KISSY Documentation

Release alpha

kissyteam@gmail.com

June 03, 2011

CONTENTS

CHAPTER

ONE

快速上手

这是一份介绍如何使用 KISSY 的入门教程，教你从最基本的地方开始熟悉这可爱的 KISSY ^o^
如果你是如下情况之一的开发人员，都可以通过这份文档慢慢使用起 KISSY:

1. 以前使用过 jQuery，YUI，Ext 等前端类库，但都不是很满意，想尝试个新鲜点的类库；
2. 前端开发新手，只有基本的 JS 基础，正好想学一个 JS 类库来提高自身技能；
3. 完全是个菜鸟，也没关系！学习 KISSY 会让你逐渐对前端技术产生足够的兴趣；
4. 至于前端大牛，直接进入KISSY的 源码世界 贡献你的智慧吧！

1.1 KISSY 愿景

> 小巧灵活，简洁实用，使用起来让人感觉愉悦.
>
> Keep It
>
> > Simple & Stupid, Short & Sweet, Slim & Sexy...
>
> Yeah!

下面就开始我们的 KISSY 新奇之旅吧~~

1.2 内容大纲

1.2.1 Setup

如何使用 KISSY ？

- 先将 https://github.com/kissyteam/kissy/raw/master/build/kissy.js 引入到你的页面；
- 接着，稍加 编写一些代码，就可以实现下面的效果了：

使用 KISSY 实现上面的效果，只需大约 10 行代码.
下一节将会详细介绍上面的示例，请看 Hello KISSY ~

1.2.2 Hello KISSY

在前一小节的示例中, 用到了 KISSY 常用的 DOM, Event, Anim 等功能.

完整代码如下:

```
KISSY.ready(function(S){
    var DOM = S.DOM, Event = S.Event,
        btn = DOM.get('#demo-btn');

    Event.on(btn, 'click', function() {
        DOM.attr(btn, 'disabled', true);

        S.Anim('#demo-img', 'left: 400px; opacity: 0', 2, 'easeOut', function() {
            S.Anim('#demo-txt',
                   'left: 0; opacity: 1; fontSize: 28px',
                   2, 'bounceOut').run();
        }).run();
    });
});
```

这个例子中,

1. KISSY.ready() 指在 DOM 加载完毕之后执行代码. 就像 jQuery 中的 $(document).ready().
2. S.DOM, KISSY 的 DOM 模块 [1] , 提供常用 DOM 操作, 如元素选择/遍历, 样式的获取/修改等等.
3. S.Event, KISSY 的 Event 模块 [2] , 提供事件处理功能, 如事件添加/删除, mouseenter/mouseleave 事件的支持等.
4. DOM.get(selector), 根据给出的 selector 获取符合条件的 第一个节点; 另外还有一个类似的方法叫做 DOM.query(selector) , 与前者不同的是, 得到的是 所有 符合条件的元素.
5. DOM.attr(elem, name, val), 获取/设置元素某个属性, 这里, 在动画开始前给按钮设置不可用状态.
6. S.Anim(), 提供动画效果, 通过给元素设定参数, 就可以让这个元素动态地从当前参数变化到设定的目标参数.
7. DOM.get/DOM.query 也可以直接用 S.get/S.query 来调用 , 是一样的.

DOM/Event 是最基本的功能, 掌握了这两个, 就能基本使用 KISSY 了. 接下来将介绍 Node 对象的使用.

参考链接

1.2.3 Node 初步

如果你使用过 jQuery, YUI3 或 Ext, 那对链式风格应该已有了解. 链式风格的书写让人使用更加方便, 代码看起来也更整洁.

KISSY 也提供了这种链式风格的支持.

简易 API

KISSY 提供两种使用方式:

1. S.one - 根据 css selector, 返回 Node 对象;

[1] DOM API 文档
[2] Event API 文档

2. `S.all` - 根据 css selector, 返回 `NodeList` 对象;

上面两个方法, 和 `S.get / S.query` 是遥相呼应的. 唯一的不同是, `get/query` 返回的是原生 DOM `Node/NodeList`.

而 `KISSY.Node/NodeList` 类似 jQuery 全局对象, 但只包含 DOM/Event 等方法, 我们可以这样写代码 :

```
S.one('#test').parent().next().html('<p>').on('click', function() { /* ... */ });
```

基本上和 jQuery 的语法风格是一样的, 大部分 API 也一样.

使用范例

这里我们拿先前玉伯博客 [3] 上的 `KISSY.all('.good-student')` 举个 Node 使用的小例子.

实现的效果就是, 点击按钮之后, 好多好学生都慢慢漂入 Taobao 中^o^

再来看看源码:

```
1  KISSY.ready(function(S) {
2      var DOM = S.DOM, Event = S.Event, timers = [];
3
4      S.NodeList.prototype.icanfly = function() {
5          var targetX = 600, targetY = 200,
6              maxX = 750, maxY = 350;
7
8          S.each(this, function(item, i) {
9              var x = 0, y = 0, speed = Math.random() * 80;
10             timers[i] = S.later(function() {
11                 x += Math.random() * speed * (x > maxX ? -1 : 1);
12                 y += Math.random() * speed * (y > maxY ? -1 : 1);
13                 DOM.css(item, { left: x, top: y });
14                 if(x > targetX && y > targetY && x < maxX && y < maxY) {
15                     timers[i].cancel();
16                 }
17             }, 100, true);
18         });
19     };
20
21     S.one('#go').on('click', function() {
22         S.each(timers, function(timer) { timer.cancel() });
23         S.all('.good-student').appendTo('#taobao').icanfly();
24     });
25 })
```

在这个小例子中, 先从 21 行开始看:

1. `S.one('#go').on('click', function(){});`, 选择 id 为 go 的元素, 即 button, 然后绑定点击事件.
2. `S.all('.good-student').appendTo('#taobao').icanfly();`, 获取所有 class 为 good-student 的元素, 即那些所有蓝色背景的小框, 然后 `appendTo` 到 id 为 taobao 的容器中, 最后执行第 4 行定义的 `icanfly` 动作.
3. `S.NodeList.prototype.icanfly`, 为 `NodeList` 添加一个 `icanfly` 方法, 再给 `NodeList` 中每个对象设置一个随机运动速度的定时器 `timer`, 然后定时器不断修改该对象的位置, 到达目标区域时清楚定时器.
4. 另外, 原生 `DOMNode` 和 Node 对象的相互转换, 可以使用 `new S.Node(anElment)` 将 `DOMNode` 转换成 `Node` 对象; 使用 `node.getDOMNode()` 获得对应的 `DOMNode`; 对于 `NodeList` 也有对应的方法, 移步见 [4].

[3] Join Taobao 例子
[4] Node API 文档

使用 Node 可以让你一直 . 下去, 只要你愿意!

好了, 关于 Node 就告一段落, 下面会讲述目前 Web 站点中经常被使用的技术 -- Ajax 异步请求!

参考链接

1.2.4 Ajax

KISSY 中提供了 Ajax 异步请求功能, 下面仅介绍最常用的 getScript 方法, 其他 ajax 模块中的方法, 如 get, post 等 , 请参考 API 文档 [5] .

getScript 异步请求成功后, 可以执行请求回来的数据, 与 jQuery $.getScript 是一致的.

这种请求方式, 我们一般称为 JSONP, 是最常用的跨域请求方式. 关于跨域可参见 [6] .

使用 Ajax

在日常 Web 应用中, 经常用到异步请求, 比如, 异步加载一些评论信息, 异步加载图片列表等, 这样可以减少页面初始加载时的请求, 减轻服务器压力, 也加快主页面的初始打开速度.

下面介绍个如何通过 getScript 方法去请求 flickr 上提供的图片数据.

先见示例:

再来看代码:

```
KISSY.ready(function(S){
    var API = 'http://api.flickr.com/services/rest/?'
        params = {
            'method': 'flickr.favorites.getPublicList',
            'api_key': '5d93c2e473e39e9307e86d4a01381266',
            'user_id': '26211501@N07',
            'per_page': 10,
            'format': 'json',
            'jsoncallback': 'getFavorites'
        },
        photoList = S.one('#photo-list');

    S.one('#fetch-btn').on('click', function() {
        S.one(this).attr('disabled', true);
        photoList.addClass('loading');
        S.getScript(API + S.param(params));
    });

    window.getFavorites = function(data) {
        var html = 'Fetch photo failed, pls try again!';

        if (data.stat === 'ok') {
            html = '';
            S.each(data.photos.photo, function(item, i){
                html += '<img src="http://farm' + item.farm + '.static.flickr.com/'
                        + item.server + '/' + item.id + '_' + item.secret + '_t.jpg" />';
            });
        }
        photoList.removeClass('loading').html(html);
```

[5] Ajax API 文档
[6] JSON 介绍

```
30      }
31  });
```

稍加解释下:

1. 当点击按钮时, 组装 API 参数后, 发送请求. ps: 这里各个参数可以在 flickr API 文档中找到 [7], 最后的 jsoncallback 指定回调函数的名字.
2. 组装参数时, 使用到了 `S.param` 方法, 是将对象 data 转换为 URL 中的参数字符串, 且是经过 encodeURI 编码的.
3. 在回调函数中, 获取传入的 json 数据, 稍加拼装就可以使用啦~
4. 最后的 `window.getFavorites` 是将方法 `getFavorites` 暴露给全局, 因为当请求后调用的就是全局范围内的 `getFavorites` 函数, 如果是对象中的方法, 同样可以将该对象暴露给全局后再调用其方法.

在简单介绍完 `S.getScript` 之后, 下面将讲述 KISSY 提供的动画支持 Anim ...

参考链接

1.2.5 Anim

关于 JS 中的动画, 让我映象最为深刻的就是 jQuery 首页上最经典的例子:

```
$("p.neat").addClass("ohmy").show("slow");
```

相当简单的一行代码就可以出现元素的渐隐效果, 从那时她就深深吸引了当初只知道 flash 能做动画的我.

现在 KISSY.Anim 保持了同样精简高效的风格, 很让我心动.

S.Anim

KISSY 提供的动画特效支持主要由三个子模块组成 [8] :

1. `anim`, 动画的基础模块
2. `anim-easing`, 提供 easeIn/Out, elasticIn/Out, backIn/Out, bounceIn/Out 等平滑函数
3. `anim-node-plugin`, 让 `Node/NodeList` 直接支持动画调用

核心动画函数是 `S.Anim(elem, props, duration, easing, callback)` , 5 个参数分别为:

1. `elem`: 指定动画的目标元素
2. `props`: 动画属性, 可以是字符串或者普通对象
3. `duration`: 动画时长, 以秒为单位
4. `easing`: 动画平滑函数, 可以取 [9], 不同的函数呈现不同的变化效果
5. `callback`: 动画完毕后的回调函数

其实, 这个函数很简单, 在该文档的第一节就已经使用到了最基本的功能. 更多类似的小例子可见 [10].

动画的原理都是一样的, 但要实现一些很酷的动画效果, 首先得分析出动画每个部分的状态, 再把它们有顺序的组合起来.

[7] Flickr API
[9] 平滑函数
[10] 动画示例

改进版图片抓取

除了直接调用 S.Anim 实现动画，还可以通过 DOM 或 Node 中的 show/hide/toggle, fadeIn/fadeOut 等方法来间接调用动画.

在上一节的图片获取的例子中, 点击按钮之后, 图片一下子的跑出来了, 感觉不是很好.现在, 我们利用动画组件, 让图片完全加载完毕后逐渐显示出来:

每张图片获取之后, 先不显示出来, 等图片加载完成之后, 调用 S.fadeIn 渐进显示, 部分代码:

```
photoList.html(html).all('img').each(function(img) {
    img.on('load', function() {
        if(loading) {
            photoList.removeClass('loading');
            loading = false;
        }
        img.fadeIn(3);
    });
});
```

关于动画, 先介绍到这里. 其他更炫的效果, 等待你的想象与实现!

下一节, 将会介绍 KISSY 的第一个功能强大, 且在淘宝上经常能够看到的组件 -- Switchable, 并会介绍 KISSY 组件 的组织方式...

参考链接

1.2.6 Widgets

KISSY 提供种类丰富的 UI 组件, 下面介绍下强大的 Switchable.

Switchable

Switchable 是最基本的切换组件, 已有四种扩展组件 S.Tabs , S.Slide , S.Carousel , S.Accordin, 能满足大多数应用需求.

Slide 实例

Slide 效果就是 Taobao 上最最经常见的多张图片切换效果, 如:

Slide 效果, 简简单单的使用 S.Slide(selector, cfg) 即可, 如:

```
KISSY.use('switchable', function(S) {
    S.Slide('#demo1', {
        effect: 'scrolly',       //    ,  : 'scrollx', 'scrolly', 'fade'
        easing: 'easeOutStrong'  //
    });
});
```

Config 选项

Switchable 类组件, 提供了丰富的配置选项, 详见 API 文档 [11]

[11] Switchable API 文档

就是这么简单~~

Switchable 组件先介绍到这里. 更多关于 Switchable 相关的例子见 [12]

下一节将介绍, 如果你需要自己写个组件, 该注意什么? -- 自定义组件 ...

参考链接

1.2.7 自定义组件

为何要以组件形式

将 JS 代码组件化的好处:

1. 提高代码复用
2. 模块间保持独立, 不会导致多个开发人员合作时产生的冲突
3. 封装性好, 只提供 API 接口给外部调用

KISSY 中, 通过 `add( name, fn )` 方法来添加新的模块. 在 KISSY 内部, 代码也是这么组织的.

下面通过个小例子来说明如何开发自定义组件.

组件开发示例

在淘宝交易结束前有个评价环节, 里面有一个星星打分功能, 如下示例, 在这里就把这个打分做成一个小 KISSY 组件.

完整代码: startscore.js

一些说明:

1. 首先, 想好组件的名字, 见名知意, 模块名字统一小写, 而暴露给外部的组件名称使用单词首字母大写, 如 `StarRating`;
2. 通过 `KISSY.add('starrating', function(S){ });` 加入新模块到 KISSY 中, 这里也可以使用 `KISSY.app('XXX');` 创建特定的应用, 然后用 `XXX.add('starrating', function(S){});` 给特定应用 XXX 添加一个模块;
3. 接下来是声明一些模块内的公共变量, 像 `S.DOM, S.Event` 都会用到, 另外一些如组件自己的 class 钩子;
4. 默认的配置信息, `defaultConfig`, 提供了使用者如果没有设置时的默认值;
5. 通过 `S.augment(StarRating, { });` 添加属性及方法, 每个方法在注释中写明含义, 入口参数及其类型. 另外, 开发者需要想好哪些属性/方法需要对外提供及命名方式如何等. 在这个例子中, 只添加了 `_init` 私有方法, 用来构建所需 DOM, 绑定事件;
6. 最后, 在使用时只需要创建一个对象即可, 如, `new S.StarRating('#J_Rating', config);`

罗罗嗦嗦这么一大堆后, 不知道你是否觉得简单? 非常推荐基于 KISSY 尝试去实现一个组件, 一切都很简单的^o^

注意: 上面的 StarScore 组件仅是示范，实际应用中，会更复杂些.

下一节将介绍如何对现有的 KISSY 组件进行扩展 ---- 扩展 Switchable

[12] Switchable Demo 页面

1.2.8 扩展已有组件

KISSY 提供的现有组件，能满足大部分需求. 但不可避免地，在某些特殊场景下，并没有考虑及实现特定的功能. 所以需要我们在已有组件基础上进行扩展.

下面，通过介绍如何实现 kwicks 效果，以此来介绍如何对 KISSY 已有组件进行扩展.

扩展 Switchable

先看效果:

一组图片，鼠标 hover 或者 click 某张图片时，这张图片完整显示，余下图片稍加压缩显示. 这样的效果，也常能在一些网站 [13] 上见到. 这与 Switchable 的切换效果很类似. 所以我们可以基于 Switchable 来实现这个 kwicks 效果.

完整代码(约 80 行): kwicks.js

这样扩展之后，引入 `kwicks.js` ，通过 `new S.Kwicks('#J_kwicks');` 调用即可~~

Note: 在 KISSY 中，写扩展和写组件，其实代码结构很类似. 只是扩展组件是在已有组件的基础上进行的，有很多代码都是可以重用的，这边，需要注意的是:

1. 可以使用 `S.extend()` 进行扩展，或者以插件形式，如 `Switchable.Plugins.push()` 来组织代码，具体参考 `S.Swichable.Plugins`;
2. 配置选项的提供，如果父类已经有了，本身就不必提供了.

发布你的扩展

当写完扩展后，可以把她放到 KISSY Gallery [14] 中，这个项目空间专门用于存放社区贡献的正式组件. 所以你可以把 自己写的 KISSY 组件/扩展 都可以提交到这里，这样别人也可以使用你的组件/扩展了.

下面，就拿 kwicks 组件的发布来举个例子，

1. 在下步之前，需要你会使用 git 的一些最基本操作，如果你还不知道，请先看 git 的基本使用文档吧 [2] ;
2. 首先进入到 KISSY Gallery [15] 的 github 网页上，将这个工程 fork 到自己的 github 账户中，这样，github 会帮你建立一个 如 [16] 的 gallary 工程空间，且你有权进行读写操作;
3. 当 clone 到本地之后，进入 kissy-gallery 目录，新建一个以你的组件名字小写的目录，如 `kwicks`;
4. 将你的代码组织好后，放入这个目录，目录结构一般包含:
   - 说明文件，如 `README`，写一些说明等;
   - 源代码文件，如 `kwicks.js`，源代码代码组织请尽量遵循 KISSY组件开发流程 ;
   - 测试文件，如 `kwicks.html`，给出你的组件的基本使用方式;
5. 提交并 Push 到你的 gallery 工程上;
6. 如果你觉得这个组件/扩展非常稳定了，就可以到你的 gallery 工程，如 [4] 上发起合并请求(Pull Request)，等待审核后就可以合并到 kissyteam 的 kissy-gallery 中了.

[13] kwicks 示范
[14] GIT 文档
[15] KISSY Gallery
[16] My Gallery

这样发布之后，其他同学也能 fork 你的代码并很有可能会发邮件向你询问某某某东西了，赶快试一下吧~

这节讲述了组件扩展及其发布的相关内容，我们可以通过扩展满足 99% 的需求，还有 1% 的那些，估计就得从头开始实现了.

所以还是得多多练习，才能逐渐深入! 下一步 ...

参考链接

1.2.9 下一步

在经过上面的介绍后，是否让你对 KISSY 了解了，不，应该是进一步的了解了??

- 如果你是 JS 大牛，我想弱弱的问一下，这里面有没哪些不对的地方？如果有，请指出~ [17]
- 如果你是 JS 新手，不知道你会不会使用 KISSY 了？如果不会，麻烦告诉我哪些地方不会，我会再仔细地写下~
- 如果你是后台开发，会不会暂时放下 jQuery / YUI / ExtJS，转而尝试使用 KISSY 了？我希望听到肯定的回答!

下一步的下一步

如果还有下一步的话，我会推荐这些内容给你，进入更深层次的 KISSY 世界:

1. 如果你对 KISSY 有 任何 疑问，可以邮件至 kissyteam(at)gmail.com
2. 如果你想研读 KISSY 源码，欢迎 fork http://github.com/kissyteam/kissy ，并可以提出意见，帮助我们不断完善 KISSY.
3. 如果你想贡献出自己的想法 + 源码，欢迎 fork http://github.com/kissyteam/kissy-sandbox，这里是我们的实验基地，用来存放各种正在开发中的组件，当实验组件开发完成并测试 ok 之后，会转成正式组件被放到 kissy-gallery 中.
4. 如果你有兴趣参与到 KISSY 组件的开发，可以参考 组件开发流程.
5. 其他的，还没想到，就等你啦!!

最后

终于到最后了，一句话: `KISSY -- An Enjoyable JavaScript Library. So Enjoy yourself!`

联系我们

[17] Liz

CHAPTER

TWO

API文档

2.1 Seed

by 承玉

2.1.1 Loader

by 承玉，fool2fish

弥补 javascript 语言机制的不足，提供类似其他语言原生的模块化机制。
refer： kissy模块化实践

KISSY.getScript

KISSY.**getScript**(url, config)
动态加载目标地址的资源文件

Parameters

- url (string) -- js/css 的资源地址
- config (object) -- 动态加载配置对象，包括

  config.charset
  类型 string，资源文件的字符编码

  config.success
  类型 function，资源加载成功后回调函数
  - < 1.2 中对于 css 文件是理解回调，而不会等 css 加载完毕
  - 1.2+ 会等 css 加载完毕

  config.error
  类型 function，超时或发生错误时回调函数。当资源文件为 css 文件时不支持

  config.timeout
  类型 number，单位为秒，默认 5 秒。超时后触发 error 回调。当资源文件为 css 文件是不支持

Return type HTMLELement

Returns 创建的 link 节点或 script 节点

KISSY.**getScript**(url, success, charset)
相当于调用 `KISSY.getScript(url , { success : success , charset : charset });`

KISSY.add

添加/注册模块,和KISSY.use一起使用，形成KISSY的模块加载体系

定义单个模块

当你需要编写一个新的模块来满足你的需求时,请使用此方式

KISSY.add(name, fn)

添加模块定义

Parameters

- name (string) -- 模块名称
- fn (function) -- 模块定义函数，若该模块的所有依赖项都已经载入完毕，则 fn 立即执行

范例: 一段典型的模块声明代码

```
KISSY.add("yourmod",function(S){

    var D = S.DOM, E = S.Event
    //  ( ) ...

    /**
     *
     */
    function YourMod(args){
        var self = this;
        //  ...
        //    ...
        //  ...
    }

    /**
     *
     */
    S.augment(YourMod, S.EventTarget, {
        /**
         *
         */
        publicMethod:function(){
            var self = this;
        },
        /**
         *
         */
        _privateMethod:function(){
            var self = this;
        }
    })

    //  ...

    S.YourMod = YourMod;
    return YourMod;
});
```

注册单个模块

当你需要注册某个已有模块时,请使用此方式

KISSY.add(name, config)

注册单个模块

Parameters

- name (string) -- 模块名称
- config (object) -- 模块描述信息

范例: 注册单个模块

```
KISSY.add('yourmod',{
    fullpath:"http://xx.com/custommod.js",
    cssfullpath:"http://xx.com/custommod.css",
    requires:["depmod1","depmod2"]
})
```

注册多个模块

当你需要注册多个已有模块时,请使用此方式

KISSY.add(config)

注册多个模块

Parameters config (object) -- 模块描述信息

范例: 注册多个模块

```
KISSY.add({
    "custommod":{
        fullpath:"http://xx.com/custommod.js",
        cssfullpath:"http://xx.com/custommod.css",
        requires:["depmod1","depmod2"]
    },
    "custommod2":{
        fullpath:"http://xx.com/custommod2.js",
        requires:["depmod1","depmod2"]
    }
});
```

指明了模块 custommod 与 custommod2 的模块文件路径以及其依赖的模块名称，
那么当 use 该模块时就会从 fullpath 中载入模块定义和其依赖模块的定义，
以及从 cssfullpath 中载入模块的样式表

Note:

- `fullpath` 也可以直接是 css 文件，这时该模块为 css 模块，如果某个 js 模块依赖项配置了 css 模块，则 js 模块的初始化会等待 css 模块对应的 css 文件完毕，具体方法可见： lifesinger 的探索 。
- 依赖的模块也要进行注册。该函数用在模块注册文件中,kissy 1.2 使用包机制取代模块注册，仍然兼容1.2以前模块注册方式。

KISSY.add (v1.2)

添加模块

`KISSY.add`(name, fn[, config])

Parameters

- name (string) -- 模块名
- fn (function) -- 模块定义函数
- config (object) -- 模块的一些格外属性，包括

  `config.attach`
  类型 bool,默认 true,表示模块定义时是否执行定义函数 fn，只有在 use 时才执行，懒加载原则

  `config.requires`
  类型 Array<string>,模块的一些依赖项，如果需要载入 css 则，数组项为 .css 结尾名字

Note: 模块名 `name` 也可以省略，不过部署阶段需要使用 `KISSY Module Compiler` .

范例: 添加模块

```
KISSY.add("yourmod",function(S){},
    {
        attach:false,//         fn    use
        requires:['depMod1','depMod2','./mod.css'] //
                                                   //   css     js     mod.css
    }
);
```

如果依赖的模块 `depMod1` 以及 `depMod2` 的定义函数有返回值，例如

```
KISSY.add("depMod1",function(){
    function Mod(){}
    return Mod;
});

KISSY.add("depMod2",function(){
    function Mod(){}
    return Mod;
});
```

那么该返回值会作为参数传入依赖 depMod1 以及 depMod2 的模块的定义函数，例如

```
KISSY.add("custommode",function(S,DepMod1,DepMod2){
    //use DepMod1 to refer depmod1's return value
},{requires:["depmod1","depmod2"]});
```

当模块名称 name 为包内模块< 参见 >时，则requires的模块名称可使用相对路径 refer 包内其他模块

```
// tc/mods/mod1    tc/mods/mod2
KISSY.add("tc/mods/mod1",function(){},requires:['./mod2']);
```

包配置

为了摆脱模块必须使用前注册的繁琐

KISSY.**config**(config)

Parameters config (object) -- 包含 key 为 packages 的配置项，包括

config.**packages**
类型数组，每个数组项为一个包的配置，一个包配置包括 4 项：

packages.**name**
类型字符串，表示包名

packages.**tag**
类型字符串，最好为时间戳，用于刷新客户端本包的模块文件缓存

packages.**path**
类型字符串，表示包所在的 url 路径，相对路径表示相对于当前页面路径，如果需要相对于当前

```
var scripts=document.getElementsByTagName("script");
alert(scripts[scripts.length-1].src);
```

packages.**charset**
类型字符串，表示宝贝所有模块定义文件的编码格式，默认 utf-8

范例: 包配置

```
KISSY.config({
    packages:[
        {
            name:"tc", //
            tag:"20110323", //      js
                            //    ?t=20110323
            path:"../", //
            charset:"gbk" //
        }
    ]
});
```

当要在包内添加模块时，必须遵守一些约定：

1. 一个模块的文件必须放在以包名命名的目录中
2. 模块的名字必须取名从包目录开始到当前模块文件的文件路径名称，例如 mod1.js 位于 tc/mods 下，则 mod1.js 的模块取名：

```
KISSY.add("tc/mods/mod1",function(){});
```

Note: 模块名也可以省略，不过部署阶段需要使用 KISSY Module Compiler .

压缩模块

若线上环境使用 kissy-min.js ，则请使用 closure compiler 对所有模块文件进行压缩， 例如 mod.js 压缩为 mod-min.js ，放在模块文件的同级目录下。

代码更新机制

由于动态加载的 js 文件不是写在页面中，所以不能从页面添加时间戳，并且1.2 loader新增的约定加载也不能配置具体模块文件路径，因此 1.2 loader 提供了在包级别添加时间戳的机制

```
KISSY.config({
    packages:[
        {
            name:"1.2", //
            path:"http://xx.com/"
        }
    ]
});
```

当更改包内模块后，只需修改tag属性。

```
KISSY.config({
    packages:[
        {
            name:"1.2", //
            tag:"20110323",
            path:"http://xx.com/"
        }
    ]
});
```

那么下载动态加载的 js 文件路径后面会自动加上： ?t=20110323

Note: 也可以不修改时间戳 tag 而是直接修改 path，这样的话每次更新都需要新建一个目录包括更新后的全部代码.

静态部署

部署时也可以不采用动态加载，仅仅将 kissy loader 作为代码组织的一种方式，将所有的模块打包到一个文件静态引入放在页面中， 当使用 KISSY.use 时如果模块已经过静态引入在页面中，则不会发送请求，这时建议所有模块的属性都设置为

```
KISSY.add("custommod",function(){},{attach:false});
```

attach 设置为 false，表示只有在 use 时才会运行模块定义函数，消除模块过多而导致的页面初始化时的停滞问题。若定义模块时不写模块名则默认 attach 为 false .

KISSY.use

使用模块,和KISSY.add一起使用，形成KISSY的模块加载体系

KISSY.use(modNames[, callback])

Parameters

- modNames (string) -- 以 ， 分割的模块名称集合字符串，例如 KISSY.use("custommod,custommod2");
- callback (function) -- 当 modNames 中所有模块加载完毕后执行

范例: 使用模块

```
// 1.2 , function DepMod1 DepMod2
KISSY.use("depMod1,depMod2",function(S,DepMod1,DepMod2){
});
```

如果使用经过配置的包内的模块，则这些包内模块不需要事先注册，直接 use 即可， 如果模块名以 / 结尾，则自动加后缀 index ，例如 use("mods/m1/") 相当于 use("mods/m1/index") ，即自动加载 m1 目录下的 index.js

---

Note: 注意 kissy < 1.2 时使用模块前必须注册

2.1.2 Seed

by 承玉，玉伯

KISSY.version

KISSY.version

类型 string ，KISSY 类库的版本号。可通过 KISSY.version 获取。

KISSY.Config

KISSY.Config

类型 object ，目前暴露

Config.debug

类型 boolean，可以通过设置 KISSY.Config.debug = true 来开启 debug 模式。debug 模式默认关闭，但在以下情况下会自动开启：

1.引入的 js 文件是未压缩版本，比如 <script src=''kissy.js''></script>

2.访问的 url 路径中，带有 ks-debug 参数，比如 http://www.taobao.com/?ks-debug

debug 模式开启时，源码中的 S.log 会利用浏览器的 console 对象输出调试信息。debug 模式关闭时，不会输出调试信息。

KISSY.mix

KISSY.**mix**(receiver, supplier[, overwrite = true, whitelist])

将 supplier 对象的成员复制到 receiver 对象上。

Parameters

- receiver (object) -- 属性接受者对象。
- supplier (object) -- 属性来源对象。
- overwrite (boolean) -- 是否覆盖接受者同名属性。
- whitelist (Array<string>) -- 属性来源对象的属性白名单，仅在名单中的属性进行复制。

Returns receiver 属性接受者对象。

Return type object

例如：

```
var S = KISSY,
r = { a: 'a', b: 'b' };

S.mix(r, { c: 'c' });
S.log(r.c); // => 'c'

S.mix(r, { a: 'a2' }, false);
S.log(r.a); // => 'a'

S.mix(r, { e: 'e', f: 'f' }, true, ['f']);
S.log(r.e); // => undefined
S.log(r.f); // => 'f'
```

该方法在 KISSY 里具有非常重要的地位。 JavaScript 是一门动态语言，利用 mixin 特性，可以很方便的实现特性的静态复制和动态修改。

KISSY.merge

KISSY.**merge**(s1, s2[, ...])

将多个对象的成员合并到一个新对象上。参数中，后面的对象成员会覆盖前面的。

Parameters

- s1 (object) -- 属性源
- s2 (object) -- 属性源

Returns 合并属性后的新对象。

Return type object

例如：

```
var S = KISSY,
a = { a: 'a' },
b = { b: 'b' },
c = { b: 'b2', c: 'c' };

var o = S.merge(a, b, c);
S.log(o.a); // => 'a'
```

```
S.log(o.b); // => 'b2'
S.log(o.c); // => 'c'
```

简单情况下 merge 方法常用来合并配置信息。推荐使用 Base 处理属性配置。

KISSY.augment

KISSY.**augment**(r, s[, ov=true, wl])

将 s.prototype 的成员复制到 r.prototype 上。

Parameters

- r (function) -- 将要扩充的函数
- s (function|object) -- 扩充来源函数或对象。非函数对象时复制的就是 s 的成员。
- ov (boolean) -- 是否覆盖 r 已经有的原型属性。
- wl (Array<string>) -- 来源属性的白名单列表，只有列表中的被扩充。

Returns r

Return type function

例如：

```
var S = KISSY,
Shoutable = {
    shout: function() { alert('I am ' + this.name + '.'); }
};

function Dog(name) { this.name = 'Dog ' + name; }
function Pig(name) { this.name = 'Pig ' + name; }

S.augment(Dog, Shoutable);
S.augment(Pig, Shoutable);

new Dog('Jack').shout(); // => I am Dog Jack.
new Pig('Mary').shout(); // => I am Pig Mary.
```

augment 方法在 KISSY 里非常基础非常重要。 传统 OO 语言里，可以通过继承或接口来实现共性方法。 在 JavaScript 里，通过 mixin 特性，一切变得更简单。 augment 是动态语言 mixin 特性的体现，灵活运用，能让代码非常优雅简洁。

KISSY.extend

KISSY.**extend**(r, s[, px, sx])

让函数对象 r 继承函数对象 s

Parameters

- r (function) -- receiver,将要继承的子类函数
- s (function|object) -- supplier,继承自的父类函数
- px (object) -- prototype members, 需要添加/覆盖的原型成员
- sx (object) -- static members, 需要添加/覆盖的静态成员。

Returns r

Return type function

例如：

```
var S = KISSY;

function Bird(name) { this.name = name; }
Bird.prototype.fly = function() { alert(this.name + ' is flying now!'); };

function Chicken(name) {
    Chicken.superclass.constructor.call(this, name);
}
S.extend(Chicken, Bird,{
    fly:function(){
        Chicken.superclass.fly();
        alert("it's my turn");
    }
});

new Chicken('Frank').fly();
```

extend 方法是 KISSY 里类继承的实现方式。书写 JavaScript 代码时，请忘记传统 OO 里的继承体系。 还 JavaScript 本色，给代码一身轻松。

Note: 子类方法中可通过 superclass 来访问父类函数的原型，进而调用父类方法.

KISSY.namespace

KISSY.**namespace**(n1[, ..., global=false])

将 s.prototype 的成员复制到 r.prototype 上。

Parameters

- n1 (string) -- 命名空间字符串，如 "fp.search" 或 "KISSY.fp.ad"
- global (boolean) -- 是否第一个点之前的字符串作为全局变量，默认 false 添加到 KISSY

Returns 最后创建的命名空间对象

Return type object

例如：

```
var S = KISSY;

S.namespace('app', 'test'); //   KISSY.app   KISSY.test
S.namespace('app.Shop'); //   KISSY.app.Shop
S.namespace("TC.mods",true); //  window.TC.mods
```

namespace 方法提供了最基本的命名空间管理。但对于模块的命名空间推荐采用 `kissy 1.2 loader` 机制. Deprecated since version 1.2.

KISSY.app

KISSY.**app**(name[, sx])

创建应用对象，为全局 window 中名字为 name 的变量。

Parameters

- name (string) -- 应用对象名称
- sx (object) -- mix 到应用对象的属性以及值。

Returns 创建的应用对象

Return type object

例如：

```
KISSY.app('FrontPage');
FrontPage.namespace('app'); //   FrontPage.app
FrontPage.add('slide', function(FP) {
    // module code
    //  FP   FrontPage
});
```

app 方法为基于 KISSY 的应用提供了最基本的代码组织方式。通过 app 创建的应用对象，自动具有了 add 和 namespace() 方法。 Deprecated since version 1.2: 简化架构，两层构建应用：应用 = KISSY 环境 + 业务模块。模块通过 kissy 1.2 loader 进行命名空间隔离， 模块统统通过 KISSY.add 加入 KISSY 环境中。

KISSY.log

KISSY.log(msg[, cat='log', src])

输出调试信息

Parameters

- msg (string) -- 调试信息
- cat (string) -- 调试信息类别。可以取 info, warn, error, dir, time 等 console 对象的方法名，默认为 log.
- src (string) -- 调试代码所在的源信息

Note: 只有在 debug 模式下，才会输出调试信息。debug 模式的说明请参考 Config

KISSY.error

KISSY.error(msg)

抛出错误异常

Parameters msg (string) -- 异常信息

Note: 只有在 debug 模式下，才会抛出异常。debug 模式的说明请参考 Config

KISSY.guid

KISSY.guid(prefix)

返回全局唯一 id.

Parameters prefix (string) -- 唯一 id 前缀

2.1.3 Web

by 承玉，玉伯

KISSY.ready

KISSY.**ready**(fn)
Parameters fn (function) -- 回调函数，在 DOM 加载完毕时执行。

例如：

```
KISSY.ready(function(S) {
    // code
});
```

这是 KISSY 外部代码的基本调用方式。 为了保证代码执行时，依赖的 DOM 结构已准备好，推荐尽可能的将代码写在通过 ready 注册的函数里。

Note: 在 DOM 加载完毕后，依旧可以通过 ready 添加函数，此时会立刻执行。

KISSY.available

KISSY.**available**(id, fn)
Parameters

- id (string) -- 页面元素 id
- fn (function) -- 回调函数，在 id 元素可用时立刻执行。

Note: 当需要比 KISSY.ready() 反应更快的探测到某个元素可用时使用。

KISSY.isWindow

KISSY.**isWindow**(o)
判断参数是否为浏览器 window
Parameters o -- 需要判断的对象

目前实现为：

```
isWindow: function(o) {
    return S.type(o) === 'object'
        && 'setInterval' in o
        && 'document' in o
        && o.document.nodeType == 9;
}
```

有更好的实现，欢迎提出.

KISSY.globalEval

KISSY.**globalEval**(code)

在全局作用域下执行代码字符串，避免 eval 的作用域链

Parameters code (string) -- 代码字符串

2.1.4 Lang

by 承玉，玉伯

KISSY.clone

KISSY.**clone**(o[, filter])

创建一个 或数组的深拷贝, 并且返回。

Parameters

- o (object|Array) -- 待深拷贝的对象或数组。
- filter -- 过滤函数，返回 false 不拷贝该元素。传入参数为 * 待克隆值为数组，参数同 KISSY.filter() ，上下文对象为全局 window * 待克隆值为普通对象，参数为对象的每个键，每个键对应的值，当前对象，上下文对象为当前对象。

Returns 拷贝后的新对象

Type object

例如

```
var S = KISSY;
var a={x:{y:{z:1}}}
var b=S.clone(a); // => b={x:y:{z:1}} , b!==a
var c=S.clone(a,function(v,k){if(k=="z") return false;}) // => c={x:{y:{}}}
```

KISSY.each

KISSY.**each**(o, fn[, context])

遍历数组中的每一项，执行指定方法。

Parameters

- o (Array|object) -- 需要遍历的数组或对象
- fn (function) -- 执行时，接收 3 个参数：
  - 当 o 为数组时，参数为当前数组单项值，当前 index，数组 o
  - 当 o 为对象时，参数为当前值 (value)，当前键 (key)，对象 o
- context (object) -- fn 的上下文对象，不指定为全局 window

例如

```
var S = KISSY,
arr = [1, 2, 3, 4, 5],
sum = 0;

S.each(arr, function(item) {
    sum += item;
```

```
});
S.log(sum); // => 15
```

KISSY.later

KISSY.later(fn[, when, periodic, o, data])

延迟执行指定函数 fn

Parameters

- fn (function) -- 延迟执行的函数。
- when (number) -- 延迟时间，单位是毫秒。
- periodic (boolean) -- 是不是周期性执行。默认为 false.
- o (object) -- fn 上下文对象
- data (Array) -- 传递的参数。可以为单个对象，最后会转换成数组，依次传递给执行函数。

Returns

timer 对象。包含属性 .. attribute:: timer.interval

是否周期执行

timer.cancel

取消定时器

Return type object

例如

```
var S = KISSY;

S.later(function(data) {
    S.log(data);
}, 0, false, null, 'I am later data.');
```

KISSY.filter

KISSY.filter(arr, fn[, context])

遍历数组，过滤出符和条件的数组项。

Parameters

- arr (Array) -- 需要遍历的数组。
- fn (function) -- 过滤函数。执行时，接收 3 个参数：当前项、当前 index, 数组。
- context (object) -- fn 执行的上下文对象

Returns 返回符合过滤函数的新数组

Return type Array

例如

```
var S = KISSY,
arr = [1, 2, 3, 4, 5];

var ret = S.filter(arr, function(item) {
    return item % 2 === 0;
});
S.log(ret); // => [2, 4]
```

KISSY.map

New in version 1.2.

KISSY.**map**(arr, fn[, context])

创建一个新数组，数组结果是在对每个原数组元素调用指定函数的返回值。

Parameters

- arr (Array) -- 需要遍历的数组。
- fn (function) -- 能够根据原数组当前元素返回新数组元素的函数。
- context (object) -- 执行 `fn` 时的 `this` 值。

Returns 返回符合根据指定函数调用得到新数组

Return type Array

Note: 原数组保持不变

例如

```
function makePseudoPlural(single) {
    return single.replace(/o/g, "e");
}

var singles = ["foot", "goose", "moose"];
var plurals = S.map(singles, makePseudoPlural);  // => ["feet", "geese", "meese"]

var a = S.map("Hello World",function(x) {
    return x.charCodeAt(0);
}); // => [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```

KISSY.inArray

KISSY.**inArray**(elem, arr)

判断元素 elem 是否在数组 arr 中。

Parameters

- elem -- 任意对象
- arr -- 数组

Returns elem 是否在数组 arr 中。

Return type boolean

KISSY.indexOf

KISSY.**indexOf**(elem, arr)

返回元素 elem 在数组 arr 中的序号。

Parameters

- elem -- 任意对象
- arr -- 数组

Returns 元素 elem 在数组 arr 中的序号。

Return type number

KISSY.lastIndexOf

KISSY.**lastIndexOf**(elem, arr)

返回元素 elem 在数组 arr 中最后出现的序号。

Parameters

- elem -- 任意对象
- arr -- 数组

Returns 元素 elem 在数组 arr 中最后出现的序号。

Return type number

KISSY.unique

KISSY.**unique**(arr[, keepLast=false])

返回一个新数组，仅包含 arr 去重后的值

Parameters

- arr (Array) -- 包含重复元素的数组
- keepLast (boolean) -- 遇到重复值是保留第一次出现还是保留最后一次出现的元素

Returns 包含 arr 去重后的数组

Return type Array

例如

```
KISSY.unique([a, b, a],true) => [b, a]

KISSY.unique([a, b, a]) => [a, b]
```

KISSY.isArray

KISSY.**isArray**(o)

判断是否数组

Parameters o -- 判断参数

KISSY.isBoolean

KISSY.**isBoolean**(o)
判断是否布尔值。
Parameters o -- 判断参数

KISSY.isDate

KISSY.**isDate**(o)
判断是否日期
Parameters o -- 判断参数

KISSY.isEmptyObject

KISSY.**isEmptyObject**(o)
判断是否空对象（没有任何可遍历的属性）。
Parameters o -- 判断参数

例如

```
var S = KISSY;

S.isEmptyObject({}); // => true
S.isEmptyObject([]); // => true
S.isEmptyObject({ a: 'a' }); // => false
```

KISSY.isFunction

KISSY.**isFunction**(o)
判断是否函数
Parameters o -- 判断参数

KISSY.isNull

KISSY.**isNull**(o)
判断是否 null
Parameters o -- 判断参数

KISSY.isNumber

KISSY.**isNumber**(o)
判断是否有效数值。
Parameters o -- 判断参数

Note: `NaN` 和 `Infinity` 也返回 true

KISSY.isObject

KISSY.**isObject**(o)

判断是否为对象

Parameters o -- 判断参数

KISSY.isPlainObject

KISSY.**isPlainObject**(o)

判断是否是普通对象，通过 {} 或 new FunctionClass/Object() 创建的，不包括内置对象以及宿主对象。

Parameters o -- 判断参数

例如

```
var S = KISSY;

S.isPlainObject({}); // => true
S.isPlainObject(new Date()); // => false
S.isPlainObject(document.body); // => false
```

KISSY.isRegExp

KISSY.**isRegExp**(o)

判断是否正则表达式

Parameters o -- 判断参数

KISSY.isString

KISSY.**isString**(o)

判断是否字符串。

Parameters o -- 判断参数

KISSY.isUndefined

KISSY.**isUndefined**(o)

判断是否 undefined

Parameters o -- 判断参数

KISSY.makeArray

KISSY.**makeArray**(o)

将对象 o 转换为数组。

Parameters o -- arguments, NodeList 等 array-like 对象或单个对象

Returns 可以代表 o 的新数组

Return type Array

例如

```
var S = KISSY;

S.makeArray('str'); // => ['str']
S.makeArray(S.query('.div')); // =>    div
S.makeArray(null); // => []
```

KISSY.substitute

KISSY.**substitute**(str, o)

将字符串中的占位符替换为对应的键值。

Parameters

- str (string) -- 包含数据占位符的模板字符串，占位符用 {} 包起来。
- o (object) -- 数据

Returns 将模板和数据结合起来的最终字符串

Return type string

例如

```
var S = KISSY,
str = '{name} is {prop_1} and {prop_2}.',
obj = {name: 'Jack Bauer', prop_1: 'our lord', prop_2: 'savior'};

S.substitute(str, obj); // => 'Jack Bauer is our lord and savior.'
```

KISSY.trim

KISSY.**trim**(str)

去除字符串两端的空白字符。

Parameters o -- 判断参数

Returns 去除空白后新的字符串

Return type string

KISSY.fromUnicode

New in version 1.2.

KISSY.**fromUnicode**(str)

将 str 中 unicode 转义的字符替换成真实字符。主要用于 taobao 用户名 cookie 读取。

Parameters str (string) -- 包含 unicode 转义的字符串

Returns unicode 转义后的字符串

Return type string

例如

```
KISSY.fromUnicode("\\u627F\\u7389") // => " "
```

KISSY.escapeHTML

New in version 1.2.

KISSY.**escapeHTML**(str)

将字符串经过 html 转义得到适合在页面中显示的内容，例如替换 < 为 `<`

Parameters str (string) -- 要显示在页面中的真实内容

Returns 经过 html 转义后的字符串

Return type string

例如

```
KISSY.escapeHTML("<a>x</a>"); // =>  "&lt;a&gt;x&lt;/a&gt;"
```

KISSY.unEscapeHTML

New in version 1.2.

KISSY.**unEscapeHTML**(str)

将字符串中的 html 实体字符替换成对应字符

Parameters str (string) -- 包含 html 实体字符的字符串

Returns 替换实体字符后的字符串

Return type string

例如

```
KISSY.unEscapeHTML("&lt;a&gt;x&lt;/a&gt;"); // =>  "<a>x</a>"
```

KISSY.now

KISSY.**now**()

返回 new Date().getTime()

KISSY.param

KISSY.**param**(o[, sep='&', eq='=', arr=true])

将对象 o 转换为参数字符串，用于发送 http 请求。

Parameters

- o (object) -- 参数键值对对象
- seq (string) -- 参数间分隔符，默认 `&`
- eq (string) -- 参数与参数值间的分隔符，默认 =
- arr (boolean) -- 参数值为数组时，参数键是否加 `[]` 即 `%5B%5D` ，默认 `true`

Returns 可用于发送请求的参数字符串

Return type string

例如

```
var S = KISSY;

S.param({ foo: 1, bar: 2 }); // => foo=1&bar=2
S.param({ foo: 1, bar: [2, 3] }); // => foo=1&bar%5B%5D=2&bar%5B%5D=3
S.param({ foo: 1, bar: [2, 3] },'&','=',false); // => foo=1&bar=2&bar=3
S.param({ foo: '', bar: 2 }); // => foo=&bar=2
S.param({ foo: undefined, bar: 2 }); // => foo=undefined&bar=2
```

See Also:

Jquery.param

KISSY.unparam

KISSY.**unparam**(str[, sep='&', eq='='])

将参数字符串 str 还原为对象。

Parameters

- o (object) -- 参数字符串
- seq (string) -- 参数间分隔符，默认 &
- eq (string) -- 参数与参数值间的分割符，默认 =

Returns 参数的对象表示

Return type object

Changed in version 1.2: key 可以不加 [] 如 v=1&v=2 => {v:[1,2]}

Note: 参数值如果是 gbk 编码的，则不会解码出对应的真实值。（用的原生 decodeURIComponent，请修改参数值为 utf-8 编码）。

例如

```
var S = KISSY;

S.unparam('foo=1&bar=2'); // => { foo: 1, bar: 2 }
S.unparam('foo=%81%47'); // gbk   => { foo: "%81%47" }    {foo: " "}
S.unparam('foo=1&bar=2&bar=3'); // => { foo: 1, bar: [2, 3] }
S.unparam('foo=1&bar%5B%5D=2&bar%5B%5D=3'); // => { foo: 1, bar: [2, 3] }
```

KISSY.startsWith

New in version 1.2.

KISSY.**startsWith**(str, prefix)

判断 str 是否以 prefix 开头

Parameters

- str (string) -- 查找字符串
- prefix (string) -- 前缀字符串

Returns str 是否以 prefix 开头

Return type boolean

KISSY.endsWith

New in version 1.2.

KISSY.**endsWith**(str, suffix)

判断 `str` 是否以 `suffix` 结尾

Parameters

- str (string) -- 查找字符串
- suffix (string) -- 后缀字符串

Returns `str` 是否以 `suffix` 结尾

Return type boolean

2.2 Core

2.2.1 DOM

by 玉伯，承玉 Changed in version 1.2: DOM 不再接受 `Node` 以及 NodeList 类型的参数，DOM 只处理选择器以及原生 DOM 节点。 KISSY Node 以及 `NodeList` 对象上的相关处理可直接调用其自身方法。

KISSY selector

KISSY 选择器内置仅支持 #id tag.class 常用形式，考虑 2/8 原则，支持以下选择器：

- #id
- tag
- .cls
- #id tag
- #id .cls
- tag.cls
- #id tag.cls

Note:

- tag 可以为 * 字符
- 支持 , 号分组

例如

```
KISSY.use("dom",function(S,DOM){
    //  class   J_on   a
    var els=DOM.query("a.J_on");
});
```

当加载了 `sizzle` 模块时，KISSY 支持 jQuery 支持的所有 CSS 选择器，具体请参考： Sizzle Documents

例如

```
/**

**/
KISSY.use("dom,sizzle",function(S,DOM){
    //          a
    var els=DOM.query("a:first-child");
});

/**

**/
KISSY.add("biz/custom",function(S,DOM){
    //          a
    var els=DOM.query("a:first-child");
},{
    requires:["dom","sizzle"]
});
```

DOM.query

DOM.**query**(selector[, context=document])
获取符合选择器的所有元素。
Parameters
- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- context (string|HTMLElement) -- 选择器参考上下文，'#id' 或者 dom 节点。

Returns 符合选择器字符串的 dom 节点数组
Return type Array<HTMLElement>

KISSY.**query**(selector[, context=document])
DOM.query() 的快捷方式

DOM.get

DOM.**get**(selector[, context=document])
获取符合选择器的第一个元素。相当于调用 DOM.query(selector,context)[0]

KISSY.**get**(selector[, context=document])
DOM.get() 的快捷方式

DOM.filter

DOM.**filter**(selector, filter[, context=document])
获取符合选择器以及过滤参数的所有元素。
Parameters
- selector (string) -- 选择器字符串，格式参见 KISSY selector
- filter (string|function) -- 过滤选择器或函数
  - 类型 string 时，格式为 tag.cls ，其他格式需要引入 sizzle

– 类型 function 时，传入参数当前 dom 节点，返回 `true` 表示保留

- context (string|HTMLElement) -- 选择器参考上下文， `#id` 或者 dom 节点。

Returns 符合选择器字符串以及过滤参数的 dom 节点数组

Return type Array<HTMLElement>

DOM.test

`DOM.test(selector, filter[, context=document ])`

判断根据选择器获取的所有元素是否都符合过滤条件。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (string|function) -- 过滤选择器或函数，具体详见 `DOM.filter()`
- context (string|HTMLElement) -- 选择器参考上下文， `#id` 或者 dom 节点。

Returns 选择器获取的所有元素是否都符合过滤条件。

Return type boolean

DOM.hasClass

`DOM.hasClass(selector, value)`

判断符合选择器的所有元素中是否有某个元素含有特定 class.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 样式类 class，多个用空格分隔，表示同时包含多个样式类

Returns 是否符合选择器的元素中存在某个元素含有特定样式类 value

Return type boolean

DOM.addClass

`DOM.addClass(selector, value)`

给符合选择器的所有元素添加指定 class.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 样式类 class，多个用空格分隔

DOM.removeClass

`DOM.removeClass(selector, value)`

给符合选择器的所有元素移除指定 class.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 样式类 class，多个用空格分隔

DOM.replaceClass

DOM.**replaceClass**(selector, oldClassName, newClassName)

将符合选择器的所有元素的老 class 替换为新 class.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- oldClassName (string) -- 样式类 class，多个用空格分隔 ，需要删除的样式类
- newClassName (string) -- 样式类 class，多个用空格分隔 ，需要添加的样式类

DOM.toggleClass

DOM.**toggleClass**(selector, value)

操作符合选择器的所有元素，如果存在值为 value 的 class, 则移除掉，反之添加。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 样式类 class，多个用空格分隔 ，需要 toggle 的样式类

DOM.removeAttr

DOM.**removeAttr**(selector, name)

移除符合选择器的所有元素的指定属性。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 属性名称

DOM.attr

DOM.**attr**(selector, name)

获取符合选择器的第一个元素的属性值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 属性名称

Returns 对应属性名的属性值

DOM.**attr**(selector, name, value)

给符合选择器的所有元素设置属性值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 属性名称
- value -- 属性值

DOM.**attr**(selector, kv)

给符合选择器的所有元素设置属性值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- kv (object) -- 属性名与属性值的键值对

例如

```
var S = KISSY, DOM = S.DOM;

//
DOM.attr('img', { src: 'kissy.png', width: 400, height: 400 });
```

DOM.hasAttr

New in version 1.2.

DOM.**hasAttr**(selector, attrName)

判断符合选择器的所有元素中是否有某个元素含有特定属性.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- attrname (string) -- 属性名称

Return type boolean

Returns 符合选择器的所有元素中是否有某个元素含有特定属性.

DOM.val

DOM.**val**(selector)

获取符合选择器的第一个元素所的 value 值。

Parameters selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Returns 获取符合选择器的第一个元素所的 value 值。无值时，返回空字符串。

DOM.**val**(selector, value)

给符合选择器的所有元素设置 value 值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 将要设置的 value 值

DOM.text

DOM.**text**(selector)

获取符合选择器的第一个元素所包含的文本值。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Returns 获取符合选择器的第一个元素所包含的文本值。无值时，返回空字符串。

DOM.**text**(selector, value)

给符合选择器的所有元素设置文本值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 将要设置的文本值

相当与 ie 下调用 `innerText` 以及其他浏览器下调用 `textContent` .

DOM.css

DOM.**css**(selector, name)

获取符合选择器的第一个元素的样式值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- css 样式属性名

Returns 获取符合选择器的第一个元素的样式值。

DOM.**css**(selector, name, value)

给符合选择器的所有元素设置样式值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- css 样式属性名
- value (string) -- 将要设置的样式值

DOM.**css**(selector, kv)

给符合选择器的所有元素设置样式值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- kv (object) -- 样式名与样式值的键值对，例如

```
DOM.css('.widget', {position: 'absolute', top: '10px', left: '10px'});
```

DOM.width

DOM.**width**(selector)

获取符合选择器的第一个元素的宽度值。

Parameters selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Returns 符合选择器的第一个元素的宽度值。

Note: 该方法始终返回像素值，例如

```
<div style="width: 100px;">
    <div id="test" style="width: 80%; height: 20px"></div>
</div>
<script>
    var S = KISSY, DOM = S.DOM,
    elem = S.get('#test');

    DOM.css(elem, 'width'); //   80%
    DOM.css(elem, 'height'); //   20px

    DOM.width(elem); //   80
    DOM.height(elem); //   20
</script>
```

DOM.**width**(selector, value)

给符合选择器的所有元素设置宽度值。相当于 DOM.css(selector,"width",value)

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 宽度值

DOM.height

DOM.**height**(selector)

获取符合选择器的第一个元素的高度值。

Parameters selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Returns 符合选择器的第一个元素的高度值。

Note: 该方法始终返回像素值

DOM.**height**(selector, value)

给符合选择器的所有元素设置高度值。相当于 DOM.css(selector,"height",value)

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (string) -- 宽度值

DOM.addStyleSheet

DOM.**addStyleSheet**(cssText[, id])

将 cssText 字符串作为内联样式添加到文档中。

Parameters

- cssText (string) -- 样式内容
- id (string) -- 内联样式所在 style 节点的 id

DOM.show

DOM.**show**(selector)

显示符合选择器的所有元素。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

DOM.hide

DOM.**hide**(selector)

隐藏符合选择器的所有元素。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

DOM.toggle

DOM.**toggle**(selector)

将符合选择器的所有元素切换显示/隐藏两个状态。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

DOM.offset

DOM.**offset**(selector)

获取符合选择器的第一个元素相对页面文档左上角的偏移值。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Return type object

Returns

相对页面文档左上角的偏移值，包括两个属性

DOM.**left**
: 类型 number，相对页面文档左上角的横坐标

DOM.**top**
: 类型 number，相对页面文档左上角的纵坐标

DOM.**offset**(selector, value)

给符合选择器的所有元素设置偏移值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- value (object) -- 偏移对象，包括两个属性 `left`，``top`` ，格式同获取偏移的返回值。

DOM.scrollTop

DOM.**scrollTop**(node)
获取窗口或元素的 scrollTop 值。
Parameters node (Window|HTMLElement) -- 某个 iframe 的 contentWindow 或当前 window 或某个节点。
Return type number
Returns 窗口或元素的 scrollTop 值。

DOM.**scrollTop**(num)
New in version 1.2: 设置窗口 scrollTop 值。
Parameters num (number) -- 将要设置的 scrollTop 值
Return type number
Returns 设置的值

DOM.**scrollTop**(node, num)
New in version 1.2: 设置窗口或元素的 scrollTop 值。
Parameters node (Window|HTMLElement) -- 某个 iframe 的 contentWindow 或当前 window 或某个节点。
Return type number
Returns 设置的值

DOM.scrollLeft

DOM.**scrollLeft**(node)
获取窗口或元素的 scrollLeft 值。
Parameters node (Window|HTMLElement) -- 某个 iframe 的 contentWindow 或当前 window 或某个节点。
Return type number
Returns 窗口或元素的 scrollLeft 值。

DOM.**scrollLeft**(num)
New in version 1.2: 设置窗口 scrollLeft 值。
Parameters num (number) -- 将要设置的向左滚动值
Return type number
Returns 设置的值

DOM.**scrollLeft**(node, num)
New in version 1.2: 设置窗口或元素的 scrollLeft 值。
Parameters node (Window|HTMLElement) -- 某个 iframe 的 contentWindow 或当前 window 或某个节点。
Return type number
Returns 设置的值

DOM.docHeight

DOM.**docHeight**()
获取页面文档 document 的总高度。

Return type number

Returns 页面文档 document 的总高度。

DOM.docWidth

DOM.**docWidth**()
获取页面文档 document 的总宽度。
Return type number

Returns 页面文档 document 的总宽度。

DOM.viewportHeight

DOM.**viewportHeight**()
获取当前可视区域(viewport)的高度值。
Return type number

Returns 当前可视区域(viewport)的高度值。

DOM.viewportWidth

DOM.**viewportWidth**()
获取当前可视区域(viewport)的宽度值。
Return type number

Returns 当前可视区域(viewport)的宽度值。

DOM.scrollIntoView

DOM.**scrollIntoView**(selector[, container = window, top = true, hscroll = true])
使当前选择器匹配的第一个元素出现在指定容器可视区域内。
Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- container (window|HTMLElement) -- 指定容器
- top (boolean) -- 是否强制元素的上边界与容器的上边界对齐，左边界和左边界对齐
- hscroll (boolean) -- 是否允许容器左右滚动以保证元素显示在其可视区域。

DOM.parent

DOM.**parent**(selector[, filter])
获取符合选择器的第一个元素的祖先元素。
Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (number|string|function) -- 过滤条件，整数外的例子参见 DOM.filter() 的相应参数

Returns 符合选择器的第一个元素的祖先元素。

Return type HTMLElement

filter 举例：

```
var S = KISSY, DOM = S.DOM,
elem = S.get('#id');

//   elem.parentNode
DOM.parent(elem);

//   elem.parentNode.parentNode
DOM.parent(elem, 2);

//   elem         container class
DOM.parent(elem, '.container');

//   elem        tagName   ul
DOM.parent(elem, 'ul');

//   elem        rel    data
DOM.parent(elem, function(p) {
    return DOM.attr(p, 'rel') == 'data';
});
```

DOM.next

DOM.**next**(selector[, filter])

获取符合选择器的第一个元素的下一个同级节点。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (string|function) -- 过滤条件，格式参见 DOM.filter() 的相应参数

Returns 符合选择器的第一个元素的下一个同级节点。

Return type HTMLElement

DOM.prev

DOM.**prev**(selector[, filter])

获取符合选择器的第一个元素的上一个同级节点。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (string|function) -- 过滤条件，格式参见 DOM.filter() 的相应参数

Returns 符合选择器的第一个元素的上一个同级节点。

Return type HTMLElement

DOM.siblings

DOM.**siblings**(selector[, filter])
获取符合选择器的第一个元素的相应同级节点。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (string|function) -- 过滤条件，格式参见 DOM.filter() 的相应参数

Returns 符合选择器的第一个元素的相应同级节点。

Return type Array<HTMLElement>

DOM.children

DOM.**children**(selector[, filter])
获取符合选择器的第一个元素的相应子节点。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- filter (string|function) -- 过滤条件，格式参见 DOM.filter() 的相应参数

Returns 符合选择器的第一个元素的相应子节点。

Return type Array<HTMLElement>

DOM.contains

DOM.**contains**(container, contained)
判断某一容器（container）是否包含另一（contained）节点。

Parameters

- container (string|HTMLElement) -- 容器节点。字符串格式参见 KISSY selector 获取匹配的第一个元素。
- contained (string|HTMLElement) -- 检测节点。字符串格式参见 KISSY selector 获取匹配的第一个元素。

Returns container 是否包含 contained 节点。

Return type boolean

DOM.create

DOM.**create**(html[, props = {}, ownerDoc = document])
创建 dom 节点

Parameters

- html (string) -- dom 节点的 html
- props (object) -- 属性键值对象
- ownerDoc (HTMLDocument) -- 节点所属文档

Return type HTMLFragment|HTMLElement

Returns 创建出的 dom 节点或碎片列表

举例：

```
var S = KISSY, DOM = S.DOM;

//   document.createElement('div')
DOM.create('<div>');
DOM.create('<div />');
DOM.create('<div></div>');

//   document.createTextNode('text')
DOM.create('text');

//
DOM.create('<a>', { href: 'hot.html', title: 'Hot Page' });

// KISSY v1.1.5+
//     val, css, html, text, data, width, height, offset
DOM.create('<a>', { href: 'hot.html',
                    title: 'Hot Page',
                    css: {color: 'blue'},
                    text: 'Test Link'
          });

//
DOM.create('<img src="logo.png" alt="logo" />');
```

DOM.html

DOM.**html**(selector)

获取符合选择器的第一个元素的 innerHTML.

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

Returns 符合选择器的第一个元素的 innerHTML.

Return type string

DOM.**html**(selector, html[, loadScripts, callback])

给符合选择器的所有元素设置 innerHTML 值.

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- html (string) -- 将要设置的 html 值
- loadScripts (boolean) -- 是否执行 html 中的内嵌脚本
- callback (function) -- 操作成功后的回调函数

DOM.remove

DOM.**remove**(selector)

将符合选择器的所有元素从 DOM 中移除。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

DOM.insertBefore

DOM.**insertBefore**(newNode, refNode)
将 newNode 插入到 refNode 之前。
Parameters
- newNode (string|HTMLElement) -- 插入的节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。
- refNode (string|HTMLElement) -- 参考节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。

DOM.insertAfter

DOM.**insertAfter**(newNode, refNode)
将 newNode 插入到 refNode 之后。
Parameters
- newNode (string|HTMLElement) -- 插入的节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。
- refNode (string|HTMLElement) -- 参考节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。

DOM.append

DOM.**append**(node, parent)
将 node 追加到 parent 子节点最后。
Parameters
- node (string|HTMLElement) -- 插入的节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。
- parent (string|HTMLElement) -- 参照父节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。

DOM.prepend

DOM.**prepend**(node, parent)
将 node 追加到 parent 子节点最前。
Parameters
- node (string|HTMLElement) -- 插入的节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。
- parent (string|HTMLElement) -- 参照父节点。字符串格式参见 KISSY selector ，获取匹配的第一个元素。

DOM.data

DOM.**data**(selector[, name])
获取符合选择器的第一个元素的扩展属性（expando）。
Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 扩展属性名称

Returns

- 对应扩展属性名的属性值，如果不存在返回 undefined
- 如不指定扩展属性名，则取得所有扩展属性键值对象，如果当前还没设置过扩展属性，则返回空对象，可以直接在该空对象上设置

Note: embed，object，applet 这三个标签不能设置 expando 。 如果判断是否设置了扩展属性，请使用 hasData()

DOM.**data**(selector, name, data)

给符合选择器的所有元素的扩展属性（expando）。设置扩展属性 name 为 data。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 扩展属性名称
- value -- 扩展属性值

举例

```
var S = KISSY, DOM = S.DOM;

//    img    data-size   expando ,   400;
DOM.data('img', 'data-size', 400);

//     img   ,   data-size   expando  ;
DOM.data('img', 'data-size');

var p=DOM.create("<p>");

DOM.hasData(p); // => false

var store=DOM.data(p); // => store={}

store.x="y"; // =>    DOM.data(p,"x","y");

DOM.removeData(p,"x");

DOM.data(p,"x"); // => undefined

DOM.hasData(p,"x"); // => false

DOM.hasData(p) // => false

DOM.data("p") // =>
```

DOM.removeData

DOM.**removeData**(selector[, name])

将符合选择器的所有元素的对应扩展属性(expando)删除。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 扩展属性名称。如果指定 name，则只删除名为 name 的 expando 。如果不指定 name，则删除元素的整个 expando 。

举例：

DOM.hasData

DOM.**hasData**(selector[, name])

New in version 1.2: 判断是否符合选择器的所有元素中的一个存在对应的扩展属性(expando)值。

Parameters

- selector (string|HTMLCollection|Array<HTMLElement>) -- 字符串格式参见 KISSY selector
- name (string) -- 扩展属性名称。如果指定 name，则判断是否存在指定的扩展属性值。否则判断是否存在任意扩展属性值

Return type boolean

举例：

DOM.unselectable

New in version 1.2.

DOM.**unselectable**(selector)

使符合选择器的所有元素都不可以作为选择区域的开始。

Parameters selector (string| HTMLCollection| Array<HTMLElement>) -- 字符串格式参见 KISSY selector

---

Note: 在 ie 下会引发该元素鼠标点击获取不到焦点，在 firefox 下要得到同样的效果则需要阻止 mousedown 事件。

2.2.2 Node

by 承玉 Changed in version 1.2: Node 包括 DOM 模块的所有功能，推荐采用 Node 模块取代 DOM 模块， 你只需要把 KISSY.all 看做 jquery 中的 $ 就可以了， 链式操作你会喜欢的！ 获取模块

```
//    kissy.js    KISSY.Node/KISSY.NodeList    KISSY.Node=KISSY.NodeList
KISSY.use("node",function(S,Node){
    var NodeList=Node;
});
```

特色方法：

NodeList

class Node.**NodeList**(html[, props, ownerDocument])

Parameters

- string| HTMLElement| Text| Window| HTMLDocument| HTMLCollection| ArrayList<HTMLElement>|NodeList --
  - string：html 字符串，例如 ，根据该字符串生成 NodeList 对象，代表节点个数为 html 字符串实际产生的 DOM 节点个数.
  - HTMLElement|Text|Window|HTMLDocument： 把原生 DOM 节点包装成一个 NodeList 对象，这个情景一般可用 S.all 代替.
  - HTMLCollection|ArrayList<HTMLElement> ： 将原生节点列表包装为一个 NodeList 对象.
  - NodeList：从当前 NodeList 对象中克隆一个新对象返回.
- props (object) -- 属性键值对，对生成的 NodeList 对象代表的原生 DOM 节点设置属性。仅当 html 参数为 html 字符串时使用。
- ownerDocument (HTMLDocument) -- 该 NodeList 产生的原生 DOM 节点所属的文档对象。仅当 html 参数为 html 字符串时使用。

Note: New in version 1.2: 推荐除了需要生成文本节点的情况下，统统使用 all() 代替。

例如：

得到一个包装新文本节点的 KISSY NodeList

```
var nl=new NodeList("aaa");
nl.getDOMNode().nodeType ==3 // => true : Html Text Node
```

得到一个包装新 dom element 的 KISSY NodeList

```
var nl=NodeList.all("<div></div><p></p>");
var domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
```

得到一个包装现有 dom element 的 KISSY NodeList

```
var domNodes=document.getElementsByTagName("div");
var nl=NodeList.all(domNodes);
domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
domNodes[0].nodeType == 1 // => true : Html Element
```

当然 NodeList.all 可以直接获取选择器字符串匹配的节点列表

```
var nl=NodeList.all("div"); //       text node
domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
domNodes[0].nodeType == 1 // => true : Html Element
```

all()

选择器

NodeList.all(selector[, context])
根据选择器字符串得到节点列表
Parameters
- selector (string) -- 选择器字符串
- HTMLELement| Document| NodeList -- 选择器上下文，.. versionadded 1.2 NodeList 时取第一个元素

Return type NodeList

NodeList.all(element)
Parameters element (HTMLElement) -- 包装成 NodeList 类型的原生 dom 节点

Return type NodeList

NodeList.all(elementArray)
Parameters elementArray (Array<HTMLElement>|HTMLCollection) -- 包装成 NodeList 类型的原生 dom 节点集合

Return type NodeList

NodeList.all(nodeList)
Parameters nodeList (NodeList) -- 克隆出一个新的 NodeList 对象

Return type NodeList

在第一种形式中，NodeList.all() 找到所以匹配选择器参数的原生 dom 节点，然后创建一个新的 NodeList 对象来指向这些元素，例如

```
NodeList.all("div.foo");
```

选择器上下文 默认情况下是在文档根节点开始依据选择器字符串开始匹配元素查找。但是一个上下文可以作为可选的第二个参数来限定查找范围，例如在事件处理器 范围内进行查找匹配元素：

```
var $=NodeList.all;
$('div.foo').on("click",function() {
  $('span', this).addClass('bar');
});
```

当对 span 的选择限定在 this 中时，只有位于点击元素内的 span 节点会被设置格外的 class.也可以通过 $(this).all("span") 来实现限定搜索范围.

使用原生 DOM 节点 第二第三种方法使用现有的原生 dom 节点来创建 NodeList 对象。常用的应用场景是从事件处理器的 this 关键字中创建 NodeList 对象

```
var $=NodeList.all;
$('div.foo').on("click",function() {
  $(this).slideUp();
});
```

这个例子会导致元素点击时以滑动的效果隐藏. 因为事件处理器中的 this 默认指向原生 dom 节点，在调用 slideUp 方法前一定要通过 NodeList.all 构建 NodeList 对象.

克隆 NodeList 对象　当一个 NodeList 对象作为参数传递给 NodeList.all 时，会返回该对象的克隆对象。返回的克隆对象和源对象指向同一个原生 dom 节点集合。

创建

NodeList.all(html[, ownerDocument])

Parameters

- html (string) -- 用来得到 dom 节点的 html 字符串
- ownerDocument (HTMLDocument) -- 创建的新节点所在的文档

Return type NodeList

如果一个字符串传递给 Node.all 作为参数，Node.all 会检查该字符粗韩是否像一个html串（例如形式为 <tag ... >）。如果不是，那么字符串参数就会作为一个选择器字符串，进行选择元素操作。如果字符串是一个html片段，那么就会创建相应的 dom 节点，并且返回一个新的 NodeList 对象指向产生的 dom 节点。

Note: 为了确保各个浏览器的兼容性，html 片段必须是结构完整的，例如请包含结束标签：

```
NodeList.all("<a href='http://docs.kissyui.com'></a>");
```

如果创建单个元素不带任何属性和子节点，也可以

..code-block:: javascript

NodeList.all(``<a>'')

例子

得到一个包装新 dom element 的 KISSY NodeList

```
var nl=NodeList.all("<div></div><p></p>");
var domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
```

得到一个包装现有 dom element 的 KISSY NodeList

```
var domNodes=document.getElementsByTagName("div");
var nl=NodeList.all(domNodes);
domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
domNodes[0].nodeType == 1 // => true : Html Element
```

当然 NodeList.all 可以直接获取选择器字符串匹配的节点列表

```
var nl=NodeList.all("div"); //       text node
domNodes=nl.getDOMNodes();
domNodes[0].nodeType == 1 // => true : Html Element
domNodes[0].nodeType == 1 // => true : Html Element
```

找到页面第一个表单的所有输入框

```
NodeList.all("input",document.forms[0]);
```

设置页面的背景色为黑色

```
NodeList.all(document.body).css("background","black");
```

隐藏一个表单内的所有输入框

```
NodeList.all(myForm.elements).hide();
```

one()

NodeList.**one**(args...)

- 如果参数为选择字符串，找不到则返回 null
- 其他情况下等同于 NodeList.all(args...)

Returns null 或者 NodeList 对象

Return type NodeList

例子:

```
NodeList.one("#noexist") // => null

NodeList.one() //   => NodeList.all()
```

add()

New in version 1.2: 参数包含以下形式：

NodeList.**add**(selector[, context])

返回包含合并选择器字符串匹配的元素和当前节点列表元素的新 NodeList 对象

Return type NodeList

NodeList.**add**(element)

Return type NodeList

NodeList.**add**(elementArray)

Return type NodeList

NodeList.**add**(nodeList)

Return type NodeList

参数同 `all()` 保持一致.

Note: 调用该方法并不会改变当前 NodeList 对象.

例子:

```
<p>1</p><div>2</div>

<script>
    var pdiv = $("p");
    var all=pdiv.add("div");  //  pdiv will not change
    all.css("color","red"); // => 1,2
</script>
```

item()

NodeList.item(index)

获取包含当前节点列表 index 位置处的单个原生节点的新 NodeList 对象

Returns null 或者包含一个原生节点的 NodeList 对象

Return type NodeList

例子:

```
<div class='a' id='a'>
</div>
<div class='a' id='b'>
</div>

<script>
    NodeList.all(".a").item(0).attr("id")  // => a
    NodeList.all(".a").item(1).attr("id")  // => b
    NodeList.all(".a").item(2)  // => null
</script>
```

slice()

NodeList.slice(start[, end])

New in version 1.2: 获取包含当前节点列表选定范围内原生节点的新 NodeList 对象

Parameters

- start (number) -- 范围开始位置
- end (number) -- 范围结束位置，忽略的话结束坐标为当前列表末尾

Return type NodeList

Note: 调用该方法并不会改变当前 NodeList 实例

例子:

```
<div class='a' id='a'>
</div>
<div class='a' id='b'>
</div>
<div class='a' id='c'>
</div>
<div class='a' id='d'>
</div>

<script>
    var as=NodeList.all(".a");
    var subs=as.slice(1,3); // => subs != as
    subs.length // => 2
    subs.item(0).attr("id") // => b
    subs.item(1).attr("id") // => c
</script>
```

scrollTop()

NodeList.**scrollTop**()

New in version 1.2: 得到当前节点列表第一个节点的滚动条的垂直位置。

Return type number

例子：

得到一行的 scrollTop

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { margin:10px;padding:5px;border:2px solid #666; }
  </style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
<body>
  <p>Hello</p><p></p>
<script>
    KISSY.use("node,sizzle",function(S,Node){
        var p = Node.all("p:first");
        Node.all("p:last").text( "scrollTop:" + p.scrollTop() );
    });
</script>
</body>
</html>
```

Demo

NodeList.**scrollTop**(value)

New in version 1.2: 设置当前节点列表每个节点的滚动条的垂直位置。

Parameters value (number) -- 新的滚动条所在位置

Return type NodeList

Returns 自身 this

例子：

设置 div 的 scrollTop

```
<!DOCTYPE html>
<html>
<head>
  <style>
div.demo {
background:#CCCCCC none repeat scroll 0 0;
border:3px solid #666666;
margin:5px;
padding:5px;
position:relative;
width:200px;
```

```
height:100px;
overflow:auto;
}
  p { margin:10px;padding:5px;border:2px solid #666;width:1000px;height:1000px; }
  </style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
<body>
  <div class="demo"><h1>lalala</h1><p>Hello</p></div>
<script>
    KISSY.use("node",function(S,Node){
        Node.all("div.demo").scrollTop(300);
    });
</script>

</body>
</html>
```

Demo

scrollLeft()

`NodeList.scrollLeft()`
New in version 1.2: 得到当前节点列表第一个节点的滚动条的横向位置。
Return type number

`NodeList.scrollLeft(value)`
New in version 1.2: 设置当前节点列表每个节点的滚动条的横向位置。
Parameters value (number) -- 新的滚动条所在位置
Return type NodeList
Returns 自身 this

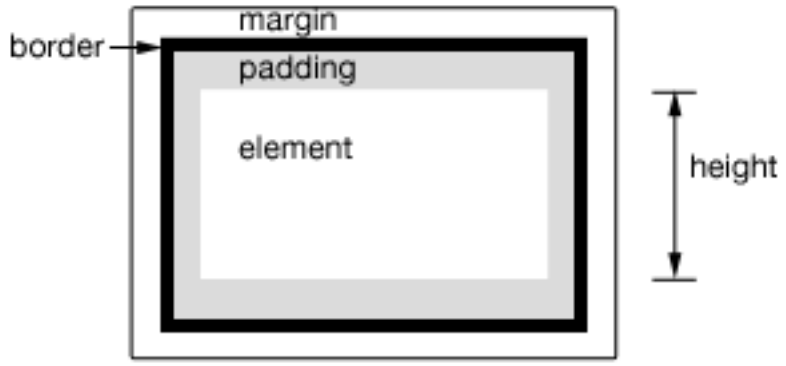
例子可参考： `scrollTop()`

height()

读

`NodeList.height()`
得到当前节点列表第一个节点的计算高度
Return type number

和 `css('height')` 的区别在于该函数返回不带单位的纯数值，而前者则返回带单位的原始值（例如 `400px` ）．当需要数值计算时，推荐该方法，如图所示：(from jquery)



New in version 1.2: 该方法也可以用来得到 windw 和 document 的高度

```
KISSY.use("node",function(S,Node){
    Node.all(window).height(); //           DOM.viewportHeight()
    Node.all(document).height(); //  html       DOM.docHeight()
});
```

例子　得到各种各样的高度，黄色高亮区域代表 iframe 体。

```
<!DOCTYPE html>
<html>
<head>
  <style>
  body { background:yellow; }
  button { font-size:12px; margin:2px; }
  p { width:150px; border:1px red solid; }
  div { color:red; font-weight:bold; }
  </style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
<body>
  <button id="getp">Get Paragraph Height</button>
  <button id="getd">Get Document Height</button>
  <button id="getw">Get Window Height</button>

  <div> </div>
  <p>
    Sample paragraph to test height
  </p>
<script>
    KISSY.use("node",function(S,Node){
        var $=Node.all;
        function showHeight(ele, h) {
          $("div").text("The height for the " + ele +
                        " is " + h + "px.");
        }
        $("#getp").on("click",function () {
          showHeight("paragraph", $("p").height());
```

```
        });
        $("#getd").on("click",function () {
          showHeight("document", $(document).height());
        });
        $("#getw").on("click",function () {
          showHeight("window", $(window).height());
        });
    });
</script>

</body>
</html>
```

效果

写

NodeList.**height**(value)

设置当前列表每个元素的 css height 值。

Parameters value (number|string) -- 代表像素的整数值，或数字加上其他单位的字符串值.

Note: 在现代浏览器中，css height 属性不包括 padding，border 或者 margin.

例子

```
<!DOCTYPE html>
<html>
<head>
  <style>div { width:50px; height:70px; float:left; margin:5px;
        background:rgb(255,140,0); cursor:pointer; }  </style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
<body>
  <div></div>
  <div></div>

  <div></div>
  <div></div>
  <div></div>
<script>
    KISSY.use("node",function(S,Node){
        var $=Node.all;
        function handle(){
            $(this).detach();
            $(this).height(30)
                 .css({cursor:"auto", backgroundColor:"green"});
        }
```

```
        $("div").on('click', handle);
    });
</script>

</body>
</html>
```

效果

width()

读

NodeList.**width**()

得到当前节点列表第一个节点的计算宽度

Return type number

和 `css('width')` 的区别在于该函数返回不带单位的纯数值，而前者则返回带单位的原始值（例如 `400px`）. 当需要数值计算时，推荐该方法。 New in version 1.2: 该方法也可以用来得到 windw 和 document 的宽度

```
KISSY.use("node",function(S,Node){
    Node.all(window).width(); //          DOM.viewportWidth()
    Node.all(document).width(); //  html       DOM.docWidth()
});
```

写

NodeList.**width**(value)

设置当前列表每个元素的 css width 值。

Parameters value (number|string) -- 代表像素的整数值，或数字加上其他单位的字符串值.

Note: 在现代浏览器中，css width 属性不包括 padding，border 或者 margin.

例子可参考 `NodeList.height()`

addStyleSheet()

NodeList.**addStyleSheet**(cssText[, id])

New in version 1.2: 将 cssText 字符串作为内联样式添加到文档中。

Parameters

- cssText (string) -- 样式内容
- id (string) -- 内联样式所在 style 节点的 id

Return type NodeList

Returns 当前对象

Note: 该方法只可以在 window 和 document 上调用，例如：

```
KISSY.use("node",function(S,Node){
    var $=Node.all;
    $(window).addStyleSheet("p {color:red;}"); //
    //
    $(document).addStyleSheet("p {color:red;}","addCss");
});
```

---

append()

NodeList.**append**(content)

将参数内容插入到当前节点列表中的每个元素的末尾.

Parameters content (HTMLELement|string|NodeList) -- 将要插入的内容

Return type NodeList

该方法插入指定内容到当前节点列表的最后一个元素后面（如果要插入到第一个元素前面，请用 prepend() ）.

该方法和 appendTo() 功能一样。最大的区别在于语法不同以及参数意义不同。当使用 append 方法时，当前节点列表为参数内容的插入容器。 而对于 appendTo 当前节点列表则为要插入的元素，而参数则为目标容器.

例如如下 HTML:

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

你可以创建 NodeList 并把它立即插入到指定容器中:

```
KISSY.use("node",function(S,NodeList){
    NodeList.all('.inner').append('<p>Test</p>');
});
```

内层的每个 div 元素都得到了新内容

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    Hello
    <p>Test</p>
  </div>
  <div class="inner">
    Goodbye
    <p>Test</p>
  </div>
</div>
```

你可以把页面上已有的元素 prepend 到另外一个:

```
KISSY.use("node",function(S,NodeList){
    NodeList.all('.container').append($('h2'));
});
```

如果当前节点列表只包括一个节点，那么他将会被移到目标容器中（而不是克隆）：

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
  <h2>Greetings</h2>
</div>
```

但是如果当前节点列表包括多余一个节点，则除了第一个节点外，其他节点都添加的是参数节点的克隆节点。

例子

在所有段落中添加一些 html 字符串

```
    <!DOCTYPE html>
    <html>
    <head>
      <style>
      p { background:yellow; }
    </style>
     <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
    </head>
    <body>
      <p> is I would like to say </p>
    <script>
        KISSY.use("node",function(S,NodeList){
            NodeList.all("p").append("<strong>Hello</strong>");
        });
    </script>

    </body>
    </html>
```

Demo

给所有段落添加一个文本节点

```
    <!DOCTYPE html>
    <html>
    <head>
      <style>
      p { background:yellow; }
    </style>
     <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
```

```
</head>
<body>
  <p>I would like to say: </p>
  <p>I also would like to say: </p>
<script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("p").append(document.createTextNode("Hello"));
    });
</script>

</body>
</html>
```

Demo

给所有段落添加一个 NodeList 对象

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
 </head>
 <body>
    <strong>Hello world!!!</strong><p>I would like to say: </p>

<script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("p").append(NodeList.all("strong"));
    });
</script>

</body>
</html>
```

Demo

appendTo()

NodeList.**appendTo**(containers)

将当前节点列表中的每个元素插入到容器列表的每个元素的最后一个子节点后面.

Parameters content (HTMLELement|string|NodeList) -- 将要插入的内容

- HTMLELement|NodeList: 已有或新创建的节点
- string: 选择器字符串，查找已有的容器节点

Return type NodeList

appendTo 和 append() 功能一样，只不过参数意义不同.

考虑下面 html 字符串:

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建元素后立即插入到多个已有元素:

```
NodeList.all('<p>Test</p>').appendTo('.inner');
```

每个内层 div 元素都得到了新内容

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    Hello
    <p>Test</p>
  </div>
  <div class="inner">
    Goodbye
    <p>Test</p>
  </div>
</div>
```

我们也可以把一个已有元素插入到另一个

```
NodeList.all('h2').appendTo(NodeList.all('.container'));
```

如果容器列表只有一个节点，那么当前节点列表会被移动到容器内(不是克隆):

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
  <h2>Greetings</h2>
</div>
```

不过如果有多个目标容器，那么除了第一个目标容器，当前节点列表的复制节点会被插入到其他目标容器

例子

把多个 span 插入到已有元素

```
<!DOCTYPE html>
<html>
<head>
  <style>div { background:yellow; }</style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
```

```
<body>
    <div id="foo">FOO! is </div>

    <span>I have something to say... </span>

    <span> once more : </span>

    <script>
        KISSY.use("node",function(S,NodeList){
            NodeList.all("span").appendTo(NodeList.all("#foo"));
        });
    </script>
</body>
</html>
```

Demo

prepend()

NodeList.**prepend**(content)

将参数内容插入到当前节点列表中的每个元素的开头.

Parameters content (HTMLELement|string|NodeList) -- 将要插入的内容

Return type NodeList

该方法插入指定内容到当前节点列表的第一个元素前面（如果要插入到最后一个元素后面，请用 append() ）.

该方法和 prependTo() 功能一样。最大的区别在于语法不同以及参数意义不同。当使用 prepend 方法时，当前节点列表为参数内容的插入容器。 而对于 prependTo 当前节点列表则为要插入的元素，而参数则为目标容器.

例如如下 HTML:

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

你可以创建 NodeList 并把它立即插入到指定容器中:

```
KISSY.use("node",function(S,NodeList){
    NodeList.all('.inner').prepend('<p>Test</p>');
});
```

内层的每个 div 元素都得到了新内容

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    <p>Test</p>
    Hello
  </div>
  <div class="inner">
    <p>Test</p>
    Goodbye
```

```
  </div>
</div>
```

你可以把页面上已有的元素 prepend 到另外一个:

```
KISSY.use("node",function(S,NodeList){
    NodeList.all('.container').prepend($('h2'));
});
```

如果当前节点列表只包括一个节点，那么他将会被移到目标容器中（而不是克隆）：

```
<div class="container">
    <h2>Greetings</h2>
    <div class="inner">Hello</div>
    <div class="inner">Goodbye</div>
</div>
```

但是如果当前节点列表包括多余一个节点，则除了第一个节点外，其他节点都添加的是参数节点的克隆节点。

例子

在所有段落中添加一些 html 字符串

```
    <!DOCTYPE html>
    <html>
    <head>
      <style>
      p { background:yellow; }
    </style>
     <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
    </head>
    <body>
      <p>I would like to say: </p>
    <script>
        KISSY.use("node",function(S,NodeList){
            NodeList.all("p").prepend("<strong>Hello</strong>");
        });
    </script>

    </body>
    </html>
```

Demo

给所有段落添加一个文本节点

```
    <!DOCTYPE html>
    <html>
    <head>
```

```
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
 </head>
 <body>
  <p> is I would like to say </p>
  <p> is also I would like to say</p>
 <script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("p").prepend(document.createTextNode("Hello"));
    });
 </script>

 </body>
 </html>
```

Demo

给所有段落添加一个 NodeList 对象

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
 </head>
 <body>
   <p> is I would like to say </p> <strong>Hello world!!!</strong>

 <script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("p").prepend(NodeList.all("strong"));
    });
 </script>

 </body>
 </html>
```

Demo

prependTo()

NodeList.**prependTo**(containers)

将当前节点列表中的每个元素插入到容器列表的每个元素的开头.

Parameters content (HTMLELement|string|NodeList) -- 将要插入的内容

- HTMLELement|NodeList: 已有或新创建的节点
- string: 选择器字符串，查找已有的容器节点

Return type NodeList

prependTo 和 prepend() 功能一样，只不过参数意义不同.

考虑下面 html 字符串:

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建元素后立即插入到多个已有元素:

```
NodeList.all('<p>Test</p>').prependTo('.inner');
```

每个内层 div 元素都得到了新内容

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    <p>Test</p>
    Hello
  </div>
  <div class="inner">
    <p>Test</p>
    Goodbye
  </div>
</div>
```

我们也可以把一个已有元素插入到另一个

```
NodeList.all('h2').prependTo(NodeList.all('.container'));
```

如果容器列表只有一个节点，那么当前节点列表会被移动到容器内(不是克隆):

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

不过如果有多个目标容器，那么除了第一个目标容器，当前节点列表的复制节点会被插入到其他目标容器

例子

把多个 span 插入到已有元素

```
<!DOCTYPE html>
<html>
<head>
  <style>div { background:yellow; }</style>
  <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
</head>
<body>
    <div id="foo">FOO!</div>

    <span>I have something to say... </span>

    <span> once more : </span>

    <script>
        KISSY.use("node",function(S,NodeList){
            NodeList.all("span").prependTo(NodeList.all("#foo"));
        });
    </script>
</body>
</html>
```

Demo

insertBefore()

NodeList.**insertBefore**(target)

将当前列表中的每个元素插入到目标元素之前.

Parameters target (HTMLElement|string|NodeList) -- 将要插入的元素

- string：选择器字符串
- HTMLElement|NodeList：已有或新建的元素

before() 和该方法的功能一样，只不过参数意义不同，该函数表示当前节点列表被插入到参数目标节点之前， 而 before 则表示参数节点被插入到当前节点之前.

例如:

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建节点并立即把它插入到一些元素之前

```
NodeList.all('<p>Test</p>').insertBefore('.inner');
```

结果为

```
<div class="container">
  <h2>Greetings</h2>
  <p>Test</p>
  <div class="inner">Hello</div>
  <p>Test</p>
  <div class="inner">Goodbye</div>
</div>
```

我们也可以操纵现有元素

```
NodeList.all('h2').insertBefore(NodeList.all('.container'));
```

如果目标节点只有一个，那么当前节点就会移动到目标节点之前

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

如果有多个目标节点，那么除了第一个目标节点外，其他目标节点前会被插入当前节点的克隆

例子

把段落插入到 div 节点之前

```
    <!DOCTYPE html>
    <html>
    <head>
      <style>
      p { background:yellow; }
    </style>
     <script>
    if(location.host=="localhost"){
        document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }else{
        document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
    }
   </script>
    </head>
   <body>
  <div id="foo">FOO!</div><p>I would like to say: </p>

    <script>
        KISSY.use("node",function(S,NodeList){
            NodeList.all("p").insertBefore("#foo");
        });
    </script>

    </body>
    </html>
```

Demo

before()

NodeList.**before**(content)

Changed in version 1.2: 将参数内容插入到当前列表中每个元素之前.

Parameters content (HTMLElement|string|NodeList) -- 将要插入的元素

- string：html 字符串
- HTMLElement|NodeList：已有或新建的元素

insertBefore() 和该方法的功能一样，只不过参数意义不同，``insertBefore`` 表示当前节点列表被插入到参数目标节点之前， 而该方法则表示参数节点被插入到当前节点之前.

例如:

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建节点并立即把它插入到一些元素之前

```
NodeList.all('.inner').insertBefore('<p>Test</p>');
```

结果为

```
<div class="container">
  <h2>Greetings</h2>
  <p>Test</p>
  <div class="inner">Hello</div>
  <p>Test</p>
  <div class="inner">Goodbye</div>
</div>
```

我们也可以操纵现有元素

```
NodeList.all('.container').before(NodeList.all('h2'));
```

如果目标节点只有一个，那么当前节点就会移动到目标节点之前

```
<h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

如果有多个目标节点，那么除了第一个目标节点外，其他目标节点前会被插入当前节点的克隆

例子

把段落插入到 div 节点之前

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
 </head>
<body>
<div id="foo">FOO!</div><p>I would like to say: </p>

<script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("#foo").before(NodeList.all("p"));
    });
</script>

</body>
</html>
```

Demo

after()

NodeList.**after**(content)

Changed in version 1.2: 将参数内容插入到当前列表中每个元素之后.

Parameters content (HTMLElement|string|NodeList) -- 将要插入的元素

- string：html 字符串
- HTMLElement|NodeList：已有或新建的元素

insertAfter() 和该方法的功能一样，只不过参数意义不同， insertAfter 表示当前节点列表被插入到参数目标节点之后， 而该方法则表示参数节点被插入到当前节点之后.

例如:

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建节点并立即把它插入到一些元素之前

```
NodeList.all('.inner').after('<p>Test</p>');
```

结果为

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
```

```
  <p>Test</p>
  <div class="inner">Goodbye</div>
  <p>Test</p>
</div>
```

我们也可以操纵现有元素

```
NodeList.all('.container').after(NodeList.all('h2'));
```

如果目标节点只有一个，那么当前节点就会移动到目标节点之后

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

如果有多个目标节点，那么除了第一个目标节点外，其他目标节点前会被插入当前节点的克隆

例子

把段落插入到 div 节点之前

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
</head>
<body>

<p>I would like to say: </p>  <div id="foo">FOO!</div>

<script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("#foo").after(NodeList.all("p"));
    });
</script>

</body>
</html>
```

Demo

insertAfter()

NodeList.**insertAfter**(target)

将当前列表中的每个元素插入到目标元素之后.

Parameters target (HTMLElement|string|NodeList) -- 将要插入的元素

- string：选择器字符串
- HTMLElement|NodeList：已有或新建的元素

`after()` 和该方法的功能一样，只不过参数意义不同，该函数表示当前节点列表被插入到参数目标节点之后， 而 `after` 则表示参数节点被插入到当前节点之后.

例如:

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

我们可以创建节点并立即把它插入到一些元素之前

```
NodeList.all('<p>Test</p>').insertAfter('.inner');
```

结果为

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <p>Test</p>
  <div class="inner">Goodbye</div>
  <p>Test</p>
</div>
```

我们也可以操纵现有元素

```
NodeList.all('h2').insertAfter(NodeList.all('.container'));
```

如果目标节点只有一个，那么当前节点就会移动到目标节点之前

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

如果有多个目标节点，那么除了第一个目标节点外，其他目标节点前会被插入当前节点的克隆

例子

把段落插入到 div 节点之前

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { background:yellow; }
</style>
 <script>
if(location.host=="localhost"){
    document.writeln('<script src="http://localhost/kissy_git/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}else{
    document.writeln('<script src="http://yiminghe.github.com/kissy/build/kissy.js"'+'>'+'<'+'/'+'script>');
}
</script>
 </head>
<body>
<p>I would like to say: </p><div id="foo">FOO!</div>

<script>
    KISSY.use("node",function(S,NodeList){
        NodeList.all("p").insertAfter("#foo");
    });
</script>

</body>
</html>
```

Demo

转发方法:

该类方法调用会被转发给 DOM ，原 DOM 对应方法的第一个参数传入一个原生 DOM 节点数组，而这个原生 DOM 节点数组则是由当前的 KISSY NodeList 对象得到的。

Node 模块会对返回值进行处理:

- 如果返回值为单个节点或节点数组则包装为 NodeList
- 如果返回值为 undefined ，则返回调用者 NodeList 对象
- 其他，直接返回

具体方法包括：

- filter()
- test()
- hasClass()
- addClass()
- removeClass()
- replaceClass()
- toggleClass()
- removeAttr()
- attr()
- hasAttr()
- val()

- text()
- css()
- show()
- hide()
- toggle()
- offset()
- scrollIntoView()
- parent()
- next()
- prev()
- siblings()
- children()
- contains()
- html()
- remove()
- data()
- removeData()
- hasData()
- unselectable()

2.2.3 Anim

by 承玉

获取动画构造器

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

通过 use 加载 anim 模块：

```
KISSY.use("anim",function(){
    var Anim=S.Anim;
    //  Anim
});
```

New in version 1.2: KISSY 1.2 可直接通过依赖注入，从函数参数中取得

```
KISSY.use("anim",function(S,Anim){
    //  Anim
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器接口

class Anim.**Anim**(elem, props[, duration=1, easing='easeNone', callback, nativeSupport=true])
得到绑定于某个 dom 节点的动画实例
Parameters

- props (object) -- 动画结束的 dom 样式值，例如

```
{
    width:"100px",
    height:"100px"
}
```

表示节点将从当前宽高经过动画平滑变化到宽 100px 与高 100px

Note: 也可以设置 scrollLeft 或者 scrollTop，这时会直接对元素的滚动属性产生动画。

- elem (选择器字符串或是通过 S.get 获得的原生 dom 节点) -- 作用动画的元素节点
- duration (number) -- 动画持续时间，以秒为单元
- easing (string) -- 动画平滑函数，可取值 ``easeNone'',''easeIn'',''easeOut'',''easeBoth'',''easeInStrong'', ``easeOutStrong'',''easeBothStrong'',''elasticIn'',''elasticOut'', ``elasticBoth'',''backIn'',''backOut'',''backBoth'', ``bounceIn'',''bounceOut'',''bounceBoth''

  效果预览，可以参考 Robert Penner 博士的： easing_demo.html
- callback (function) -- 动画结束回调
- nativeSupport (boolean) -- 是否在支持css动画的浏览器上使用原生机制

实例动画对象

通过 var anim=Anim(...) 来实例化一个动画对象。

实例方法

Anim.**run**()
在动画实例上调用，开始当前动画实例的动画.

Anim.**stop**(finish=false)
在动画实例上调用，结束当前动画实例的动画.
Parameters finish (boolean) -- flasy 时，动画会在当前帧直接停止；为 true 时，动画停止时会立刻跳到最后一帧。

在节点实例上开始动画

Node.**animate**(props[, duration=1, easing='easeNone', callback, nativeSupport=true])
在当前节点作用动画

Node.**stop**(finish)
New in version 1.2: 停止在当前节点作用动画
Parameters finish (boolean) -- flasy 时，动画会在当前帧直接停止；为 true 时，动画停止时会立刻跳到最后一帧。

参数可见 接口部分

Node.**show**([speed, callback])
元素以动画效果显示
Parameters
- speed (number) -- 动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**hide**([speed, callback])
元素以动画效果隐藏
Parameters
- speed (number) -- 动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**toggle**([speed, callback])
当前元素为显示时动画效果隐藏，否则动画效果显示
Parameters
- speed (number) -- 动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**fadeIn**([speed=1, callback])
元素渐隐效果显示
Parameters
- speed (number) -- 单位秒，动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**fadeOut**([speed=1, callback])
元素渐隐效果隐藏
Parameters
- speed (number) -- 单位秒，动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**slideDown**([speed=1, callback])
元素从上到下滑动显示
Parameters
- speed (number) -- 单位秒，动画持续时间，设置无动画
- callback (function) -- 动画结束后回调函数

Node.**slideUp**([speed=1, callback])
元素从下到上隐藏
Parameters
- speed (number) -- 单位秒，动画持续时间，设置无动画

- callback (function) -- 动画结束后回调函数

2.3 Component

2.3.1 DataLazyload

数据延迟加载组件.

构造器接口

class DataLazyload.**DataLazyload**(containers, config)

Parameters

- containers (object) -- array, 图片所在容器(可以多个), 默认为 [doc]
- config (object) -- 类型对象, 实例对象所需的配置, 详见下节列出的各项

```
S.DataLazyload( cfg );
```

配置项

DataLazyload.**mod**

(optional): {String} 懒处理模式, 默认是 `manul', 可取:

- `auto' : 自动化. html 输出时, 不对 img.src 做任何处理
- `manul' : 输出 html 时, 已经将需要延迟加载的图片的 src 属性替换为 IMG_SRC_DATA

> Attention: 对于 textarea 数据, 只有手动模式

DataLazyload.**diff**

(optional): {Number} 当前视窗往下, diff px 外的 img/textarea 延迟加载, 适当设置此值, 可以让用户在拖动时感觉数据已经加载好, 默认为当前视窗高度(两屏以外的才延迟加载),

DataLazyload.**placeholder**

(optional): {String} 图像的占位图, 默认无

DataLazyload.**execScript**

(optional): {Boolean} 是否执行 textarea 里面的脚本, 默认为 true

实例属性

DataLazyload.**containers**

(读写) : {Array} 图片所在容器(可以多个), 默认为 [doc]

DataLazyload.**config**

(读写) : {Object} 配置参数

DataLazyload.**images**

(只读) : {Array<String>} 需要延迟下载的图片列表

self.**areaes**

(只读) : {Array<String>} 需要延迟处理的 textarea列表

DataLazyload.**callbacks**
(只读): {Object} 和延迟项绑定的回调函数, 元素列表和函数列表一一对应

DataLazyload.**threshold**
(只读): {Number} 需要开始延迟的 Y 坐标值

实例方法

DataLazyload.**addCallback**(el, fn)
添加回调函数. 当 el 即将出现在视图中时, 触发 fn

DataLazyload.**loadCustomLazyData**(containers, type)
static, 加载自定义延迟数据

2.3.2 Overlay

by 承玉

父类 Overlay

获取构造器

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

通过 use 加载 overlay 模块：

```
KISSY.use("overlay",function(){
    var Overlay=S.Overlay;
    //  Overlay
});
```

New in version 1.2: KISSY 1.2 可直接通过依赖注入，从函数参数中取得

```
KISSY.use("overlay",function(S,Overlay){
    //  Overlay
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器接口

class Overlay.**Overlay**(config)
Parameters config (object) -- 类型对象，实例对象所需的配置
例如一个简单的配置项：

```
{
    width:"200px",
    render:"#container"
}
```

config 配置项详解

Overlay.**prefixCls**
(optional): kissy 1.2 新增，类型字符串，默认值为 ``ks-``，样式类名前缀，如默认弹出层根元素会加上样式类：''ks-overlay''， kissy 1.2 版本以前设置无效，都为 ``ks-''.

Overlay.**srcNode**
(optional): 类型选择器字符串，取第一个节点作为弹出层的根节点，例如设置

```
{
    srcNode : "#overlay_test"
}
```

作用于页面

```
<div id='overlay_test'>

</div>
```

则会把 overlay_test 转化为弹出层根节点。当不设置时表示新建一个 HTMLElement 插入到页面中.

Overlay.**width**
(optional): 类型字符串或者整数，弹出层宽度。整数表示单元为 px。

Overlay.**height**
(optional): 类型字符串或者整数，弹出层高度。整数表示单元为 px。

Overlay.**elCls**
(optional): 类型字符串，将要添加到弹出层根元素的样式类。

Overlay.**content**
(optional): 类型字符串，设置弹出层的内容 html。

Overlay.**zIndex**
(optional): 类型整数，设置弹出层的 z-index css属性值。默认 9999。

Overlay.**x**
(optional): 类型整数，设置弹出层相对于文档根节点的 x 坐标。

Overlay.**y**
(optional): 类型整数，设置弹出层相对于文档根节点的 y 坐标。

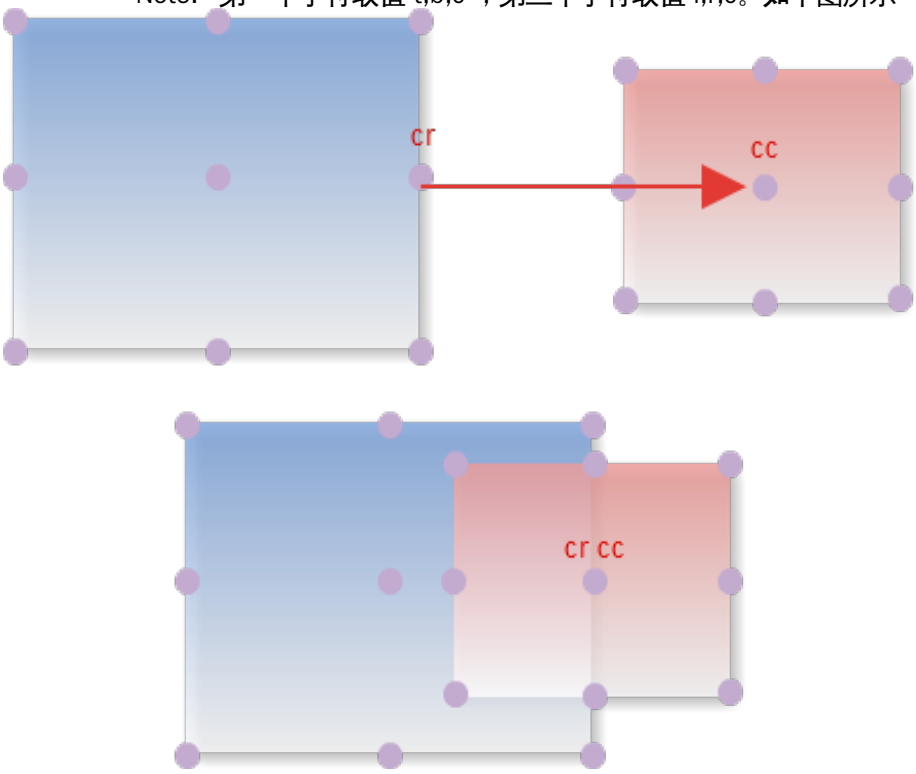
Overlay.**xy**
(optional): 类型整数数组，相当于将数组第一个元素设置为 x 的值，将数组的第二个元素设置为 y 的值。

Overlay.**align**
(optional): 类型对象，弹出层对齐的相关配置，例如

```
{
    node: null,         //          falsy
    points: ['tr','tl'], //       overlay   tl     tr
    offset: [0, 0]      //       overlay     node   points
                        //       x        y
}
```

points 字符串数组元素的取值范围为 t,b,c 与 l,r,c 的两两组合，分别表示 top,bottom,center 与 left,right,center 的两两组合， 可以表示 9 种取值范围。

Note: 第一个字符取值 t,b,c ，第二个字符取值 l,r,c。如下图所示



`Overlay.effect`
New in version 1.2: 类型对象。显示或隐藏时的特效支持，包括以下属性：

`Overlay.effect`
类型string，可取值 `none` （无特效），`fade` （渐隐显示），`slide` （滑动显示）。默认 `none`

`Overlay.easing`
同 `Anim` 的 easing 参数配置。

`Overlay.duration`
类型 number，动画持续时间，以秒为单元

`Overlay.resize`
New in version 1.2: (optional): 类型对象，拖动调整大小的配置，例如：

```
{
    minWidth:100, //
    maxWidth:1000, //
    minHeight:100, //
    maxHeight:1000, //
    handlers:["b","t","r","l","tr","tl","br","bl"] //       8
}
```

`handlers` 配置表示的数组元素可取上述八种值之一，t,b,l,r 分别表示 top,bottom,left,right，加上组合共八种取值， 可在上，下，左，右以及左上，左下，右上，右下进行拖动。

实例属性

当根据配置实例化 overlay 得到当前实例后，可调用实例上的 get 方法得到实例的特定属性以及 set 方法设置属性的值，例如

```
var o = new Overlay({ xy : [400,200] });
//alert
alert(o.get("xy"));
o.set("xy",[100,200]);
//alert
alert(o.get("xy"));
```

可获取属性列表

`Overlay.x`
（读写）：相对于页面绝对横坐标，类型参见配置

`Overlay.y`
（读写）：相对于页面绝对纵坐标，类型参见配置

`Overlay.xy`
（读写）：相当与一次同时读写 `x` 和 `y` 属性，类型参见配置

`Overlay.align`
（读写）：弹出层的对齐信息，类型参见配置

`Overlay.visible`
（读写）：弹出层的显示与否，类型 boolean

`Overlay.el`
（只读）：获取弹出层的根节点，类型 `KISSY.Node` 。

Note: 必须在调用 `render()` 方法之后才可以获取

`Overlay.contentEl`
（只读）：获取弹出层真正内容所在的节点，类型 `KISSY.Node` 。

Note: 必须在调用 `render()` 方法之后才可以获取。

弹出层的 html 结构如下

```
<div><!--      -->
    <div><!--      --->
        <!--       -->
    </div>
</div>
```

一般调用弹出层的 `render()` 方法后，可通过获取 `contentEl` 属性获取内容所在节点，来动态修改弹出层的内容。

实例方法

`Overlay.render()`
渲染当前实例，生成对应的 dom 节点并添加到页面文档树中。

Note: 取 el 与 contentEl 属性值前必须调用过该方法。

Overlay.**show**()
显示弹窗，位置根据 align 或者 xy 确定。

Overlay.**hide**()
隐藏弹窗

Overlay.**align**(node, points, offset)
Parameters
- node (string|KISSY.Node|HTMLDOMNode) -- 类型对齐的参考元素
- points (Array<string>) -- 对齐的参考位置
- offset (Array<number>) -- 相对对齐元素的偏移

相当于调用

```
set("align",{
    node: node,
    points: points,
    offset: offset
});
```

Note: 调用该方法前请先调用 render().

Overlay.**center**()
将弹出层放在当前视窗中央。

Note: 调用该方法前请先调用 render().

Overlay.**move**(x, y)
Parameters
- x (number) -- 相对文档左上角横坐标
- y (number) -- 相对文档左上角纵坐标

相当于调用

```
set("xy",[x,y]);
```

触发事件

Overlay.**hide**
当弹出层隐藏时触发

Overlay.**show**
当弹出层显示时触发

Overlay.**beforeVisibleChange**
当弹出层隐藏或显示前触发，传给事件处理函数的参数为一个对象，格式如下：

```
{
    newVal : //   boolean     false     true
    prevVal : //   boolean
}
```

Note: 当该事件的函数处理器返回 false ，则会阻止将要进行的显示或隐藏动作。

子类 Dialog

获取对话框构造器

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

Dialog 属于 overlay 模块，通过 use 加载 overlay 模块：

```
KISSY.use("overlay",function(){
    var Dialog=S.Dialog;
    //  Dialog
});
```

New in version 1.2: KISSY 1.2 可直接通过依赖注入，从函数参数中取得

```
KISSY.use("overlay",function(S,Overlay){
    //  Overlay
    var Dialog=Overlay.Dialog;
});
```

构造器接口

class Overlay.**Dialog**(config)

Parameters config (object) -- 类型对象，实例对象所需的配置

例如一个简单的配置项：

```
{
    width:"200px",
    render:"#container"
}
```

对话框的 DOM 结构

```
<div> <!--      -->
    <div> <!--      -->
        <div> <!--      -->
        </div>

        <div> <!--      -->
        </div>
```

```
        <div> <!--      -->
        </div>
    </div>
</div>
```

config 配置项详解　除了 content 配置项外与 Overlay 的配置项完全相同，但是 Dialog 新增了一些配置项，如下所示：

Overlay.**headerContent**
　类型字符串，对话框的标题 html.

Overlay.**bodyContent**
　类型字符串，对话框的体 html。

Overlay.**footerContent**
　类型字符串，对话框的底部 html。

Overlay.**closable**
　类型 boolean，对话框右上角是否包括关闭按钮

Overlay.**draggable**
　类型 boolean，是否允许拖动头部移动，注意启用时需同时 use("dd") ，例如

```
KISSY.use("dd,overlay",function(S,DD,Overlay){
    new Overlay.Dialog({
        draggable : true
    });
});
```

Overlay.**aria**
　New in version 1.2: 类型 boolean，是否开启 aria 支持。默认 false 。开启后，窗口显示出来时自动获得焦点并且 tab 键只能在窗口内部转移焦点。

Overlay.**constrain**
　类型 boolean 或者选择器字符串 ，和 draggable 配合，限制拖动的范围，
　　取值 true 时，只能在当前视窗范围内拖动。
　　取值选择器字符串时，则在限制拖动范围为根据该选择器字符串取到的第一个节点所在区域。
　　取值 false 时，可任意移动，例如：

```
KISSY.use("dd,overlay",function(S,DD,Overlay){
    new Overlay.Dialog({
        draggable : true,
        contrain:true //
    });
});

KISSY.use("dd,overlay",function(S,DD,Overlay){
    new Overlay.Dialog({
        draggable : true,
        contrain:"#container" //      container
    });
});
```

实例方法

同 Overlay 。

实例属性

对话框 Dialog 实例可以获得弹出层 Overlay 实例的所有属性，除此之外还有：

`Overlay.header`
: （只读）类型 `KISSY.Node` ，获得对话框的头部节点。

`Overlay.body`
: （只读）类型 `KISSY.Node` ，获得对话框的体部节点。

`Overlay.footer`
: （只读）：类型 `KISSY.Node` ，获得对话框的底部节点。

Note: 以上三个属性在获取前必须调用过 `render()` 方法。

`Overlay.closable`
: （读写）同相应配置项，设置右上角拖放区域有无。

`Overlay.draggable`
: （读写）同相应配置项，设置头部是否可以拖放。

`Overlay.constrain`
: （读写）同相应配置项，设置拖放区域范围。

触发事件

同弹出层 `Overlay` ，包括 `show` , `hide` , `beforeVisibleChange` .

子类 Popup

获取对话框构造器

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

Popup 属于 overlay 模块, 通过 use 加载 overlay 模块：

```
KISSY.use("overlay",function(){
    var Popup = S.Popup;
    //   Popup
});
```

New in version 1.2: KISSY 1.2 可直接通过依赖注入, 从函数参数中取得

```
KISSY.use("overlay",function(S,Overlay){
    //  Popup
    var Popup = Overlay.Popup;
});
```

构造器接口

class Overlay.Popup(container, config)

Parameters container (object) -- 容器对象, 可取:

- 选择器字符串, 此时会取第一个节点作为弹出层的根节点;
- HTMLElement 或直接传入 KISSY Node 对象;
- 也可不设置, 此时, 表示新建 HTMLElement;

param object config 类型对象, 实例对象所需的配置

例如一个简单的配置项：

```
{
    width:"200px",
    render:"#container"
}
```

config 配置项详解 与 Overlay 的配置项完全相同, 除此之外, 还有：

Overlay.trigger

指定触发器, 类型为 选择器字符串 或 HTMLElement 或 KISSY Node 对象

```
KISSY.use("dd,overlay",function(S,DD,Overlay){
    new Overlay.Dialog({
        draggable : true,
        contrain:true //
    });
});

KISSY.use("dd,overlay",function(S,DD,Overlay){
    new Overlay.Dialog({
        draggable : true,
        contrain:"#container" //      container
    });
});
```

Overlay.triggerType

触发类型, 可取 `click', `mouse', 默认为 `click'.

- 取 `click' 时, 表示当点击触发器元素时, 显示弹出层;
- 取 `mouse' 时, 表示当鼠标移入触发器元素时, 显示弹出层, 当鼠标离开触发器元素时, 隐藏弹出层;

实例方法

同 Overlay.

实例属性

同 Overlay.

触发事件

同 Overlay, 包括 show , hide , beforeVisibleChange .

2.3.3 Switchable

by 玉伯

Important: 该模块由 Switchable 基础类、插件和 Widget 类组成。Switchable 基础类抽象了切换的基本操作，通过插件机制实现了自动播放、循环、切换效果、延迟加载、倒计时动画等扩展功能，最后封装成各个 Widget 类，让用户能简明快速地调用。

父类 Switchable

Hint: Switchable 是核心类, S.Tabs/S.Slide/S.Accordion/S.Carousel 都是扩展自它.

获取构造器

页面引入 kissy.js :

```
<script src='kissy.js'></script>
```

通过 use 加载 Switchable 模块：

```
KISSY.use("switchable",function(){
    var Switchable = S.Switchable;
    //  Switchable
});
```

New in version 1.2: KISSY 1.2 可直接通过依赖注入，从函数参数中取得

```
KISSY.use("switchable",function(S,Switchable){
    //  Switchable
});
```

构造器接口

class Switchable.**Switchable**(container, config)

Parameters

- container (object) -- 所在容器, 可以是 选择器字符串, HTMLElement
- config (object) -- Switchable 的配置信息, 具体见下详解

例如一个简单的接口示例：

```
var a = new S.Tabs('#J_TSearch', {
    markupType: 0,
    navCls: 'tsearch-tab',
    contentCls: 'tsearch-panel',
    triggerType: 'mouse',
    activeTriggerCls: 'current'
});
```

config 配置项详解

`Switchable.markupType`
(optional): {Number} 默认为0. 指明 DOM 结构标记的类型, 可取 0, 1, 2:
当取 0 时, 表示 DOM 是默认结构: 通过 nav 和 content 来获取 triggers 和 panels, 即通过配置以下两个参数获取.

`Switchable.navCls`
(optional): {String}, triggers 所在容器的 class, 默认为 `ks-switchable-nav',

`Switchable.contentCls`
(optional): {String}, panels 所在容器的 class, 默认为 `ks-switchable-content',

这种方式的 DOM 结构类似于:

```
<div id="J_Slide">  <!--    -->
    <ul class="ks-switchable-nav">  <!--     -->
        <li class="ks-active">  A</li>
        <li>  B</li>
        <li>  C</li>
        <li>  D</li>
    </ul>
    <div class="ks-switchable-content">  <!--     -->
        <div>  A</div>
        <div style="display: none">  B</div>
        <div style="display: none">  C</div>
        <div style="display: none">  D</div>
    </div>
</div>
```

当取 1 时, 表示 DOM 结构 可适度灵活：通过 cls 来获取 triggers 和 panels, 即通过配置以下两个参数获取.

`Switchable.triggerCls`
(optional): {String}, 默认为 `ks-switchable-trigger', 会在 container 下寻找指定 class 的元素作为触发器

`Switchable.panelCls`
(optional): {String}, 默认为 `ks-switchable-panel', 会在 container 下寻找指定 class 的元素作为面板

这种方式的 DOM 结构类似于:

```
<div id="J_Accordion">
    <div class="ks-switchable-trigger ks-active"><i class="ks-icon"></i><h3> A</h3></div>
    <div class="ks-switchable-panel"> A<br/> A<br/> A</div>
    <div class="ks-switchable-trigger"><i class="ks-icon"></i><h3> B</h3></div>
    <div class="ks-switchable-panel" style="display:none;"> B<br/> B<br/> B</div>
    <div class="ks-switchable-trigger"><i class="ks-icon"></i><h3> C</h3></div>
    <div class="ks-switchable-panel" style="display:none;"> C<br/> C<br/> C<br/> C<br/> C</div>
    <div class="ks-switchable-trigger last-trigger"><i class="ks-icon"></i><h3> D</h3></div>
    <div class="ks-switchable-panel last-panel" style="display:none;"> D<br/> D<br/> D</div>
</div>
```

当取 2 时, 表示 DOM 结构 完全自由: 直接传入 triggers 和 panels, 即通过配置以下两个参数获取.

`Switchable.triggers`
(optional): {Array}, 默认为 [], 触发器数组

`Switchable.panels`
(optional): {Array}, 默认为 [], 面板数组

这种方式下, DOM 结构就非常自由了, 传入什么内容有你自己定, 只需要 triggers 和 panels 的数量保持一致就好.

`Switchable.hasTriggers`
(optional): {Boolean}, 默认为 true, 是否有触点

`Switchable.triggerType`
(optional): {String} `mouse' 或 `click' , 默认为 `mouse' 触发类型

`Switchable.delay`
(optional): {Number} 触发延迟时间, 单位为s, 默认为 .1 , 即 100ms

`Switchable.activeIndex`
(optional): {Number} markup 的默认激活项, 应该与此 index 一致, 默认为 0

Note: 使用此项时, 需要让激活项对应的 trigger 和 panel 的 HTMLElement, 在 DOM 结构上设置为 激活状态, 不然无法正确切换

`Switchable.activeTriggerCls`
(optional): {String} 激活某个 trigger 时设置的 class , 默认是 `ks-active'

`Switchable.switchTo`
(optional): : {Number} 初始话时, 自动切换到指定面板, 默认为 0 , 即第一个

Note: switchTo 和 activeIndex 的区别是:
- activeIndex 需要 DOM 上设置激活状态, 初始化后不会去切换状态;
- switchTo 则不需要修改 DOM, 但 switchTo 设置后, 会去切换到指定状态, 这在用了一些动画效果时, 切换动作更为明显;

`Switchable.steps`
(optional): {Number} 步长, 表示每次切换要间隔多少个 panels, 默认为 1

`Switchable.viewSize`
(optional): {Array} 可见视图区域的大小. 一般不需要设定此值, 仅当获取值不正确时, 用于手工指定大小, 默认为 []

`Switchable.autoplay`
(optional): {Boolean} 是否自动切换, 默认为 false, 开启后, 不需要触发触发器, 即可自动播放

`Switchable.interval`
(optional): {Number} 自动播放间隔时间, 以 s 为单位, 默认为 5,

`Switchable.pauseOnHover`
(optional): {Boolean} triggerType 为 mouse 时, 鼠标悬停在 slide 上是否暂停自动播放, 默认为 true

`Switchable.circular`
(optional): {Boolean} 是否循环切换, 默认为 false, 是否循环播放, 当切换到最初/最后一个时, 是否切换到最后/最初一个

`Switchable.effect`
(optional): {String} 动画效果函数, 默认没有特效, 可取 `scrollx`, `scrolly`, `fade` 或者直接传入自定义效果函数

`Switchable.duration`
(optional): {Number} 动画的时长, 默认为 .5

`Switchable.easing`
(optional): {String|Function} 动画效果, 详见 `Anim`, 默认为 `easeNone`

`Switchable.lazyDataType`
(optional): {String} 默认为 `area-data', 设置延迟加载时使用的数据类型, 可取 `area-data`, 即 `textarea` 标签 或 `img-src`, 即 `img` 标签

Note: 支持懒加载, 需要载入 S.Datalazyload, 详见 `Datalazyload`

`Switchable.aria`
(optional): {Boolean} 无障碍访问支持, 默认为 false, 即关闭

实例属性

`Switchable.container`
(只读): {HTMLElement} 容器元素

`Switchable.config`
(只读): {Object} 配置信息

`Switchable.triggers`
(只读): {Array} 触发器集合, 可以为空值 []

`Switchable.panels`
(只读): {Array} 切换面板集合, 可以为空值 []

`Switchable.content`
(只读): {HTMLElement} 存放面板的容器元素

`Switchable.length`
(只读): {Number} 触发器或面板的个数

`Switchable.activeIndex`
(只读): {Number} 当前被激活的触发器序号, 从0 开始

`Switchable.switchTimer`
(只读): {Object} 切换定时器, 一般作为内部使用

实例方法

`Switchable.switchTo(index, direction, ev, callback)`
Parameters

- index ({Number}) -- 要切换的项
- direction ({String}) -- (可选) 方向, 用于 effect, 可取 `forward', `backward', 或者不设置
- ev ({EventObject}) -- (可选) 引起该操作的事件
- callback ({Function}) -- (可选) 运行完回调, 和绑定 switch 事件作用一样

切换到某个视图

`Switchable.prev(ev)`
Parameters ev ({EventObject}) -- (可选) 引起该操作的事件

切换到上一视图

Switchable.**next**(ev)
Parameters ev ({EventObject}) -- (可选) 引起该操作的事件
切换到下一视图

触发事件

Switchable.**beforeSwitch**
切换前事件

Hint: 当该事件的函数处理器返回 false, 则会阻止切换动作.

Switchable.**switch**
切换事件

```
var tabs = new S.Tabs('#demo1');

tabs.on('switch', function(ev) {
    if (ev.toIndex === 0) {
        alert('    ');
    }
});
```

子类 Tabs

S.Tabs 接口及配置项, 与 Switchable 完全相同!

子类 Slide

默认配置

S.Slide 接口及配置项, 与 Switchable 相同, 其中以下配置项的默认值有所区别:

Switchable.autoplay 是否自动切换, 默认为 true

Switchable.circular 是否循环切换, 默认为 true

子类 Carousel

默认配置

S.Carousel 接口及配置项, 与 Switchable 相同, 其中以下配置项的默认值有所区别:

Switchable.circular 默认为 true, 即开启循环切换

新增配置

Carousel.**prevBtnCls**
(optional): {String} 前一个触发器的 cls, 默认为 `ks-switchable-prev-btn'

Carousel.**nextBtnCls**
(optional): {String} 后一个触发器的 cls，默认为 `ks-switchable-next-btn'

Carousel.**disableBtnCls**
(optional): {String} 触发器不可用时的 cls，默认为 `ks-switchable-disable-btn'

子类 Accordion

默认配置

S.Accordion 接口及配置项，与 Switchable 相同，其中以下配置项的默认值有所区别：

Switchable.markupType 默认为1，选择自定义 trigger 和 panel 的 class

Switchable.triggerType 默认为 `click'，点击触发

新增配置

Accordion.**multiple**
(optional): {Booelan} 是否开启同时展开多个面板功能，默认为 false

2.3.4 DD

By 承玉

Draggable

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Draggable = DD.Draggable;
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器

class DD.**Draggable**(config)
Parameters config (object) -- 实例化可拖放对象的配置项，包括

config.**node**
类型选择器字符串或者 HTMLElement ，将要进行拖放的节点。

config.**handlers**
类型数组，数组每个元素是选择器字符串或者 HTMLElementNode ，作为鼠标在其上按下时触发节点拖放的钩子。 如果不设置，则整个 node 作为触发钩子。

> Note: handlers 的每个元素 dom 节点必须位于配置项 node dom 子树中。

config.**cursor**
类型字符串，默认值 ``move''，handlers 元素中的每个元素要设置的鼠标样式。

config.**mode**
类型枚举字符串，默认值 ``point''，和 Droppable 关联，决定何时和可放对象开始交互（触发相应事件）， 可取值 ``point'',''intersect'',''strict''

- 在 ``point'' 模式下，只要鼠标略过可放对象，即开始和可放对象交互。
- 在 ``intersect'' 模式下，只要拖动对象和可放对象有交集，即开始和可放对象交互。
- 在 ``strict'' 模式下，只有拖动对象完全位于可放对象内，才开始和可放对象交互。

类常量

Draggable.**POINT**
等于 ``point''

Draggable.**INTERSECT**
等于 ``intersect''

Draggable.**STRICT**
等于 ``strict''

实例属性

Draggable.**node**
类型 KISSY.Node ，表示当前拖动的节点，在应用 DD.Proxy 时表示代理节点。

Draggable.**dragNode**
类型 KISSY.Node ，表示配置项中 node 的值。

> Note: 实例属性通过 get 方法获取，例如 drag.get("node")

实例方法

Draggable.**destroy**()
销毁当前可拖放对象实例，清除绑定事件。

触发事件

Draggable.**dragstart**
当可拖放对象开始被用户拖放时触发，传给事件处理函数参数为事件对象 event ，包含
Draggable.dragstart.event.**drag**
自身，当前拖放对象

Draggable.**drag**
当可拖放对象拖放过程中触发，传给事件处理函数为事件对象 event ，包含
Draggable.drag.event.**left**
type number，拖放节点应该设置的相对文档根节点的横坐标位置。

Draggable.drag.event.**top**
type number，拖放节点应该设置的相对文档根节点的纵坐标位置。

Draggable.drag.event.**pageX**
type number，当前鼠标的绝对横坐标

Draggable.drag.event.**pageY**
type number，当前鼠标的绝对纵坐标

Draggable.drag.event.**drag**
自身，当前拖放对象

Draggable.**dragend**
当用户鼠标弹起放弃拖放时触发，传给事件处理函数参数为事件对象 event ，包含
Draggable.dragend.event.**drag**
自身，当前拖放对象

Draggable.**dragenter**
同 Droppable.dropenter ，只不过该事件在当前 Draggable 对象上触发。

Draggable.**dragover**
同 Droppable.dropover ，只不过该事件在当前 Draggable 对象上触发。

Draggable.**dragexit**
同 Droppable.dropexit ，只不过该事件在当前 Draggable 对象上触发。

Draggable.**dragdrophit**
同 Droppable.drophit ，只不过该事件在当前 Draggable 对象上触发。

Draggable.**dragdropmiss**
当用户鼠标弹起但是没有放置当前 Draggable 对象到一个 Droppable 对象时触发。 传给事件处理函数参数为一个事件对象 event
Draggable.dragdropmiss.event.**drag**
自身，当前拖放对象

Note: Draggble 实例化后仅表示会根据鼠标拖放触发相应的事件，但具体怎么处理仍需要调用者自己控制， 例如可监听 drag 事件，根据事件对象参数的坐标设置拖放节点的具体位置。

```
new Draggable({node :"#d"}).on("drag",function(ev){
    this.get("node").offset({left:ev.left,top:ev.top});
});
```

Droppable

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Droppable = DD.Droppable;
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器

class DD.**Droppable**(config)
Parameters config (object) -- 可放对象的一些初始化配置，包括

config.**node**
类型选择器字符串或者 HTMLElement。可与拖动对象交互的节点。

实例属性

Droppable.**node**
类型 KISSY.Node ，获取配置项的 node 值。

实例方法

Droppable.**destroy**()
销毁可放对象实例，清除绑定事件

触发事件

Droppable.**dropenter**
当一个 Draggable 对象根据其 mode 配置达到和当前 Droppable 实例交互条件时触发。 一般即鼠标进入当前 Droppable 对象代表节点的区域，可简单理解成 mouseenter。传给事件处理函数一个事件对象 event，包含

Droppable.dropenter.event.**drag**
当前交互的 Draggable 对象

Droppable.dropenter.event.**drop**
自身，当前 Droppable 兑现。

Droppable.**dropover**
当一个 Draggable 在当前 Droppable 实例上移动时触发，可简单理解成 mouseover。 传给事件处理函数一个事件对象 event，包含

Droppable.dropover.event.**drag**
当前交互的 Draggable 对象

Droppable.dropover.event.**drop**
自身，当前 Droppable 兑现。

Droppable.**dropexit**
当一个 Draggable 离开当前 Droppable 实例时触发，可简单理解成 mouseleave。 传给事件处理函数一个事件对象 event，包含

Droppable.dropexit.event.**drag**
当前交互的 Draggable 对象

Droppable.dropexit.event.**drop**
自身，当前 Droppable 兑现。

Droppable.**drophit**
当一个 Draggable 被放置在当前 Droppable 实例时触发，传给事件处理函数一个事件对象 event，包含

Droppable.drophit.event.**drag**
当前交互的 Draggable 对象

Droppable.drophit.event.**drop**
自身，当前 Droppable 对象。

DDM

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var DDM = DD.DDM;
});
```

See Also:

KISSY 1.2 Loader 新增功能

简介

DD.**DDM**
为拖放的中央控制对象，所有的拖放实例的事件都会向其冒泡。 因此可以对整个拖放模块进行一些配置，以及对 DDM 进行全局事件监听和处理。

Note: DDM 上的事件监听处理会影响到所有的拖放实例。

属性

DDM.**bufferTimer**
: 类型 number，默认 200，表示鼠标按下多长时间后触发 dragstart 事件。可通过 DDM.set("bufferTimer",xx) 设置。

事件

DDM.**dragstart**
: 同 Draggable.dragstart ，只不过在 DDM 上触发。

DDM.**drag**
: 同 Draggable.drag ，只不过在 DDM 上触发。

DDM.**dragend**
: 同 Draggable.dragend ，只不过在 DDM 上触发。

DDM.**dragenter**
: 同 Draggable.dragenter ，只不过在 DDM 上触发。

DDM.**dragover**
: 同 Draggable.dragover ，只不过在 DDM 上触发。

DDM.**dragexit**
: 同 Draggable.dragexit ，只不过在 DDM 上触发。

DDM.**dragdrophit**
: 同 Draggable.dragdrophit ，只不过在 DDM 上触发。

DDM.**dragdropmiss**
: 同 Draggable.dragdropmiss ，只不过在 DDM 上触发。

DDM.**dropenter**
: 同 Droppable.dropenter ，只不过在 DDM 上触发。

DDM.**dropover**
: 同 Droppable.dropover ，只不过在 DDM 上触发。

DDM.**dropexit**
: 同 Droppable.dropexit ，只不过在 DDM 上触发。

DDM.**drophit**
: 同 Droppable.drophit ，只不过在 DDM 上触发。

DraggableDelegate

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var DraggableDelegate = DD.DraggableDelegate;
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器

class DD.**DraggableDelegate**(config)
返回拖放实例，委托容器内的所有 Draggable 节点的拖放行为。
Parameters config (object) -- 实例化可拖放对象的配置项，包括

config.**container**
类型选择器字符串或者 HTMLElement ，用于委托的容器节点，所有 Draggable 节点都在其内。

config.**selector**
类型选择器字符串，格式为 tag 或 tag.cls 或 .cls。用来获取容器内的 Draggable 节点。

config.**handlers**
类型数组，数组每个元素是选择器字符串，格式为 tag 或 tag.cls 或 .cls，作为鼠标在其上按下时触发节点拖放的钩子。 如果不设置，则整个可拖放节点都作为触发钩子。 其中可拖放节点通过 selector 从容器 container 中取得。

Note: handlers 的每个元素 dom 节点必须位于可拖放节点中。

Note: 对于委托的可拖节点，当和 Droppable 实例交互时， mode 统一为 POINT

实例属性

DraggableDelegate.**node**
类型 KISSY.Node ，表示当前正在拖动的被委托的容器内子节点，在应用 DD.Proxy 时表示委托节点。

DraggableDelegate.**dragNode**
类型 KISSY.Node ，表示当前正在拖动的被委托的容器内子节点。

Note: 实例属性通过 get 方法获取，例如 dragDelegate.get("node")

实例方法

DraggableDelegate.**destroy**()
销毁当前可拖放对象实例，清除绑定事件。

触发事件

继承自 Draggable .包括 dragstart , drag , dragend , dragenter , dragover , dragexit , dragdrophit , dragdropmiss .

Note: 可以通过 node 来获得正在拖放的委托子节点（或委托节点的代理节点），通过 dragNode 来获得正在拖放的代理真正子节点。

DroppableDelegate

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var DroppableDelegate = DD.DroppableDelegate;
});
```

See Also:

KISSY 1.2 `Loader` 新增功能

构造器

class `DD.DroppableDelegate`(config)

Parameters config (object) -- 可放对象的一些初始化配置，包括

`config.container`
类型选择器字符串或者 HTMLElement ，用于委托的容器节点，所有 Droppable 节点都在其内。

`config.selector`
类型选择器字符串，格式为 tag 或 tag.cls 或 .cls。用来获取容器内的 Droppable 节点。

实例属性

`DroppableDelegate.node`
类型 `KISSY.Node` ，表示当前容器内正在和 Draggble 对象交互的节点， 通过 `selector` 获取。

实例方法

`DroppableDelegate.destroy`()
销毁可放对象实例，清除绑定事件

触发事件

继承自 `Droppable` .包括 `dropenter` , `dropover` , `dropexit` , `drophit` .

Note: 可以通过 `node` 来获得当前正在和 `Draggable` 对象交互的被委托的容器内的子节点。

Proxy

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Proxy = DD.Proxy;
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器

class DD.**Proxy**(config)

Parameters config (object) -- 实例化可拖放对象的配置项，包括

config.node

类型函数，当 Draggable 对象需要代理节点时通过调用该函数产生代理节点，函数的参数为当前 Draggable 对象， 返回值类型为 KISSY.Node 。该属性有默认值

```
function(drag) {
    return new Node(drag.get("node")[0].cloneNode(true));
}
```

即代理节点和当前节点保持一致。

config.destroyOnEnd

类型 boolean，默认 false。指明是否代理节点需要每次拖放前 dragstart 生成， 拖放后 dragend 销毁。用于多 Draggable 对象共享一个 Proxy 对象实例，且要求代理节点和单个 Draggable 对象关联，或者一个 DraggableDelegate 对象共享一个 Proxy 对象实例。

实例方法

Proxy.**attach**(drag)

使当前拖放对象具备代理功能。

Parameters drag (Draggable) -- 需要具备代理功能的 Draggable 对象

Proxy.**unAttach**(drag)

解除当前拖放对象的代理功能。

Parameters drag (Draggable) -- 具备代理功能的 Draggable 对象

Proxy.**destroy**()

解除所有通过当前 Proxy 对象添加的代理功能

Scroll

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Scroll = DD.Scroll;
});
```

See Also:

KISSY 1.2 Loader 新增功能

构造器

class DD.Scroll(config)

监控容器关联的所有可拖放对象，必要时随着可拖放对象进行自动滚动。

Parameters config (object) -- 实例化可拖放对象的配置项，包括

config.node

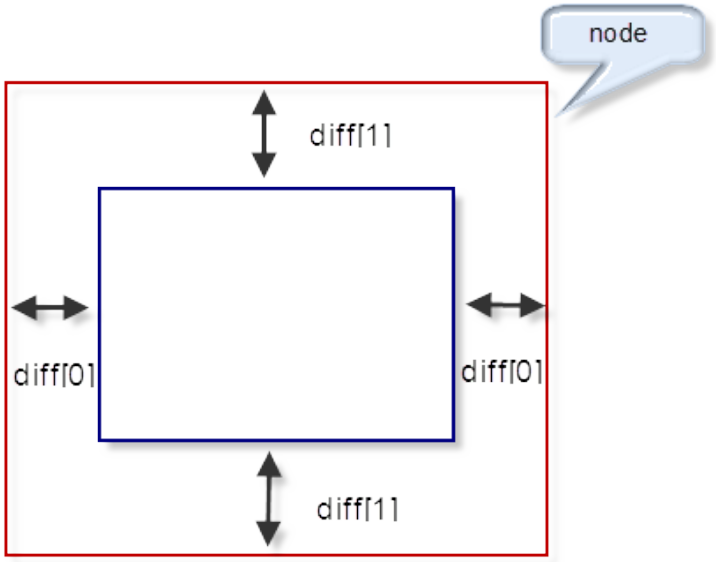
类型选择器字符串或者 HTMLElememt，自动滚动容器，随其内的可拖放节点自动滚动。

config.rate

类型 Array<number>，长度为 2，默认值 [10,10]．表示容器自动滚动的速度，数组元素 1 表示横向滚动的速度，数组元素 2 表示纵向滚动的速度。

config.diff

类型 Array<number>，长度为 2，默认值 [20,20]．当鼠标进入容器内边缘区域时开始自动滚动。 数组元素 1 表示横向容器内边缘宽度，数组元素 2 表示纵向容器内边缘宽度。如图所示



实例方法

`Scroll.attach(drag)`
注册可拖放对象到当前容器。
Parameters drag (Draggable) -- 需要使容器自动滚动的 Draggable 对象

`Scroll.unAttach(drag)`
解除当前容器关联的可拖放对象。
Parameters drag (Draggable) -- 使容器自动滚动的 Draggable 对象

`Scroll.destroy()`
解除当前容器关联的所有可拖放对象。

2.3.5 Suggest

作者： 玉伯

1. 提示补全基本功能
2. 完全跨域
3. 小巧精简，仅依赖 ks-core, 压缩后不超过 9k
4. 支持所有 A 级浏览器
5. cache 功能
6. 支持键盘控制：上下选择及回车后直接提交，ESC 键关闭
7. 支持鼠标控制：鼠标选择和点击提交功能
8. [DELAY] 支持匹配文字加亮
9. [DELAY] 动画效果

10. [DELAY] 在提示层中显示第一个搜索结果
11. [DELAY] 整合本地表单的提示记录
12. [DELAY] 关键词的模糊匹配提示功能

Suggest

构造器

class KISSY.**Suggest**(textInput, dataSource, config)

Parameters

- textInput (String|HTMLElement) -- 输入框
- dataSource (String) -- 数据源
- config (Object) -- 配置项

containerCls

悬浮提示层的class
默认为空

containerWidth

提示层的宽度，示范取值：`200px', `10%' 等，必须带单位
默认为空，和input输入框的宽度保持一致

对象属性

对象方法

触发事件

类属性

类方法

2.3.6 Template

by 文河

特性

- 模板语法,从 {{#tagName}} 开始,由 {{/tagName}} 结束(如果有结束标签的话).
- 模板变量, {{variable}} .
- 原生支持 if/elseif/else/each/! 四个标签.
- 支持嵌套.
- 容错和调试.

- 性能还不赖.
- 容易扩展.

模板语法和范例

变量

变量支持JavaScript语法里的任何有返回值的语句,比如 `name`，`user.name`，`user[0].name`，甚至可以使用方法, `KISSY.one('#template').html()`

语法

```
{{Variable}}
```

范例

```
KISSY.Template('Hello, {{name}}.')
    .render({name: 'Frank'});

Hello, Frank.

KISSY.Template('Hello, {{user.name}}.')
    .render({user: {name: 'Frank'}});

Hello, Frank.
```

if 语句

语法

```
{{#if conditions}}
    BLOCK
{{/if}}
```

范例

```
KISSY.Template('Hello, {{#if show}}{{name}}{{/if}})')
    .render({show: true, name: 'Frank'});

Hello, Frank
```

else和elseif

语法

```
{{#if conditions}}
    BLOCK
{{#elseif conditions}}
    ELSEIF BLOCK
{{#else}}
```

```
    ELSE BLOCK
{{/if}}
```

范例

```
KISSY.Template('Hello, {{#if showName}}{{name}}.{{#else}}{{nick}}{{/if}})')
    .render({showName: false, name: 'Frank', nick: 'yyfrankyy'});

Hello, yyfrankyy.

KISSY.Template('Hello, {{#if name}}{{name}}.{{#elseif nick}}{{nick}}{{/if}})')
    .render({name: 'Frank', nick: 'yyfrankyy'});

Hello, Frank.
```

each

循环读取某个变量,直接调用 KISSY.each 方法进行遍历.

语法

```
{{#each conditions as value index}}
    BLOCK
{{/each}}
```

注意 as value index 可选

范例1(使用默认的循环参数)

```
KISSY.Template('Hello, {{#each users}}<b color="{{_ks_value.color}}">{{_ks_value.user}}</b>{{/each}})')
    .render({users: [{name: 'Frank', color: 'red'}, {name: 'yyfrankyy', color: 'green']});

Hello, <b color="red">Frank</b><b color="green">yyfrankyy</b>
```

范例2(使用自定义参数,可选)

```
KISSY.Template('Hello, {{#each users as user}}<b color="{{user.color}}">{{user.name}}</b>{{/each}})')
    .render({users: [{name: 'Frank', color: 'red'}, {name: 'yyfrankyy', color: 'green']});

Hello, <b color="red">Frank</b><b color="green">yyfrankyy</b>

KISSY.Template('Hello, {{#each users as user index}}<b color="{{user.color}}">{{index}}:{{user.name}}</b>{{/each}})
    .render({users: [{name: 'Frank', color: 'red'}, {name: 'yyfrankyy', color: 'green']});

Hello, <b color="red">0:Frank</b><b color="green">1:yyfrankyy</b>
```

范例3(嵌套使用)

```
KISSY.Template('Hello, {{#each users as user}}<b color="{{user.color}}">{{#each user.names as name}}{{name}}{{/each
    .render({users: [{names: ['Frank', 'Wang'], color: 'red'}, {names: ['Frank', 'Xu'], color: 'green']});

Hello, <b color="red">FrankWang</b><b color="green">FrankXu</b>
```

单行注释

语法

```
{{#! comments}}
```

范例

```
KISSY.Template('Hello, {{#! here you go.}}{{name}}.').render({name: 'Frank'});

Hello, Frank.
```

标签嵌套

理论上支持任意标签嵌套,如果标签有关闭字符,记得关闭=,=,嵌套标签形成多代码块嵌套,作用域与JavaScript的作用域一致.

语法

```
{{#each object}}
    {{#if condition}}
        BLOCK
    {{/if}}
{{/each}}
```

范例

```
KISSY.Template('Hello, {{#each users}}{{#if _ks_value.show}}{{_ks_value.name}}{{/if}}{{/each}}.')
    .render({users: [{show: false, name: 'Frank'}, {show: true, name: 'yyfrankyy'}]});

Hello, yyfrankyy.
```

容错和调试.

容错

目前支持两种错误信息:

1. Syntax Error. 指模板在预编译阶段发生语法错误(模板编译后生成的脚本语法错误).
2. Render Error. 指模板在渲染时发生错误(运行时错误,数据错误,或者模板变量错误等).

调试

默认情况下,模板将编译时和运行时的错误,直接返回到结果里.

调试过程可调用 `KISSY.Template.log()` 方法输出渲染方法,定位脚本模板错误,并可通过引用 `jsbeauty` 来格式化生成的模板方法.

模板性能对比

https://spreadsheets.google.com/ccc?key=0ApZFGfLktT7FdDgtcGdzWV9wSzRpX2FRTElzZmVoV2c&hl=en#gid=3

扩展

模板方法仅依赖于KISSY的 `core` 部分,默认调用为:

```
KISSY.Template('template here.').render(data);
```

而:

```
var templ = KISSY.Template();
```

可直接预编译模板方法.

API

KISSY.tmpl

```
KISSY.tmpl('#template', {name: 'Frank'}).appendTo('#container');
```

语法扩展

> `KISSY.Template.addStatement()` 方法,提供扩展语法的接口,目前支持标签语法开始,关闭及一个参数传递.

比如:

```
KISSY.Template.addStatement({'while': {
    start: 'while(KS_TEMPL_STAT_PARAM){',
    end: '}'
}});
```

即可支持 `while` 语句

```
{{#while true}}
    BLOCK
{{/while}}
```

CHAPTER

THREE

范例

3.1 Seed

3.1.1 Loader

by 承玉

< 1.2

注册模块

```
KISSY.add({
   "1.1x-dep":{
        fullpath:"http://lite-ext.googlecode.com/svn/trunk/lite-ext/playground/module_package/1.1x/dep.js"
   },
   "1.1x-mod":{
        fullpath:"http://lite-ext.googlecode.com/svn/trunk/lite-ext/playground/module_package/1.1x/mod.js",
        cssfullpath:"http://lite-ext.googlecode.com/svn/trunk/lite-ext/playground/module_package/1.1x/mod.css",
        requires:["1.1x-dep"]
   }
});
```

定义模块

被依赖模块 1.1x dep

```
KISSY.add("1.1x-dep",function(){
    alert("1.1x-dep loaded");
});
```

主模块 1.1x mod

```
KISSY.add("1.1x-mod",function(){
    alert("1.1x-mod loaded");
});
```

使用模块

```
KISSY.use("1.1x-mod");
```

demo

1.2

包配置

```
KISSY.config({
    packages:[
        {
            name:"1.2", //
            tag:"20110323",//
            path:"http://lite-ext.googlecode.com/svn/trunk/lite-ext/playground/module_package/", //
            charset:"gbk" //
        }
    ]
});
```

定义模块

被依赖模块 1.2 dep

```
KISSY.add(function(){
    alert("1.2/dep loaded");
});
```

主模块 1.2 mod

```
KISSY.add(function(){
    alert("1.2/mod loaded");
},{
    requires:["./dep","./mod.css"] //    js
});
```

使用模块

```
KISSY.use("1.2/mod");
```

demo

3.2 Core

3.2.1 Anim

by 承玉

使用构造器

源码：

```
KISSY.use("anim,node",function(S,Anim,Node){
    //KISSY 1.2     var Node=S.Node ; var Anim=S.Anim
     var anim = Anim(
                '#test1',
                {
                    'background-color':'#fcc',
                    //'border': '5px dashed #999',
                    'border-wdith':'5px',
                    'border-color':"#999999",
                    'border-style':"dashed",
                    'width': '100px',
                    'height': '50px',
                    'left': '900px',
                    'top': '285px',
                    'opacity': '.5',
                    'font-size': '48px',
                    'padding': '30px 0',
                    'color': '#FF3333'
                },5,
                'bounceOut',function(){
                    alert('demo1  ');
                });
     S.one("#test1-btn").on("click",function(){
        anim.run();
     });
});
```

滚动属性动画实例

源码：

```
KISSY.use("anim",function(S,Anim){
    S.one("#test-scroll").on("click", function() {
        S.one("#test-scroll")[0].disabled = true;
        Anim(S.get("#test8"),{
            //  scrollLeft   scrollTop
            scrollLeft:500
        }, 5, undefined, function() {
            Anim(S.get("#test8"),{
                scrollLeft:0
            }, 5, undefined, function() {
```

```
                S.one("#test-scroll")[0].disabled = false;
            }).run();
        }).run();
    });
});
```

节点实例动画操作

源码：

```
KISSY.use("anim,node",function(S,Anim,Node){
    //KISSY 1.2     var Node=S.Node ; var Anim=S.Anim
    var demo_show=S.one("#demo_show"),
    demo_slide=S.one("#demo_slide"),
    demo_fade=S.one("#demo_fade");

    var anim_show=S.one("#anim_show"),
    anim_slide=S.one("#anim_slide"),
    anim_fade=S.one("#anim_fade");

    demo_show.on("click",function(){
        if(anim_show.css("display")==="none")
        anim_show.show(1);
        else
        anim_show.hide(1);
    });

    demo_slide.on("click",function(){
        if(anim_slide.css("display")==="none")
        anim_slide.slideDown();
        else
        anim_slide.slideUp();
    });

    demo_fade.on("click",function(){
        if(anim_fade.css("display")==="none")
        anim_fade.fadeIn();
        else
        anim_fade.fadeOut();
    });
});
```

3.3 Component

3.3.1 DataLazyload

基本使用

最简单的调用方式

```
KISSY.use('datalazyload', function(S, DataLazyload) {
    S.ready(function(S) {
        S.DataLazyload( { mod: 'auto' } );
```

```
    });
});
```

这样, 页面加载时就会自动延迟所有图片的下载, 以及延迟特定 textarea 里的 html 渲染.

自动模式

使用自动模式时, 设置配置项中的 `mod` 为 `auto' , 如下:

```
var dataLazyload =DataLazyload({
    placeholder : " .png"
    mod:'auto'
});
```

页面上出现大量图片元素时,

```
<img src="xx.png" />
```

会把当前视窗外的 img 的 src 保存在自定义属性中, 并将 src 替换为 placeholder (不指定为空), 当滚动导致该图片出现在当前视窗时将 src 设置已经保存的真实值.

手动模式

采用手动模式时, 需要自行在输出页面时, 可以不设置 img 的 src 属性, 但是必须设置 img 的 `data-ks-lazyload` 自定义属性为真实图片地址, 如:

```
<img data-ks-lazyload="xx.jpg" />
```

当滚动导致该图片出现在当前视窗时会将 src 设置为真实地址.

textarea 延迟加载

Note: 这种情况下和模式的手动自动没关系!

将页面中需要延迟的 DOM 节点, 放在

```
<textarea class='ks-datalazyload invisible'/>dom code</textarea/>
```

里. 可以添加 hidden 等 class, 但建议用 invisible (visiblity:hidden), 并设定 height = `实际高度', 这样可以保证滚动时无缝连接. 当滚动导致该 textarea 出现在当前视窗时会将该 textarea 内的 html 添加到新生成的 div 中, 并用新生成的 div 替换该 textarea .

另外注意:

1. 延迟 callback 约定：dataLazyload.addCallback(el, fn) 表示当 el 即将出现时, 触发 fn.
2. 所有操作都是最多触发一次, 比如来回拖动滚动条时, 只有 el 第一次出现时会触发 fn 回调.

全部示例

- manual 模式
- auto 模式

3.3.2 Overlay

by 承玉

从 markup 中构建 overlay

最常见的场景，弹出层 html 已经在页面 html 中，前端要做的是适时将它显示出来，例如页面中存在以下 html ：

```
<style>
    #popup1 {
        position:absolute;
        left:-9999px;
        top:-9999px;
    }
</style>

<div id='popup1'>

</div>

<button id="btn1">Show</button>
<button id="btn2">Hide</button>
```

代码分解

```
<style>
    #popup1 {
        position:absolute;
        left:-9999px;
        top:-9999px;
    }
</style>
```

初始载入时，弹出层所在 div 是浮出在屏幕之外而隐藏的，当点击 Show 按钮时，该弹层对齐在 Hide 按钮旁边，当点击 Hide 按钮时，已经显示的弹层就隐藏了。

获得 overlay 对象　通过 srcNode 配置项配置从已存的 dom 节点来生成 `Overlay` 对象：

```
KISSY.use("overlay",function(S,Overlay){
    //   kissy < 1.2 ,   Overlay = S.Overlay;

    var popup = new Overlay({
        srcNode:S.one("#popup1"), //     dom
        width: 300, //
        height: 200,    //
```

```
        align: {    //
            node: '#btn2',
            points: ['tr', 'tl'],
            offset: [50, 0]
        }
    });

});
```

当点击 Show 按钮时会触发弹出层的显示，以及点击 Hide 按钮时会触发弹出层的隐藏。

```
S.one('#btn1').on('click', function() {
    //       Hide
    popup.show();
});

S.one('#btn2').on('click', function() {
    //
    popup.hide();
});
```

最终 demo

全新创建一个 Dialog

有时可能弹窗本身并没有在 html 中存在，而是由脚本完全生成的，这时就不需要 `srcNode` 配置了，直接配置相关属性后即可完全由脚本生成所需的 dom 节点。

生成 dialog 对象

注意：要使得弹出对话框头部可拖动，需要 `use("dd")` 使用拖放模块:

```
KISSY.use("overlay,dd",function(S,Overlay){

    // if kissy >= 1.2
    var Dialog=Overlay.Dialog;

    //   kissy
    var Dialog=S.Dialog;

    //     dialog
    var dialog = new Dialog({
        width: 400,  //
        bodyStyle:{
            height: 300 //
        },
        headerContent: 'this is title', //     html
        footerContent: 'footer',    //     html
        bodyContent: 'content', //     html
        mask: true,  //
        draggable: true //
    });

});
```

触发 dialog 对象显示

当点击按钮时，首先调用 `render()` 渲染 `Dialog` 对象，使得 dialog 生成的 dom 节点加入到文档树中，再调用 `center()` 使得对话框位置在当前视窗中央，最后调用 `show()` 显示 dialog：

```
S.one("#btn4").on("click", function() {
    dialog.render();
    dialog.center();
    dialog.show();
});
```

最终 demo

全部 demo

- KISSY.Overlay 1.1.6
- KISSY.Overlay 1.2.0

3.3.3 Switchable

by 乔花

常见使用方式

组织 HTML 结构

通常情况下, Switchable 组件的 HTML 结构为外层一个容器元素,内部又分成 trigger 列表 和 panel 列表 两部分,trigger 列表可无, 如下的 HTML 结构最为常见:

```
<div id="J_Slide">  <!--    -->
    <ul class="ks-switchable-nav">  <!--     -->
        <li class="ks-active">  A</li>
        <li>  B</li>
        <li>  C</li>
        <li>  D</li>
    </ul>
    <div class="ks-switchable-content">  <!--     -->
        <div>  A</div>
        <div style="display: none">  B</div>
        <div style="display: none">  C</div>
        <div style="display: none">  D</div>
    </div>
</div>
```

但注意: 这种结构并不固定, 且有时需要根据不同组件, 不同需求修改结构并定义它们的样式;

JS 初始化

通过容器元素的 id 和相关配置, 构建 `Switchable` 对象：

```
KISSY.use("switchable",function(S,Switchable){
    //   kissy < 1.2 ,   Switchable = S.Switchable;

    var s = new Switchable.Slide('#J_Slide', {
        effect: 'scrolly',
        easing: 'easeOutStrong',
        countdown: true,
        countdownFromStyle: 'width:18px'
    });
});
```

最终 demo

全部示例

Tabs

- 普通标签页

Slide

- 淘宝首页卡盘
- 有啊首页开盘
- 有啊滚动新闻条
- 土豆今日焦点

Carousel

- 普通旋转木马
- 类似于首页上的旋转木马

Accordion

- 普通手风琴

3.3.4 DD

by 承玉

Draggable & Proxy

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Draggable = DD.Draggable;
});
```

See Also:

KISSY 1.2 Loader 新增功能

demo

分解

准备节点

```
<div id='test-drag' style='border:1px solid red;
                    background:blue;width:100px;
                    height:100px;color:white;'>
  drag me
</div>
```

设置代理节点样式

```
.ks-dd-proxy {
    opacity:0.2;
    *filter:alpha(opacity=20);
}
```

载入 dd 模块

```
KISSY.use("dd",function(S,DD){

});
```

初始化 draggable 对象

```
var drag=new DD.Draggable({
    node:'#test-drag',
    cursor:'move'
});
```

初始化 proxy 对象　然后 proxy 对象和 draggable 绑定

```
new Proxy().attach(drag);
```

监控事件，处理移动

```
drag.on("drag",function(ev){
    drag.get("node").offset({
        left:ev.left,
        top:ev.top
    });
});
```

Droppable

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var Droppable = DD.Droppable;
});
```

See Also:

KISSY 1.2 Loader 新增功能

demo

分解

准备节点

```
<div id="container" class="container">
    <div id="c1" class="component">
        intersect drag
    </div>

    <div id="c2" class="component">
        point drag
    </div>

    <div id="c3" class="component">
        strict drag
    </div>

    <div id="drop">
        drop zone
    </div>
</div>
```

获取模块定义

```
KISSY.use("node,dd", function (S, Node, DD) {
    var DDM = DD.DDM,
        Draggable = DD.Draggable,
        Droppable = DD.Droppable;
});
```

全局监控　开始拖放前保存节点的定位信息：

```
DDM.on("dragstart", function(ev) {
    var c = ev.drag;
    p = c.get("dragNode").css("position");
});
```

拖放中，根据位置信息设置节点坐标

```
DDM.on("drag", function(ev) {
    var c = ev.drag;
    /**
     * node   dragNode
     * node :    proxy node
     */
    c.get("node").offset(ev);
});
```

拖放结束后，恢复节点的定位信息

```
DDM.on("dragend", function(ev) {
    var c = ev.drag;
    c.get("dragNode").css("position", p);
});
```

初始拖放对象　实例化 3 个普通的拖实例以及一个放实例

```
var c1 = new Draggable({
    node:"#c1",
    //
    // intersect :   enter
    // strict : drag     drop
    // point :    drop
    //  point
    mode:Draggable.INTERSECT
});


var c3 = new Draggable({
    node:"#c3",
    mode:Draggable.STRICT
});


var c2 = new Draggable({
    node:"#c2"
});
```

```
var drop = new Droppable({
    node:"#drop"
});
```

监听放实例的 drophit 事件　当在 drop 区域放入 draggable 对象时，该 draggable 代表的节点被放入 drop 区域中

```
function onhit(ev) {
    ev.drag.get("node").css("margin", "5px 10px");
    ev.drag.get("node").appendTo(ev.drop.get("node"));
    ev.drag.destroy();
}

drop.on("drophit",onhit);
```

DraggableDelegate

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var DraggableDelegate = DD.DraggableDelegate;
});
```

得到 DraggableDelegate 构造器。

See Also:

KISSY 1.2 Loader 新增功能

demo

参考使用 Draggable 实现的 同样例子 .

分解

准备节点

```
<div id="container3" class="container">

    <div class="component">
        <div class="cheader">

        </div>
        delegate drag
    </div>
```

```
    <button id="add_delegate">add delegate drag</button>


    <div id="drop3">
        drop zone
    </div>
</div>
```

载入 dd 模块

```
KISSY.use("node,dd", function (S, Node, DD) {
    var DDM = DD.DDM,
        DraggableDelegate = DD.DraggableDelegate,
        Droppable = DD.Droppable;
});
```

初始化拖放委托对象

- 指明容器以及容器内需要委托的可拖放节点

```
var delegate = new DraggableDelegate({
        container:"#container3",
        handlers:['.cheader'],
        selector:'.component'
    });
```

- 生成 Droppable 对象

```
var drop = new Droppable({
        node:"#drop3"
    });
```

监控 Draggable 集中在 DDM 上处理移动

```
var p;
/**
 *
 */
DDM.on("dragstart", function(ev) {

    var c = ev.drag;
    p = c.get("dragNode").css("position");
});

DDM.on("drag", function(ev) {

    var c = ev.drag;
    /**
     * node   dragNode
     * node :    proxy node
     */
    c.get("node").offset(ev);
});
```

```
DDM.on("dragend", function(ev) {
    var c = ev.drag;
    c.get("dragNode").css("position", p);
});
```

监控 drophit 事件　将被委托的节点放入 Droppable 区域

```
function onhit(ev) {
    ev.drag.get("dragNode").css("margin", "5px 10px");
    ev.drag.get("dragNode").appendTo(ev.drop.get("node"));
    ev.drag.get("dragNode").one(".cheader")[0].className="cheader2";
}

drop.on("drophit",onhit);
```

DroppableDelegate & Scroll

引入

页面引入 kissy.js：

```
<script src='kissy.js'></script>
```

New in version 1.2: 通过 use 加载 dd 模块：

```
KISSY.use("dd",function(S,DD){
    var DroppableDelegate = DD.DroppableDelegate,
        Scroll = DD.Scroll;
});
```

See Also:

KISSY 1.2 Loader 新增功能

demo

基于拖放委托 + 容器自动滚动的拖放排序

代码分解

准备节点

```
<div id="container2" class="container">
    <div class="component">
        <div class="cheader">

        </div>
        <span>
        drag proxy1
            </span>
    </div>
```

```
    <div class="component">
        <div class="cheader">

        </div>
         <span>
        drag proxy2
        </span>
    </div>

    <div class="component">
        <div class="cheader">

        </div>
         <span>
        drag proxy3
        </span>
    </div>
</div>
```

获取 dd 模块

```
KISSY.use("node,dd", function(S, Node, DD) {

    var DDM = DD.DDM,
        DraggableDelegate = DD.DraggableDelegate,
        DroppableDelegate = DD.DroppableDelegate,
        Draggable = DD.Draggable,
        Droppable = DD.Droppable,
        Scroll = DD.Scroll,
        Proxy = DD.Proxy;

});
```

初始化模块类实例　生成 DraggableDelegate 对象

```
var dragDelegate = new DraggableDelegate({
    container:"#container2",
    handlers:['.cheader'],
    selector:'.component'
});
```

生成 DroppableDelegate 对象

```
var dropDelegate = new DroppableDelegate({
    container:"#container2",
    selector:'.component'
});
```

生成 Proxy 对象，并关联到 DraggableDelegate 对象

```
var proxy = new Proxy({
    /**
     *
     * @param drag
     */
```

```
    node:function(drag) {
        var n = S.one(drag.get("dragNode")[0].cloneNode(true));
        n.attr("id", S.guid("ks-dd-proxy"));
        n.css("opacity", 0.2);
        return n;
    },
    destroyOnEnd:true
});

proxy.attach(dragDelegate);
```

生成指定容器的 Scroll 对象，并关联到 DraggableDelegate 对象

```
var s=new Scroll({
    node:"#container2"
});

s.attach(dragDelegate);
```

监控移动　在 DraggableDelegate 上监听移动事件，并移动相应的被委托节点

```
dragDelegate.on("drag", function(ev) {

    var c = this;
    /**
     * node   dragNode
     * node :    proxy node
     */
    c.get("node").offset(ev);
});
```

交换节点位置　当触发 dragover 事件时，交换当前 DraggableDelegate 的被委托节点与对应 DroppableDelegate 的被委托节点

```
dragDelegate.on("dragover", function(ev) {
    var drag = ev.drag;
    var drop = ev.drop;
    var dragNode = drag.get("dragNode"),
            dropNode = drop.get("node");
    var middleDropX = (dropNode.offset().left * 2 + dropNode.width()) / 2;
    if (ev.pageX > middleDropX) {
        var next = dropNode.next();
        if (next && next[0] == dragNode) {

        } else {
            dragNode.insertAfter(dropNode);
        }
    } else {
        var prev = dropNode.prev();
        if (prev && prev[0] == dragNode) {

        } else {
            dragNode.insertBefore(dropNode);
        }
```

```
    }
});
```

更多

more on github

3.3.5 Suggest

作者： 玉伯

CHAPTER

FOUR

子项目集

4.1 Editor

by 承玉 , 玉伯

4.1.1 doc

引入

引入 css

1. 在页头加入 reset css (淘宝页面一般都已引入)

```
<link href="http://a.tbcdn.cn/s/kissy/1.1.x/cssreset/reset-min.css" rel="stylesheet"/>
```

2. 在页头加入编辑器淘宝主题 css

```
<!--[if lt IE 8]>
<link href="http://a.tbcdn.cn/s/kissy/1.1.x/editor/theme/cool/editor-pkg-sprite-min.css" rel="stylesheet"/>
<![endif]-->
<!--[if gte IE 8]><!-->
<link href="http://a.tbcdn.cn/s/kissy/1.1.x/editor/theme/cool/editor-pkg-min-datauri.css" rel="stylesheet"/>
<!--<![endif]-->
```

Note: 1.1.x 表示 1.1.6 或者 1.1.7.

引入 javascript

只要引入外部脚本

```
<script src="http://a.tbcdn.cn/s/kissy/1.1.7/??kissy-min.js,uibase/uibase-pkg-min.js,dd/dd-pkg-min.js,overlay/overl
```

Note: 如果页面已经引入了 `kissy 1.1.6` ，则上述脚本不引入，转而引入

```
<script src='http://a.tbcdn.cn/s/kissy/1.1.6/editor/editor-pkg-min.js'></script>
```

加入 textarea

```
<textarea id="textareaId" style="width:90%;height:200px"></textarea>
```

该 textarea 将被编辑器组件替换。

Note: 宽高一定要用行内样式设定，否则各个浏览器编辑器大小会有差别！

初始化编辑器

构造器:

class KISSY.**Editor**(textareaId, config)

获得编辑器实例

Parameters

- textareaId (string) -- 编辑器装饰的文本输入框 id，注意 id前加 #
- config (Object) -- 编辑器以及其插件的配置项。包括：

config.attachForm
类型 boolean， 是否监控编辑器所属的表单提交

config.baseZIndex
类型 number，编辑器内弹窗z-index下限。

Note: 为防止和其他页面元素互相覆盖，请保证这个数字整个页面最高

---

config.customStyle
类型 string，编辑区域的自定义样式

config.focus
类型 boolean，是否编辑器初始自动聚焦（光标位于编辑器内）

config.pluginConfig
类型 object，编辑器插件的相应配置

例如：

```
KISSY.ready(function(S){
    S.Editor("#textareaId",{
        //
        "attachForm":true,

        //    z-index ,    ,
        "baseZIndex":10000,

        //
        customStyle:"p{color:red;}",

        //
        focus:true
    }).use(plugins);
});
```

使用插件

api

KISSY.Editor.use(plugins)

Parameters plugins (string) -- 需要加载的插件名称, 多个以，号分隔

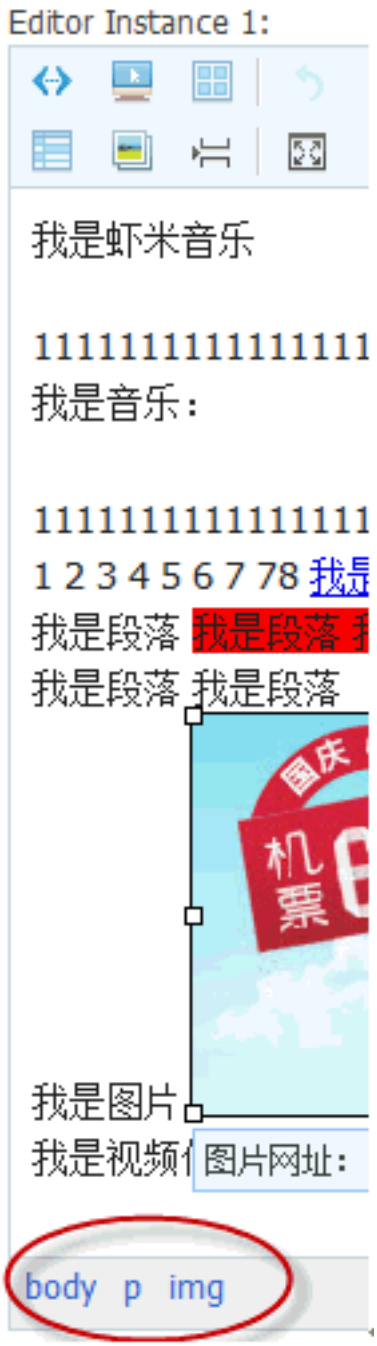
例如：

```
S.Editor("#textareaId").use("font,image,separator")
```

插件名称列表

elementpaths

主要用户开发, 显示光标处所在dom路径.



sourcearea

点击后可以查看编辑器产生的html代码.

preview

点击后可以弹出新窗口，查看编辑器的内容预览.

templates

点击后可以弹出模板选择窗口，选定后可以将模板代码插入到编辑器光标所在处.

separator

间隔线.

undo

可以撤销重做.

removeformat

可以清除选择区域的编辑格式(字体，大小，加粗).

font

大小：可以改变选择区域字体的大小.

字体：可以改变选择区域的字体种类.

粗体：可以将选择区域文字加粗.

斜体：可以将选择区域文字倾斜.

下划线：可以给选择区域文字加下划线.

删除线: 可以给选择区域文字加删除线.

format

可以将光标所在处块加入标题格式.

color

设置选择区域的文本颜色.

设置选择区域的背景颜色.

list

为选择区域或光标所在处加上项目编号.

为选择区域或光标所在处加上列表编号.

indent

减少光标处的缩进量.

增加光标处的缩进量.

justify

左对齐: 光标所在块左对齐.

居中对齐: 光标所在块居中对齐.

右对齐: 光标所在块右对齐.

link

编辑选择区域的超链接.

image

输入图像地址将图像插入到光标所在处.

flash

输入flash地址将flash插入到光标所在处.

music

输入音乐地址将音乐插入到光标所在处.

smiley

选择表情并将对应表情插入到光标所在处.

table

输入表格相关参数并将对应表格插入到光标所在处.

resize

可拖动调整编辑区域大小.

pagebreak

插入分页标记.

maximize

将编辑器充满整个屏幕.

扩展插件　如果要使用以下插件，需要引入另外的 javascript 文件

```
<script src="http://a.tbcdn.cn/s/kissy/1.1.7/editor/biz/ext/editor-plugin-pkg-min.js"></script>
```

multi-upload

多图同时上传功能

具体弹窗交互：

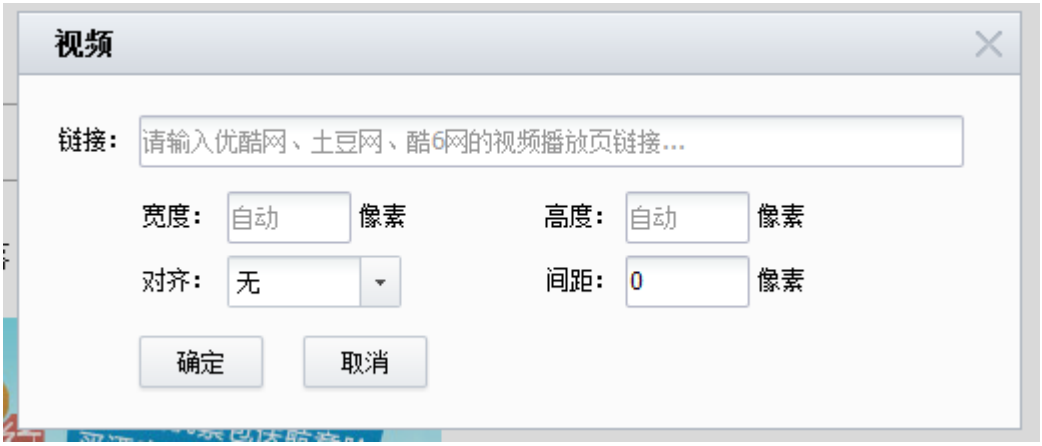
checkbox-sourcearea

勾选交互的源码切换功能

video

国内视频插入,方便直接输入url插入国内视频 flash

具体弹窗交互：



xiami-music

虾米音乐插入,可通过搜索插入虾米音乐

具体弹窗交互：

插件配置

config.pluginConfig:

类型为各个插件的具体配置，配置形式为:

```
{
        " ":
}
```

包括以下插件配置：

pluginConfig.link

超链接配置，目前包括

link.target

默认为 ``''，表示在当前窗口打开新链接，也可以指定 ``_blank'' ，则可以在新窗口打开链接。例如

```
pluginConfig : {
    link:{
        target:"_blank"
    }
}
```

pluginConfig.image.upload

上传图片配置，不需要上传功能可不配置，包括以下子配置

upload.serverUrl
接受文件数据的服务器端程序地址，格式为 multipart/form-data ，返回格式为：

正确：{``imgUrl'':''http://xx.com/yy.jpg``}

错误：{``error'':''i am error!''}

upload.serverParams
类型object，键值对。传给服务器的格外参数，如果 value 是函数则传递函数执行结果

upload.suffix
类型字符串，允许图片的后缀名

upload.fileInput
类型字符串，传给服务器的文件域名

upload.sizeLimit
类型整数，限制上传的文件大小，单位 KB，ie 下只能作为提示。

upload.extraHtml
放入图片上传区域的其他 html

例如：

```
pluginConfig: {
    //
    "image": {
        //
        upload: {
            //
            // {"imgUrl":"http://xx.com/yy.jpg"}
            // {"error":"i am error!"}
            //
            //       multipart/form-data
            serverUrl: "/code/upload/upload.jsp",
            //
            serverParams: {
                yy: function () {
                    return "xx";
                }
            },
            //
            suffix: "png,jpg,jpeg,gif",
            //  server
            fileInput: "Filedata",
            //      KB
            //
            sizeLimit: 1000,
            //k
            //
            extraHtml: "<p style='margin-top:5px;'>" + "<input type='checkbox' name='ke_img_up_watermark
        }
    }
}
```

Note: 如果页面中设置了 document.domain='xx.com'，那么这时上传图片服务器要返回一段设置 domain 的 html，例如

```
<html>
    <head>
        <script>
            document.domain="xx.com";
        </script>
    </head>
    <body>
        {"imgUrl":"http://img03.taobaocdn.com/sns_album/i3/T13fhRXftcXXb1upjX.jpg"}
    </body>
</html>
```

---

pluginConfig.**templates**
: 模板 (templates) 配置，类型数组，数组每个元素为单个模板的配置，单个配置包括

  templates.**demo**
  : 类型字符串，模板的简单介绍

  templates.**html**
  : 类型字符串，将要插入到编辑器区域的具体内容

  例如：

```
{
    "templates": [{
        //
        demo: " 1  html",
        //
        html: "<div style='border:1px solid red'> 1  html</div><p></p>"
    },
    {
        demo: " 2  html",
        html: "<div style='border:1px solid red'> 2  html</div>"
    }],
}
```

pluginConfig.**font-size**
: 字体大小配置，类型 object 或 boolean。当为 false 时，则该按钮不出现在编辑器中。当为 object 时，包含子配置

  font-size.items
  : 类型对象数组，数组每一项的配置为：

    items.**value**
    : 类型 string，真实的字体大小

    items.**attrs**
    : 类型 object，键值对添加到每个字体的对应节点上

    items.**name**
    : 类型 string，字体大小的显示值

  例如：

```
{
    "font-size": {
        //
        items: [{
            //
            value: "14px",
            //
```

```
                attrs: {
                    style: 'position: relative; border: 1px solid #DDDDDD; margin: 2px; padding: 2px;'
                },
                //
                name: " <span style='font-size:14px'> </span>" + "<span style='position:absolute;top:1px;rig
            }]
        }

    }
```

pluginConfig.font-family
字体种类配置，类型 object，包含子配置项

font-family.items
类型对象数组，字体种类配置，单个配置包含

items.name
字体显示值

items.value
设置到编辑元素的真实值

例如：

```
{
    "font-family": {
        items: [{
            //
            name: " ",
            //
            value: "SimSun"
        },
        {
            name: " ",
            value: "SimHei"
        }]
    }
}
```

pluginConfig.font-bold
是否显示粗体按钮，默认 true

pluginConfig.font-italic
是否显示斜体按钮，默认 true

pluginConfig.font-underline
是否显示下划线按钮，默认 true

pluginConfig.font-strikeThrough
是否显示删除线按钮，默认 true

pluginConfig.draft
草稿箱

例如

```
{
    "draft": {
        //
        interval: 5,
        //     ?
```

```
            limit: 10,
            //
            helpHtml: "<div " + "style='width:200px;'>" + "<div style='padding:5px;'>          " + "     " +
        }
    }
```

pluginConfig.resize
: 右下角调整大小的配置

  例如

```
{
    "resize": {
        //  y   ["x","y"]
        direction: ["y"]
    }
}
```

pluginConfig.music
: 音乐插件配置，支持音乐文件地址输入，直接播放mp3

  例如：

```
music: {
    //   url
    musicPlayer: "/music/niftyplayer.swf"
}
```

pluginConfig.multi-upload
: 图片批量上传

  例如

```
{
    "multi-upload": {
        //
        //
        //  {"imgUrl":""}
        //  {"error":""}
        //    \uxxxx
        //        multipart/form-data
        //
        serverUrl: "http://xx.com/code/upload/upload.jsp",
        //
        serverParams: {
            waterMark: function () {
                return S.one("#ke_img_up_watermark_2")[0].checked;
            }
        },
        //
        extraHtml: "<p style='margin-top:10px;'>" + "<input type='checkbox' " + "style='vertical-align:m
        //
        // :http://xx.com/yy.jpg
        //
        //http://xx.com/yy_80x80.jpg
        previewSuffix: "_80x80",

        //
        //
        previewWidth: "80px",
```

```
            //   flash
            sizeLimit: 1000 //k,
            //
            numberLimit: 15
        }
    }
```

Note: #.该插件使用 flash 技术，必须在根域名下提供 crossdomain.xml ，例如 http://www.taobao.com/crossdomain.xml ，内容如下

```
<cross-domain-policy>
    <allow-access-from domain="*.taobao.com"/>
    <allow-access-from domain="*.taobao.net"/>
    <allow-access-from domain="*.taobaocdn.com"/>
    <allow-access-from domain="*.tbcdn.cn"/>
    <allow-access-from domain="*.allyes.com"/>
</cross-domain-policy>
```

#.serverUrl 必须使用绝对地址，否则如果编辑器组件和应用页面不在同一个hostname时，firefox下请求会发到编辑器组件所在的hostname。

---

pluginConfig.video

国内视频插入，可直接输入tudou,youku,ku6的url进行视频粘贴.

例如：

```
{
    "video": {
        urlCfg: [{
            reg: /tudou\.com/i,
            //
            url: "http://bangpai.daily.taobao.net/json/getTudouVideo.htm?" + "url=@url@&callback=@callba
        }],
        //
        urlTip: "       7     ...",
        //        flash
        providers: [{
            reg: /youku\.com/i,
            width: 480,
            height: 400,
            detect: function (url) {
                var m = url.match(/id_([^.]+)\.html$/);
                if (m) {
                    return "http://player.youku.com/player.php/sid/" + m[1] + "/v.swf";
                }
                m = url.match(/v_playlist\/([^.]+)\.html$/);
                if (m) {
                    return;
                    //return "http://player.youku.com/player.php/sid/" + m[1] + "/v.swf";
                }
                return url;
            }
        }]
    }
}
```

pluginConfig.xiami-music

虾米音乐插入，无需配置, 只要 use 即可.

Note: xiami-music，video，multi-upload 为扩展插件，若需要使用请引入外部js

```
<script src="http://a.tbcdn.cn/s/kissy/1.1.x/editor/biz/ext/editor-plugin-pkg-min.js"></script>
```

注意

应用样式

展示页面同样需要引入编辑器外部样式：

```
<!--[if lt IE 8]>
<link href="http://a.tbcdn.cn/s/kissy/1.1.7/editor/theme/cool/editor-pkg-min-mhtml.css" rel="stylesheet"/>
<![endif]-->
<!--[if gte IE 8]><!-->
<link href="http://a.tbcdn.cn/s/kissy/1.1.7/editor/theme/cool/editor-pkg-min-datauri.css" rel="stylesheet"/>
<!--<![endif]-->
```

并且将从数据库中得到的编辑器生成内容用

```
<div class="ke-post"></div>
```

包裹起来.

如果使用了 一般配置中的自定义样式 ， 则需要在展示页面编辑器外部样式后重新定义你的自定义样式：

```
<style type="text/css">
    .ke-post p {
        color:red;
    }
</style>
```

后端开发人员必看

http://dev.taobao.net/?p=4033

FAQ

数据同步

form 同步

如果后端通过 form 提交（submit）来获得用户输入数据，则只需配置参数 `attachForm` .

```
//
"attachForm":true,
```

编辑器自动会监控 form 的 sbumit 事件进行同步

手动同步

涉及三个 api

editor 为调用 KISSY.Editor 返回的编辑器实例

`editor.sync()`：手动同步编辑器内容到对应 textarea

`editor.getData()`：获得当前编辑器的内容

`editor.setData(html:string)`:设置当前编辑器的内容，参数为html，类型为 string 字符串

自动保存

自动保存为 localStorage/flash 机制，保存在本地，可 .

字数统计

可参考以下 demo：

http://docs.kissyui.com/kissy-editor/demo/cdn/1.1.7/wordCount.html

其中 wordcount 代码 自行下载，随意修改。

绑定 ctrol-enter

不少的应用场景要求在编辑器编辑区域内按下 ctrol+enter 按键时，编辑器所在表单会自动提交，解决方案是监控编辑器的文档按键事件：

```
editor.ready(function () {
    KISSY.Event.on(editor.document, "keydown", function (ev) {
        if (ev.keyCode == 13 && ev.ctrlKey) {
            editor.sync();
            editor.textarea[0].form.submit();

        }
    })
});
```

其中 editor 为编辑器实例的对象，通过 `var editor=KISSY.Editor("#id",cfg)` 获取而来。

placeholder(tip) 功能

可参考以下 demo：

http://docs.kissyui.com/kissy-editor/demo/cdn/1.1.7/tip.html

其中 tip 代码 自行下载，随意修改。

4.1.2 ppt

kissy editor 开发与设计 @ webrebuild

4.1.3 demo

http://kissyteam.github.com/kissy-editor/demo/static-min-test.html

CHAPTER

FIVE

最佳编码实践

5.1 通用约定

5.1.1 文件与目录命名约定

1. 一律小写，必须是英文单词或产品名称的拼音，多个单词用连字符（-）连接。 只能出现英文字母、数字和连字符，严禁出现中文。
2. 出现版本号时，需要用字母 v 做为前缀，小版本号用点号（.）隔开，比如 global-v8.js 或 detail-v2.2.js 。
3. 该命名规范适用于 html, css, js, swf, php, xml, png, gif, jpg, ico 等前端维护的所有文件类型和相关目录。
4. js 和 css 压缩文件，统一以 -min 结尾，比如源码文件为 kissy.js，压缩后为 kissy-min.js 。

5.1.2 文件编码约定

由于历史原因，前端开发涉及的所有文本文件请统一 采用 GB2312, GBK 或 GB18030 编码 。

注意：后台采用 UTF-8 的新项目，或自主开发的项目，推荐使用 UTF-8 编码。

5.1.3 id 和 class 命名约定

1. id 和 class 的命名总规则为： 内容优先，表现为辅 。首先根据内容来命名，比如 main-nav. 如果根据内容找不到合适的命名，可以再结合表现来定，比如 skin-blue, present-tab, col-main.
2. id 和 class 名称一律小写，多个单词用连字符连接，比如 recommend-presents.
3. id 和 class 名称中只能出现小写的 26 个英文字母、数字和连字符（-），任何其它字符都严禁出现。
4. id 和 class 尽量用英文单词命名 。确实找不到合适的单词时，可以考虑使用产品的中文拼音，比如 wangwang, dating. 对于中国以及淘宝特色词汇，也可以使用拼音，比如 xiaobao, daigou. 除了产品名称和特色词汇，其它任何情况下都严禁使用拼音。
5. 在不影响语义的情况下，id 和 class 名称中可以适当采用英文单词缩写，比如 col, nav, hd, bd, ft 等，但切忌自造缩写。
6. id 和 class 名称中的第一个词必须是单词全拼或语义非常清晰的单词缩写，比如 present, col.
7. 仅在 JavaScript 代码中当作 hook 用的 id 或 class, 命名规则为 J_UpperCamelCase（请注意，J_ 后的首字母也大写！），其中字母 J 代表 JavaScript, 也是钩子（hook）的象形。注意：如果在 JavaScript 和 CSS 中都需要用到，则不用遵守本约定。

8. 在自动化测试脚本中当作 hook 用的 class, 命名规则为 T_UpperCamelCase, 其中字母 T 代表 Test.

5.2 CSS 编码规范

- 待整理

5.3 HTML 编码规范

5.3.1 基本规范

- 使用符合语义的标签书写 HTML 文档，选择恰当的元素表达所需的含义;
- 元素的标签和属性名必须小写，属性值必须加双引号;
- 元素嵌套遵循 (X)HTML Strict 嵌套规则，推荐使用Firefox插件 HTML Validator 进行检查;
- 正确区分自闭合元素和非自闭合元素。非法闭合包括：
..</br>、<script />、<iframe />，非法闭合会导致页面嵌套错误问题;
- 通过给元素设置自定义属性来存放与 JavaScript 交互的数据，属性名格式为 data-xx (例如：data-lazyload-url)

5.3.2 文档模板

```
<!doctype html>
<html>
<head>
<meta charset="utf-8" />
<title>Sample page</title>
<link rel="stylesheet" href="css_example_url" />
<script src="js_example_url"></script>
</head>
<body>
<div id="page">
    <div id="header">

    </div>
    <div id="content">

    </div>
    <div id="footer">

    </div>
</div>
<script>
//
</script>
</body>
</html>
```

5.3.3 DOCTYPE

页面文档类型统一使用HTML5 DOCTYPE。代码如下：

```
<!doctype html>
```

5.3.4 编码

声明方法遵循HTML5的规范。

```
<meta charset="utf-8" />
```

5.3.5 注释

建议对超过10行的页面模块进行注释，以降低开发人员的嵌套成本和后期的维护成本。例如：

```
<div id="sample">
    ...
</div> <!-- #sample END -->

<div class="sample">
    ...
</div> <!-- .sample END -->
```

5.3.6 元素

结构性元素

- p 表示段落。只能包含内联元素，不能包含块级元素;
- div 本身无特殊含义，可用于布局。几乎可以包含任何元素;
- br 表示换行符;
- hr 表示水平分割线;
- h1-h6 表示标题。其中 h1 用于表示当前页面最重要的内容的标题;
- blockquote 表示引用，可以包含多个段落。请勿纯粹为了缩进而使用 blockquote，大部分浏览器默认将 blockquote 渲染为带有左右缩进;
- pre 表示一段格式化好的文本;

头部元素

- title 每个页面必须有且仅有一个 title 元素;
- base 可用场景：首页、频道等大部分链接都为新窗口打开的页面;
- link link 用于引入 css 资源时，可省去 media(默认为all) 和 type(默认为text/css) 属性;
- style type 默认为 text/css，可以省去;

- `script` type 属性可以省去; 不赞成使用lang属性; 不要使用古老的<!– //–>这种hack脚本, 它用于阻止第一代浏览器（Netscape 1和Mosaic）将脚本显示成文字;
- `noscript` 在用户代理不支持 JavaScript 的情况下提供说明;

文本元素

- `a` a 存在 href 属性时表示链接，无 href 属性但有 name 属性表示锚点;
- `em,strong` em 表示句意强调，加与不加会引起语义变化，可用于表示不同的心情或语调; strong 表示重要性强调，可用于局部或全局，strong强调的是重要性，不会改变句意;
- `abbr` 表示缩写;
- `sub,sup` 主要用于数学和化学公式，sup还可用于脚注;
- `span` 本身无特殊含义;
- `ins,del` 分别表示从文档中增加(插入)和删除

媒体元素

- `img` 请勿将img元素作为定位布局的工具，不要用他显示空白图片; 必要时给img元素增加alt属性;
- `object` 可以用来插入Flash;

列表元素

- `dl` 表示关联列表，dd是对dt的解释; dt和dd的对应关系比较随意：一个dt对应多个dd、多个dt对应一个dd、多个dt对应多个dd，都合法; 可用于名词/单词解释、日程列表、站点目录;
- `ul` 表示无序列表;
- `ol` 表示有序列表, 可用于排行榜等;
- `li` 表示列表项，必须是ul/ol的子元素;

表单元素

- 推荐使用 button 代替 input，但必须声明 type;
- 推荐使用 fieldset，legend 组织表单
- 表单元素的 name 不能设定为 action, enctype, method, novalidate, target, submit 会导致表单提交混乱

5.3.7 参考文档

- http://www.w3.org/TR/html4/
- http://www.w3.org/TR/html5/
- http://reference.sitepoint.com/html/

5.4 JavaScript 语言规范

- 声明变量时，必须加上 `var` 关键字。
- 尽量减少全局变量的使用。
- 语句总是以分号结尾。
- 不要在块内声明函数。
- 标准特性优于非标准特性（如果类库有提供，优先使用类库中的函数）。
- 不要封装基本类型。
- 只在解析序列化串时使用 `eval()` 。
- 禁止使用 `with` 。
- 减少使用 `continue` 和 `break` 。
- 仅在函数内使用 `this` 。
- 使用 `Array/Object` 直接量，避免使用 `Array/Object` 构造器。
- 禁止修改内置对象的原型。

5.5 JavaScript 编码风格

5.5.1 行与缩进

语句行

1. 尽可能不要让每行超过 120 个字符;
2. 语句必须以分号作为结束符, 不要忽略分号;

空格

1. 数值操作符(如, +/-/*/% 等)两边留空;
2. 赋值操作符/等价判断符两边留一空格;
3. for 循环条件中, 分号后留一空格;
4. 变量声明语句, 数组值, 对象值及函数参数值中的逗号后留一空格;
5. 空行不要有空格;
6. 行尾不要有空格;
7. 逗号和冒号后一定要跟空格;
8. 点号前后不要出现空格;
9. 空对象和数组不需要填入空格;
10. 函数名末尾和左括号之间不要出现空格;

空行

1. 逻辑上独立的代码块使用空行分隔;
2. 文件末尾留 1~2 个空行;
3. 不要吝啬空行。尽量使用空行将逻辑相关的代码块分割开，以提高程序的可读性。

缩进

1. 以 4 个空格为一缩进层次;
2. 变量声明:
   - 多个变量声明时, 适当换行表示;
   - 参照 var 关键字, 缩进一层次;
3. 函数参数:
   - 函数参数写在同一行上;
   - 传递匿名函数时, 函数体应从调用该函数的左边开始缩进;
4. 数组和对象初始化时:
   - 如果初始值不是很长, 尽量保持写在单行上;
   - 初始值占用多行时, 缩进一层次;
   - 对象中, 比较长的变量/数值, 不要以冒号对齐;
5. 二元/三元操作符:
   - 操作符始终跟随前行;
   - 实在需要缩进时, 按照上述缩进风格;
6. 表达式中的缩进同变量声明时;

5.5.2 括号

原则: 不要滥用括号, 必要时一定要使用.

1. if/else/while/for 条件表达式必须有小括号;
2. 语句块必须有大括号;
3. 一元操作符(如 delete， typeof， void)或在某些关键词(如 return， throw， case， new) 之后, 不要使用括号;

5.5.3 变量

1. 变量如有较广的作用域, 使用全局变量; 如果是在类中, 可以设计成为一个类的成员;
2. 函数体中, 多个局部变量集中在一起声明, 避免分散;
3. 适当延迟变量的初始化;

5.5.4 字符串

1. JS 代码中, 单行字符串使用单引号;
2. JS 代码中, 多行字符串使用 + 拼接形式, 不要使用 \ 拼接;
3. HTML 中 `Element` 属性, 使用双引号;

5.5.5 命名规范

原则： * 尽量避免潜在冲突; * 精简短小, 见名知意;

1. 普通变量统一使用驼峰形式;
2. 常量使用全部大写, 多个单词以下划线分隔;
3. 枚举量, 同常量;
4. 私有变量, 属性和方法, 名字以下划线开头;
5. 保护变量, 属性和方法, 名字同普通变量名;
6. 方法和函数的可选参数, 名字以 `opt_` 开头, 使用 `@param` 标记说明;
7. 方法和函数的参数个数不固定时:
    - 可添加参数 `var_args` 为参数个数;
    - 取代使用 `arguments`;
    - 使用 `@param` 标记说明;
8. `Getter/Setter` 命名:
    - 以 `getFoo/setFoo(value)` 形式;
    - 布尔类型使用 `isFoo()/hasFoo()/canDo()/shouldDO()` 也可;
9. 命名空间:
    - 为全局代码使用命名空间, 如 `sloth.*`;
    - 外部代码和内部代码使用不同的命名空间;
10. 重命名那些名字很长的变量, 不要在全局范围内创建别名, 而仅在函数块作用域中使用;
11. 文件名应全部使用小写字符, 且不包含除 - 和 _ 外的标点符号;
12. 临时的重复变量建议以 `i`, `j`, `k`, ..., 命名;

CHAPTER

SIX

前端常用工具

6.1 KISSY Module Compiler

by 承玉

6.1.1 简介

- 是一个模块代码依赖自动合并工具。
- 是一个模块代码规范辅助工具，可以辅助规范模块编写，不仅仅只适用动态加载场景，也可以用来提高开发阶段效率。
- 是一个模块代码部署工具。结合 KISSY 1.2 Loader ，用于模块代码部署阶段，多个模块根据其依赖合并为一个文件，减少 HTTP 链接数。

6.1.2 举例

开发阶段

例如：现在有两个模块代码文件 code/package/x.js 与 code/package/y.js

x.js 中内容如下

```
KISSY.add(function(){
    return "x";
});
```

y.js 中内容如下

```
KISSY.add(function(){
    return "y";
},{requires:["./x"]});
```

x.js 路径为 http://x.cn/code/package/x.js 开发阶段在页面中通过包配置 KISSY.config 可以这样使用：

引入 KISSY

```
<script src='kissy.js'></script>
```

使用模块

```
KISSY.config({
    name:"package", //
    tag:"20110323", //      js
                    //    ?t=20110323
    path:"http://x.cn/code/", //
    charset:"gbk" //
});

KISSY.use("package/y",function(S,Y){
    // Y == "y"
});
```

那么 KISSY Loader 就会

1. 加载 y.js 。
2. 加载 x.js 。
3. 初始化 package/x 模块。
4. 初始化 package/y 模块。
5. 回调中可以得到 package/x 的模块值了。

这个过程如果模块多的话对于线上环境来说 HTTP 链接数不可承受，使用 KISSY module compiler 则可以达到所有模块压缩为一个文件，达到 http 链接数为 1 的目标。

使用 Module Compiler

简要如下:

1. 配置 module compiler 指定需要的模块，这里即 package/y
2. 配置 module compiler 指定模块查找目录，这里即 code/
3. 配置 module compiler 指定合并后文件名称，假设为 package/y.combine.js
4. 运行 module compiler 合并 package/y 及其递归依赖的其他模块到 package/y.combine.js
5. 运行 closure compiler 压缩 package/y.combine.js 为 package/y-min.js

线上部署阶段

载入 kissy 的压缩版本

```
<script src='kissy-min.js'></script>
```

使用模块部分同 开发阶段 。

线上过程：

1. 加载 package/y-min.js
2. 检查 package/x 模块，发现已经载入
3. 初始化 package/x 模块
4. 初始化 package/y 模块
5. 执行回调

于是线上应用 HTTP 链接数为 1 。若需要使用源码调试则可以在页面 url 后加上 ?ks-debug 即可开启开发阶段的加载过程.以上即为近期交易平台化项目所采用的模块化框架。

6.1.3 使用说明

该工具为一个 jar 包，推荐嵌入到 ant 中，作为项目构建的一个阶段.

Note: ant 以及相关压缩工具可直接使用 kissy-tools 中的文件， 建议直接下载整个 kissy-tools 作为项目工具一部分。

使用前请先下载 kissy 源码到本地硬盘 x:/kissy/ ，kissy-tools 源码到硬盘 x:/kissy-tools/ ，假设项目源码在 x:/code/ ，如上一节包括两个模块文件: x:/code/package/x.js 以及 x:/code/package/y.js ，则 ant 的调用为：

```
<java classname="com.taobao.f2e.Main">

    <arg value="-requires"/>
    <arg value="package/y"/>
    <arg value="-baseUrls"/>
    <arg value="x:/kissy/,x:/code/"/>
    <arg value="-encodings"/>
    <arg value="utf-8,gbk"/>
    <arg value="-outputEncoding"/>
    <arg value="utf-8"/>
    <arg value="-excludes"/>
    <arg value="core"/>
    <arg value="-output"/>
    <arg value="x:/code/package/y.combine.js"/>

    <classpath>
        <pathelement path="x:/kissy-tools/module-compiler/module-compiler.jar"/>
        <pathelement path="${java.class.path}"/>
    </classpath>

</java>
```

如上所示该工具需要一些参数，包括 requires ，baseUrls ，encodings ，outputEncoding ，excludes ，output ，具体格式如下：

requires

需要打包的模块名集合，该模块以及其递归依赖都会被一并打包。多个模块名用 ，分隔。如以上示例：package/y 。

baseUrls

模块文件查找目录集合，一般存在两个： kissy 内置模块目录以及业务模块源码目录，目录间以 ，分隔。如以上示例 x:/kissy/,x:/code/ 。

encodings

指明模块目录中文件的编码，顺序对应于 `baseUrls` ，多个编码以 ，分隔。如以上示例： `utf-8,gbk` ， 表示 `kissy` 内置模块为 `utf-8` 编码，业务模块为 `gbk` 编码。

outputEncoding

指明最后合并打包文件的编码格式。如以上示例 `utf-8` 。

excludes

指明 `requires` 递归依赖模块中不需要打包的模块名称集合，该集合内的模块是单独动态加载或静态引入的，多个模块以 ，分隔。 如以上示例 `core` 表示 `kissy` 的 `core` 模块不需要合并打包，是通过 `kissy.js` 静态引入到页面中的。

output

指明最终合并打包后的文件路径以及名称。如以上示例： `x:/code/package/y.combine.js` ， 表示该工具产生的合并后文件为 `x:/code/package/y.combine.js` 。

6.2 Sphinx 使用介绍

6.2.1 安装及使用 Sphinx

解决依赖关系

- Python 2.x (http://www.python.org/), 并将 Python 加入到 Path 中;
- Setup Tools: Python 的一个包安装/管理工具(http://pypi.python.org/pypi/setuptools, win 下有 exe, 直接安装即可);
- 安装 docutils, cmd 下进入 kissyteam/tools/docutils-0.7, 运行 python setup.py install;

安装 Sphinx

- cmd 下运行: easy_install -U Sphinx;
- cmd 下进入 kissyteam/tools/sphinx-to-github, 运行 python setup.py install;

编译文档

- cmd 进入 kissyteam 目录, 运行 make.bat html , 在 docs 下生成 html;
- 如果你想进一步了解如何创建 sphinx 工程, 见 http://code.google.com/p/pymotwcn/wiki/SphinxprojectHowto ;

6.2.2 reST 入门

reST 是一种简单的标记语言, 规则非常简单.

See Also:

- 完整的官方文档 reStructuredText User Documentation
- 中文翻译版本: http://wstudio.web.fc2.com/others/restructuredtext.html

段落

组织文档层次, 一般标题层次少于 4 级. 可以使用 :

- # 部分
- * 章节
- =, 大节
- -, 小节
- ^, 小小节
- ", 段落

其实, 不同字符(#, =, ` , `, ^, *)没有等级区别, 只是你在使用时, 需要统一一种等级标记即可.

行内标记

- 一个星号表示斜体: *text*
- 两个星号表示粗体: **text**
- 两个反斜号表示行内代码: ``text``

使用上的限制:

- 不能被嵌套,
- 内容中不需要空白字符, 如: * text* , 这个是错误的写法,
- 使用反斜杠进行转义: thisis\ *one*\ word

无序/有序列表

```
* This is a bulleted list.
* It has two items, the second
  item uses two lines.

1. This is a numbered list.
2. It has two items too.

#. This is a numbered list.
#. It has two items too.
```

嵌套列表

```
* this is
* a list

  * with a nested list
  * and some subitems

* and here the parent list continues
```

定义列表

```
term (up to a line of text)
   Definition of the term, which must be indented

   and can even consist of multiple paragraphs
```

```
next term
   Description.
```

注意 以上, 只能是单行文字, 计算源码中有换行符, 但生成出来仍然以单行形式, 如果需要保留换行符, 可使用

```
| These lines are
| broken exactly like in
| the source file.
```

嵌入源代码区块

最普通的, 可以使用 :: 标记, 表示接下来的缩进段落是表示一段代码,

```
It is not processed in any way, except
that the indentation is removed.

It can span multiple lines.
```

需要高亮的源代码

- 嵌入 HTML 源代码

```
.. raw:: html
   :file: inclusion.html



.. raw:: html

    <a href="/kissy/docs/">KISSY  </a>
```

- 嵌入 JS/CSS 等源代码, 及高亮显示(http://pygments.org/ 上列出了支持的语言)

```
.. code-block:: js

    alert('hi');
```

表格

生成这样的表格

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

使用的语法是

```
=====  =====  =======
  A      B    A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

外部链接

使用 `  <http://example.com/>`_ 表示外部链接, 也可以使用

```
    `  `_.

.. _   : http://example.com/
```

内部链接

- 设定全局唯一的引用id, 如, `.. _my-reference-label:`
- 在其他页面就可以使用这个id来引用, 如, `:ref:` <my-reference-label>``

例如

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

特殊标记

- attention, caution, danger, error, hint, important, note, tip, warning, admonition (最常用 ``note'' 和 ``warning''), 例如

```
    .. attention::

       Please attention
```

Attention: 这是一个说明框.

Caution: 这是一个说明框.

Danger: 这是一个说明框.

Error: 这是一个说明框.

Hint: 这是一个说明框.

Important: 这是一个说明框.

Note: 这是一个说明框.

Tip: 这是一个说明框.

Warning: 这是一个说明框.

自定义说明框

这是一个说明框.

- 插入图片

```
.. image:: picture.jpeg
   :height: 100px
   :width: 200 px
   :scale: 50 %
   :alt: alternate text
   :align: right
```

注释

单行

```
..   .
```

多行

```
..
```

脚注

```
Lorem ipsum [#f1]_ dolor sit amet ... [#f2]_

.. rubric:: Footnotes

.. [#f1] Text of the first footnote.
.. [#f2] Text of the second footnote.
```

CHAPTER

SEVEN

组件开发指南

7.1 KISSY 组件开发流程

欢迎加入KISSY开发小组！
仅需几个简单的步骤，你就能跟更多人分享自己的代码，Let's Go!

7.1.1 1、检出源码

折腾派：请使用git拷贝“https://github.com/kissyteam”的所有项目
简单派：请使用svn检出“http://svn.app.taobao.net/repos/fed/trunk/fed/kissy-team”目录（限淘宝）

7.1.2 2、初始化工作文件夹

想提供 UI组件 请进入“kissy-gallery”目录，解压“your-gallery.zip”文件
想提供 小功能 请进入“kissy-util”目录，解压“your-util.zip”文件
想提供 设计模式 请进入“kissy-dpl”目录，解压“your-dpl.zip”文件

这些文件夹都为你预置好了初始开发环境
你只需要将其中的组件名改成你想写的即可快速开始工作
并在文件中合适的地方为你提供了开发指引
确保你能写出漂亮的代码

解压的文件夹都是带“.dev”后缀的
这表示你的代码还在开发中
开发完成后去掉即可

7.1.3 3、编码

7.1.4 4、提交

怎么样，真的非常简单吧

剩下的就交由KISSY管理团队处理，我们会在每周三同步两个仓库的修改，然后你的大作就能被大家欣赏了。
当你需要写非常复杂的组件，或者希望重构以提高代码质量时，我们建议你参见 KISSY 组件开发流程（完整版）.

7.2 KISSY 组件开发流程（完整版）

点击查看详细大图

7.2.1 步骤说明

请先阅读 ImageZoom 调研文档

应用场景分析

一个需求到来，比如这个图片放大效果，首先我们需要这个功能能用在哪些地方，或哪些网站上已经使用了，如果有的话，就对比一下不同的情况下不同的要求，如 #slide4 . 这样以后，对比自己的需求，想好要实现什么功能，哪些功能保留，哪些功能不需要，---- 明确需求;

同类组件调研

需求明确之后，查找现有的同类组件，看看他们针对这个问题，是怎么实现的，实现哪些功能，哪些可以借鉴的地方，哪些不足的地方要避免或者改进，如 #slide6，---- 明确要实现的功能有哪些;

功能点梳理

分离出完成整个功能需要的几个核心功能点，并针对各个功能点逐个描述，如 #slide8，这也可以帮你理清思路，---- 进一步明确待实现功能;

技术方案

针对上述的几个功能点，分别给出实现方案，或者其他的技术难点，又或者是算法上的分析等，如 #slide9，---- 明确如何实现;

Public API

设计好的公共 API，并在此说明，也可以根据使用场景，给出一些范例来说明 API 的使用，如 #slide14，这里可以在开始时设计的尽量精简些，---- 明确 API 接口；

开发计划

简略或者详细的制定一个开发计划，及发布的版本和时间等，---- 明确进度；

预研报告

预研过程后总结一个报告，可将报告分享给大家，供大家一起讨论.

我们建议每个 KISSY 组件下，都存放一个 slide.html，其内容包含上述几部分内容. 这个 slide 随着你的开发过程的推进，也会不断添加更新，最后发布时连同组件源代码一起，形成非常好的知识体系，这样，给别人或是几十年后的自己阅读，也会像看文章一样的有条理.

7.3 KISSY 组件开发规范

by 承玉

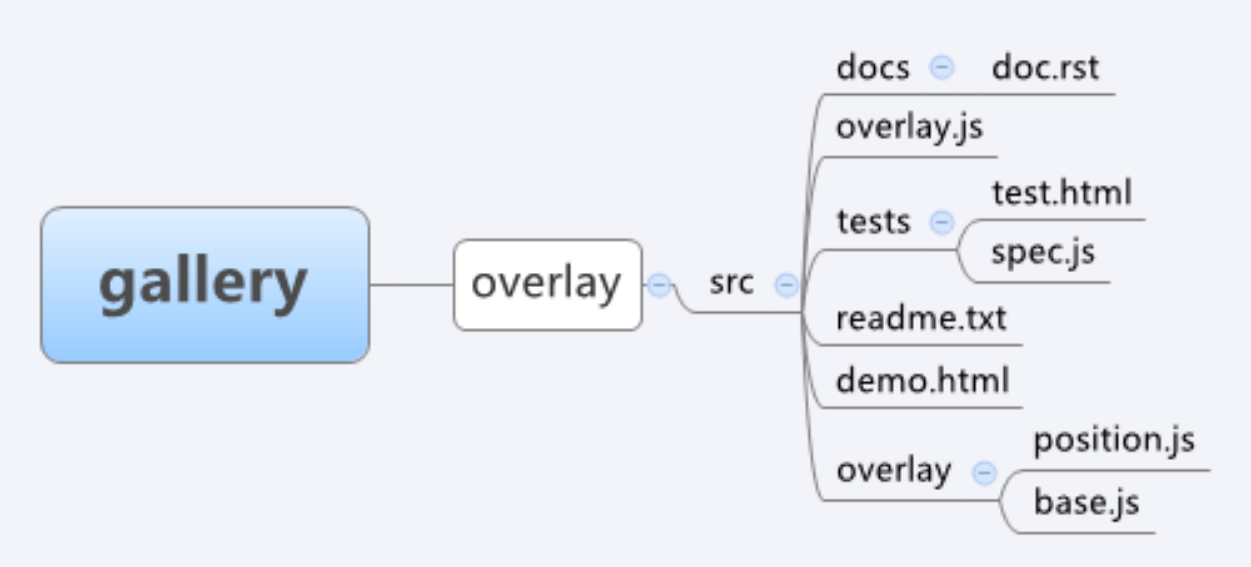
开始之前请先阅读 KISSY 组件开发流程.

7.3.1 确定 API

首先确定该组件需要公开的 api 接口包括属性名称，函数名，参数以及返回值，可参考 YUI3 ,Jquery 等类库的同类组件，尽量保持一致。 比如 Overlay，那么其公开接口肯定包含方法 `show` ，``hide`` 以及弹层内容 `content` 属性配置.

7.3.2 模块编写

必须。推荐的目录结构如下，例如组件为 Overlay 弹层，那么该组件的目录结构应为：



src 目录中必须包含和组件名相同的一个模块文件，模块名为 gallery/overlay ,用来指明该组件依赖的子模块，子模块的名约定为 gallery/overlay/xx ,如果组件比较简单也可只有这一个源码文件。例如 overlay.js

```
KISSY.add("gallery/overlay",function(S,Base){
    return Base;
},{
    './overlay/base','./overlay/position'
});
```

子模块放在 src 模块名为目录名的文件夹内，对于 KISSY 1.2 以前，需要手动将组件挂载到 KISSY 上去并且需要在模块定义处挂载，例如子模块 base.js 的编写：

```
KISSY.add("gallery/overlay/base",function(S){
    function Overlay(){}

    //    KISSY < 1.2     KISSY
    S.namespace("Gallery");
    S.Gallery.Overlay=Overlay;

    return Overlay;
});
```

子模块间也可有依赖关系，例如子模块 position.js 需要对基本模块 base.js 进行增强：

```
KISSY.add("gallery/overlay/position",function(S,Overlay){
    //  kissy < 1.2
    Overlay = S.Gallery.Overlay;

    Overlay.prototype.xx=function(){};

},{
    requires:['./base']
});
```

7.3.3 demo 编写

必须。写一个 demo.html 简单展示下这个组件怎么用，静态载入组件的所有依赖js即可，注意被依赖模块js要放在依赖js前面，例如：

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>overlay demo</title>
    </head>
    <body>
        <script src='../../../kissy/build/kissy.js'></script>
        <script src='base.js'></script>
        <script src='position.js'></script>
        <script src='overlay.js'></script>
        <script>
            KISSY.use("gallery/overlay",function(S,Overlay){
                // kissy < 1.2
                Overlay=S.Gallery.Overlay;
            });
```

```
        </script>
    </body>
</html>
```

7.3.4 readme.txt 编写

7.3.5 文档编写

可选。在 docs 目录下编写组件文档，后缀名为 rst ，可参照 KISSY Overlay 的文档 api 以及 使用文档 ，详细格式可参见 sphinx 。文档不做强求，也可直接写纯文本格式，在 demo.html 详细讲解即可。

7.3.6 单元测试编写

可选。在 tests 目录下编写单元测试代码，单元测试包括两个部分，测试准备页面以及单元测试用例脚本。

测试准备页面

编写 test.html ，引入单元测试框架 jasmine (在 kissy/tools/ 下)，例如：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Overlay Test Runner</title>
        <link rel="stylesheet" href="../../../tools/jasmine/jasmine.css">
        <script src="../../../kissy/tools/jasmine/jasmine.js"></script>
        <script src="../../../kissy/tools/jasmine/jasmine-html.js"></script>
        <script src="../../../kissy/tools/jasmine/event-simulate.js"></script>
        <script src="../../../kissy/build/kissy.js"></script>
    </head>
    <body>
        <script src='base.js'></script>
        <script src='position.js'></script>
        <script src='overlay.js'></script>
        <script src="overlay-spec.js"></script>
        <script>
            jasmine.getEnv().addReporter(new jasmine.TrivialReporter());
            jasmine.getEnv().execute(function() {
                if (parent && parent.jasmine.kissyNext) {
                    parent.jasmine.kissyNext(this.results().failedCount);
                }
            });
        </script>
    </body>
</html>
```

测试用例脚本编写

测试用例编写在脚本 overlay-spec.js 中，详细可参考 jasmine wiki ，这里简单举个例子：

```
//        suit
describe("   suit",function(){

    //   suit    spec
    it("   spec",function(){

        /*
              spec    expectation
        */
        expect("xx").toBe("xx");
        expect("yy").toBe("yy");

    });

});
```

复杂点的例子可以看 KISSY.Overlay Unit Test

PYTHON MODULE INDEX