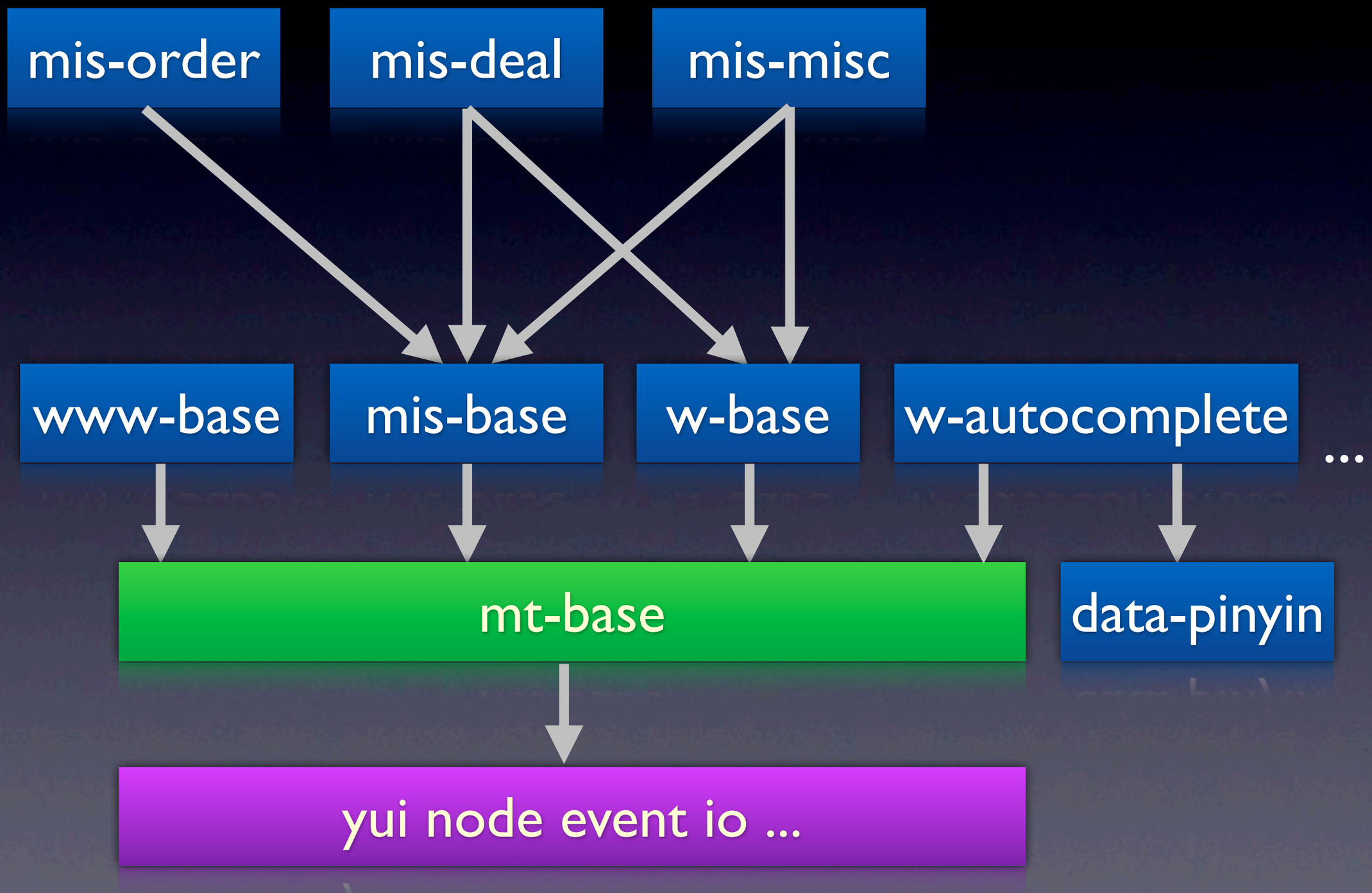
那如何划分模块呢，我总结了如下几条原则：1，抽象与应用必须脱离。更通用的功能应该放在更低的层级，应用层完全面向实际问题，在解决的过程中调用抽象出来的方法。2，职责单一。保持每个模块的职责足够简单，方便维护和可持续开发。3，粒度得当。有了combo，我们可以不必担心粒度太小，文件过多导致的速度问题。但是，从可维护的角度来考虑，粒度应该适当而不宜过小，避免海底捞针的情形出现。4，不容忍非主流。对于依赖一些额外非通用模块的功能，可以考虑独立为一个模块。

美团Javascript架构



美团的Js架构，按照抽象与应用程度，分为四层。1，最底层交给强悍的YUI3，为我们提供很跨浏览器的api和很好的框架。2，第二层是我们二次开发的一些核心方法和控件库。mt-base中包含strtotime，getRadioValue等基础方法，对Cookie的简单操作，一些封装的页面动画等。w-base，w-autocomplete等包含一些封装的控件，提供更加高效的用户体验，且便于后端人员调用。3，第三层包含美团各个分站点的一些通用模块。例如mis-base包含后台mis系统的消息系统、checkFormChanged等通用方法。这一层更加接近应用。4，最上面一层，应用模块，这些模块的方法都是用来解决实际问题的。例如mis-deal用来处理mis系统中所有deal相关页面的交互，finance-pay用来处理财务系统中的付款相关页面的交互。一些零碎的应用方法我们放在对应站点的misc模块中。

模块间关系


mis-order
mis-deal
mis-misc
www-base
mis-base
w-base
w-autocomplete
...
mt-base
data-pinyin
yui node event io ...

文件组织

core

mt.js [mt-base] w-base.js [w-base]
w-tab.js [w-tab] w-date.js [w-date] ...

www

in the near future

static

mis js

base.js [mis-base] deal.js [mis-deal]
order.js [mis-order] misc.js [mis-misc]

finance js

pay.js [finance-pay]

vendor js

jquery.min.js zeroClipboard.js ...

...

简单的封装

```
var M = window.M || {};   // new global object
if (window.YUI) {
        var YUI_META = { config };
        M._YUI = YUI;  // shield YUI
        M.add = function() { M._YUI.add.apply(M._YUI, arguments); return M; };
        M.use = function() {
            var instance = M._YUI(YUI_META);
            instance.use.apply(instance, arguments);
            return instance;
        };
}
```

为了保持向前兼容，我们定义了新的全局对象M。我们对YUI进行了简单的封装，可以使用M.add添加模块，使用M.use调用模块。请注意，每次使用M.use都会新生成一个config的实例，这个config信息主要包含一些combo服务器信息，我们自己开发模块的信息。这样loader就知道该如何计算依赖的模块了。

添加模块

```
// static/cs/js/consult.js
(function() {
M.add('cs-consult', function(Y) {
        var $MIS = Y.mt.mis, isIE = Y.UA.ie ...   // shortcut
        var CONST = 'const value' ...  // const
        Y.namespace('cs.Consult');
        Y.cs.Consult = {
            list:function(param) { ... }
        };
}, '1.0.0', { requires:M.Groups['cs']['modules']['consult'].requires });
})();
```

添加模块元信息

```
// helper/FileHelper.php
private static $groups = array(
        'core' => array(...),
        'mis' => array(...),
        'cs' => array(
            'cs-consult' => array(
                'path' =>  'cs/js/consult.js',
                'requires' => array('mis-base','w-tab'),
            ),
        ),
        ...
);
```

参考资料

- 官方文档
- YUI3 - below the surface Luke Smith
- YUI3初探 拔赤
- 从YUI2到YUI3看前端的演变 张克军

一些参考资料，感谢这些乐于分享的作者们！

Thanks

Q & A