



KISSY Component Model

组件模型

yiminghe@gmail.com

阶段1

- 代表: Suggest



代码采样

- //默认配置

```
var defaultConfig={};
```

```
function Suggest(config) {
```

```
  S.mix(config,defaultConfig,false);
```

```
  //初始化
```

```
}
```

```
//功能
```

```
S.augment(Suggest,EventTarget,{});
```

阶段1特点

- 单类单文件
- 逻辑（数据，控制，渲染）集中
- 维护性不佳

阶段2

- 代表：Switchable



给力秒杀 家装大牌联合巨献



公告 论坛 规则 安全保障 公益

- 政协常委建言扶植电商
- 高位瘫痪 开网店创业
- 卖家主动提醒买家维权
- 淘宝购物你有信心吗?



代码采样

- -----switchable.js

//主要逻辑

```
function Switchable() {}
```

-----switchable-plugin.js

//插件逻辑

```
function SwitchPlugin() {}
```

-----widge-tab.js

//扩展逻辑

```
function Tab() {}
```

```
S.extend(Tab, Switchable);
```

阶段2特点

- 总体逻辑初步文件分离.
- 单个逻辑功能（数据，控制，渲染）集中.
- 分离机制不能被其他组件公用，仅适合自身.

阶段3

- 代表：无
- 中间阶段
- Inspired by yui3 ,use base/attribute

代码采样

- ```
function Component() {
 this.on("afterXChange", this.afterXChange, this);
}

Component.ATTRS={
 x:{
 getter:function() {
 }
 };

S.extend(Component, S.Base, {
 afterXChange:function() {}
});

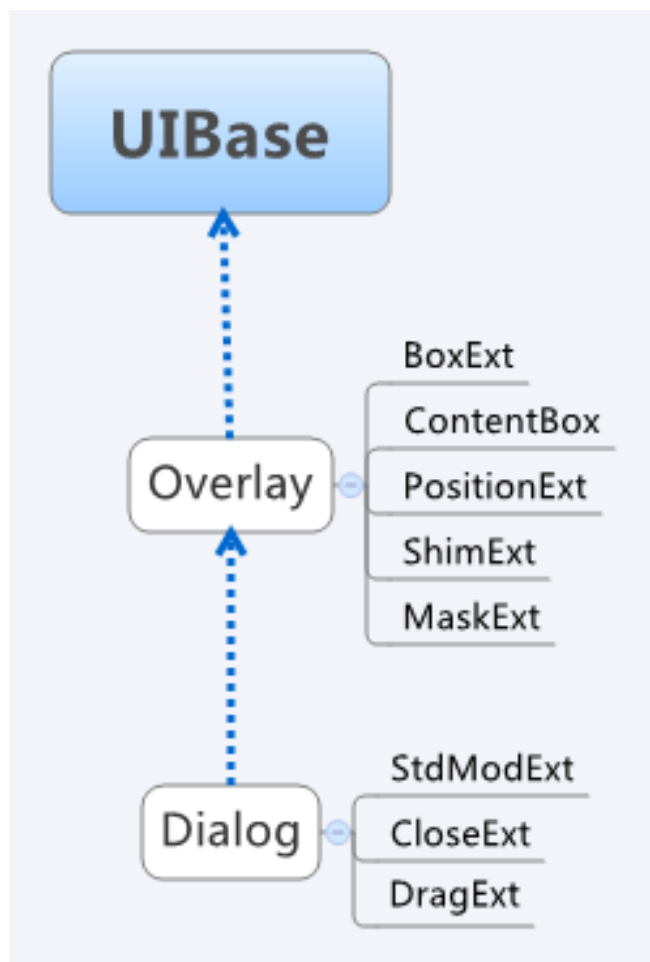
//----- use
new Component().set("x", 1);
new Component().get("x");
```
- 
- The diagram illustrates the flow of data in the provided code. A blue arrow originates from the 'afterXChange' function definition within the 'Component.ATTRS' object and points to the 'set' method call in 'new Component().set("x", 1);'. A second blue arrow points from the 'get' method call in 'new Component().get("x");' back to the 'getter' function within the 'x' attribute definition. A green dashed line with the label 'use' connects the 'afterXChange' function to the 'get' method call, indicating that the 'afterXChange' function is used by the 'get' method.

# 阶段3特点

- 完善的属性管理机制
  - 组件状态同 dom 解耦
  - 数据与渲染逻辑分离
  - 自动状态数据同步
- 
- Refer:<http://goo.gl/if8k9>

# 阶段4

- 代表：Overlay using UIBase



# 代码采样

- `var Component=UIBase.create(Parent,['MixComponent'], {`

`initializer:function() {},`

`_uiSetX:function() {},`

`renderUI:function() {},`

`bindUI:function() {},`

`//syncUI:function() {}`

`destructor:function() {}`

`}, {`

`ATTRS:{`

`x:{}`

`},`

`HTML_PARSER:{`

`yEl:".cls"`

`}`

`});`

属性x变化处理逻辑

渲染周期，生成html

绑定事件，控制逻辑

可以声明属性特征

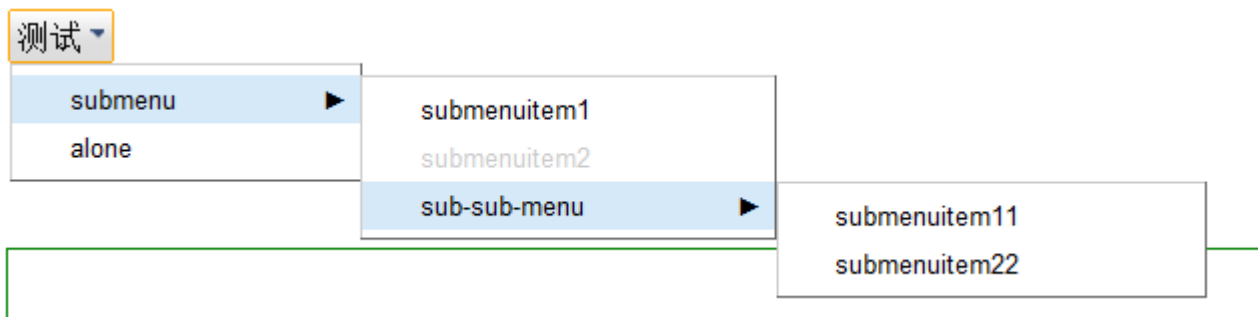
从html中获取组件属性

# 阶段4特点

- 自动属性变化关联
  - 生命周期管理
  - 扩展 mixin 支持
- 
- Refer: <http://goo.gl/sgvXP>

# 阶段5

- 代表: menu,menubutton
- Component , MVC 分离



# ModelControl

- ```
ModelControl=UIBase.create([UIBase.Box], {  
  renderUI:function() { //通知view和子组件view渲染},  
  bindUI:function() { //注册常用dom事件},  
  destructor:function() { //销毁自己以及子组件}  
}, {  
  ATTRS:{  
    view://默认渲染层, 取构造器.DefaultRender  
    children://子组件  
    parent://父组件  
  }  
})
```

Render

- 仅控制渲染盒子，申明一些默认属性
- ```
var Render =
 UIBase.create([UIBase.Box.Render], {}, {
 ATTRS:{
 //从 markup 中渲染
 srcNode:{},
 //组件 CSS class 前缀
 prefixCls:{ value:"" },
 //是否禁用
 disabled:{ value:false}
 }
 }) ;
```



# 定制组件

```
//----- render.js

var ComponentRender=UIBase.create(Component.Render,[...],{
 renderUI:function() {}, //渲染
}, {
 HTML_PARSER:{ yEl:function() {} } //markup 中渲染
});

//----- modelcontrol.js

var Component=UIBase.create(Component.ModelControl,[...],{
 bindUI:function() {},
 _handleClick:function() {}, //override 点击时操作
 _uiSetX:function() { //处理逻辑, 转发 view}
}, {
 {
 ATTRS:{x:{}},
 DefaultRender:ComponentRender
 }
});
```

# 使用组件

- `var comp=new Component ({}); //根据配置初始组件`
- `comp.addChild(new SubComponent ({})) //添加子组件`
- 
- `comp.render (); //渲染组件，开始运作`

# 阶段5特点

- UIBase MVC 逻辑分离
- Render : 渲染逻辑
- ModelControl: 数据与控制逻辑