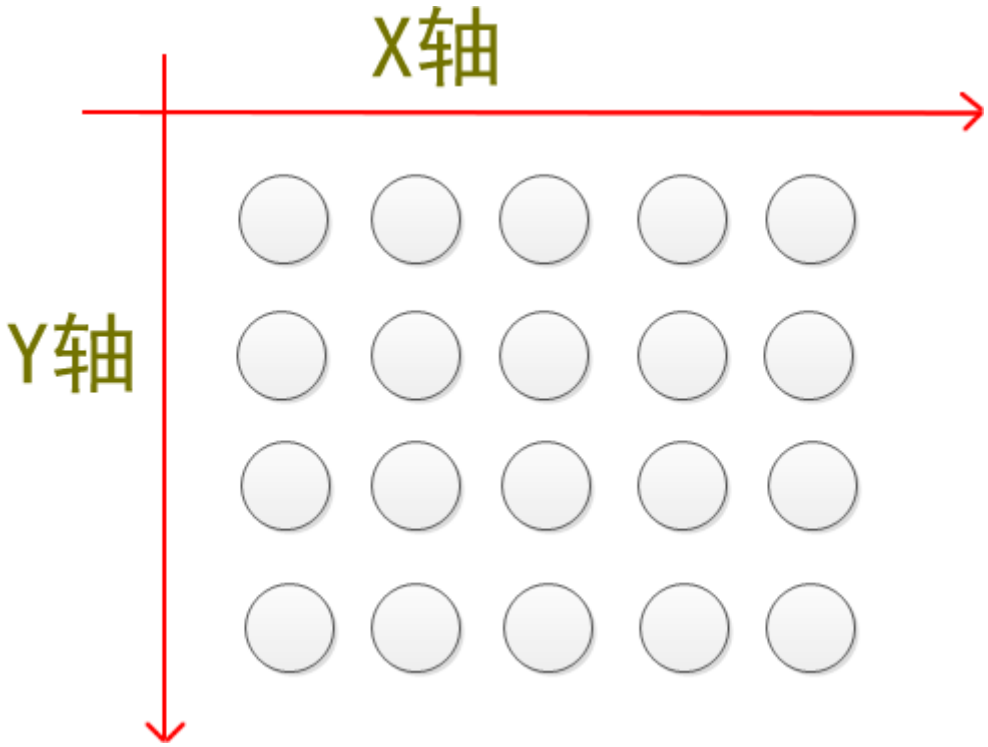
一、FocusNode

FocusNode是一个光标节点构造器。先看一下 FocusNode的结构：

```
FocusNode = function (args) {
    args        = args || {};
    this.x      = args.x || 0; //光标在坐标系中的x值
    this.y      = args.y || 0; //光标在坐标系中的y值
    this.status = ('undefined' == typeof(args.status)) ? 1 : args.status;//光标状态，是否可用
    this.data   = args.data || null;//光标节点需要保存的数据
    this.id     = args.id || null;//光标节点id
    this.parent = null;//父节点

    //以下几个事件，只有FocusNode集合节点才会被执行
    this.cache          = args.cache || 0;//光标跳出该集合时，是否记住光标所在子结点的ID
    this.saved_id       = args.saved_id || null;//光标跳出该集合时，所在子结点的ID，当光标再次跳入此集合是恢复该节点的亮起状态
    this.saved_dir      = args.saved_dir || null;//光标跳出该集合时的方向
    this.default_node   = args.default_node || null;//第一次进来默认高亮的子结点
    this.children       = [];//子结点
    this.alive_children = null;//活着的子结点
    this.is_collection  = 0;//是否是光标集合，通过是否有子结点来判断，如果有调用过addChild说明是一个实体光标节点，没有则说明是一个光标集合
};
```

可以看到每一个光标节点，在二维坐标系中都拥有一个唯一的位置(x, y)。

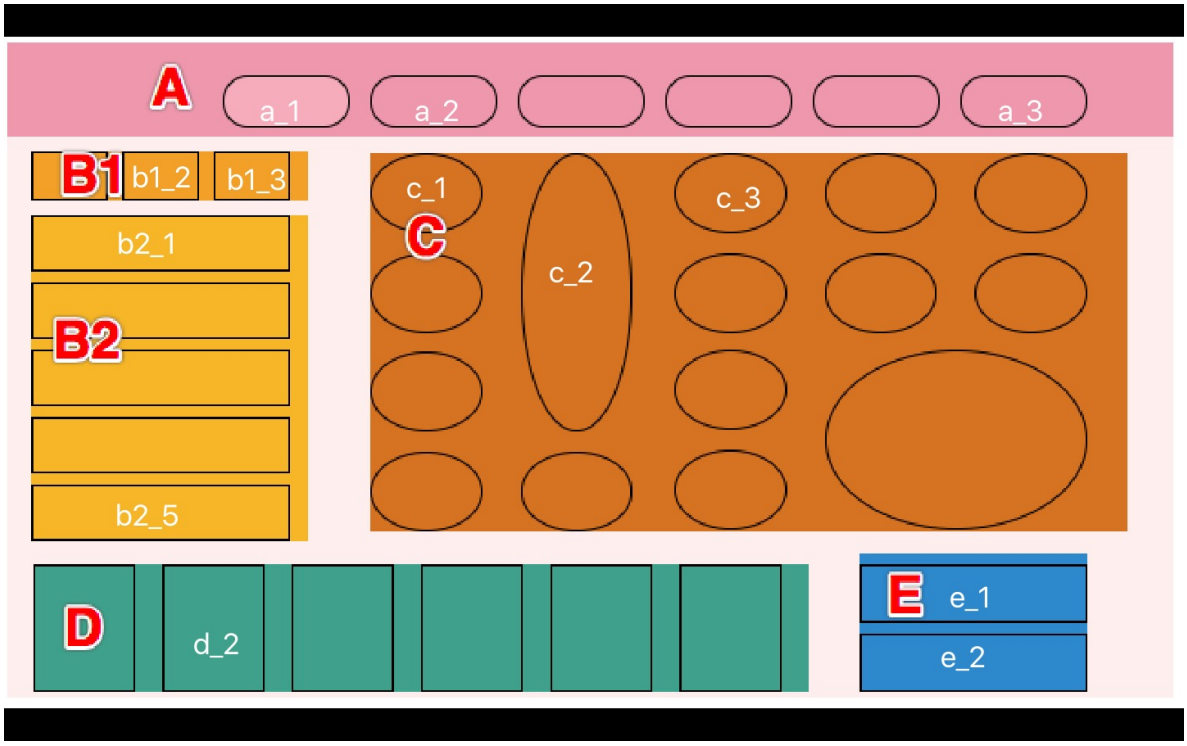
利用坐标系，光标节点就可以通过上下左右的移动，移动到想要的光标节点上。



光标节点可以拥有子节点，存放在属性children数组里面，这表示FocusNode可以嵌套。这个结构类似多叉树，根据有无子节点，我们将没有子节点的光标节点定义为**光标按钮节点**，光标按钮节点往往对应一个按钮，光标可以落焦；有子节点的光标节点定义为**光标集合节点**。

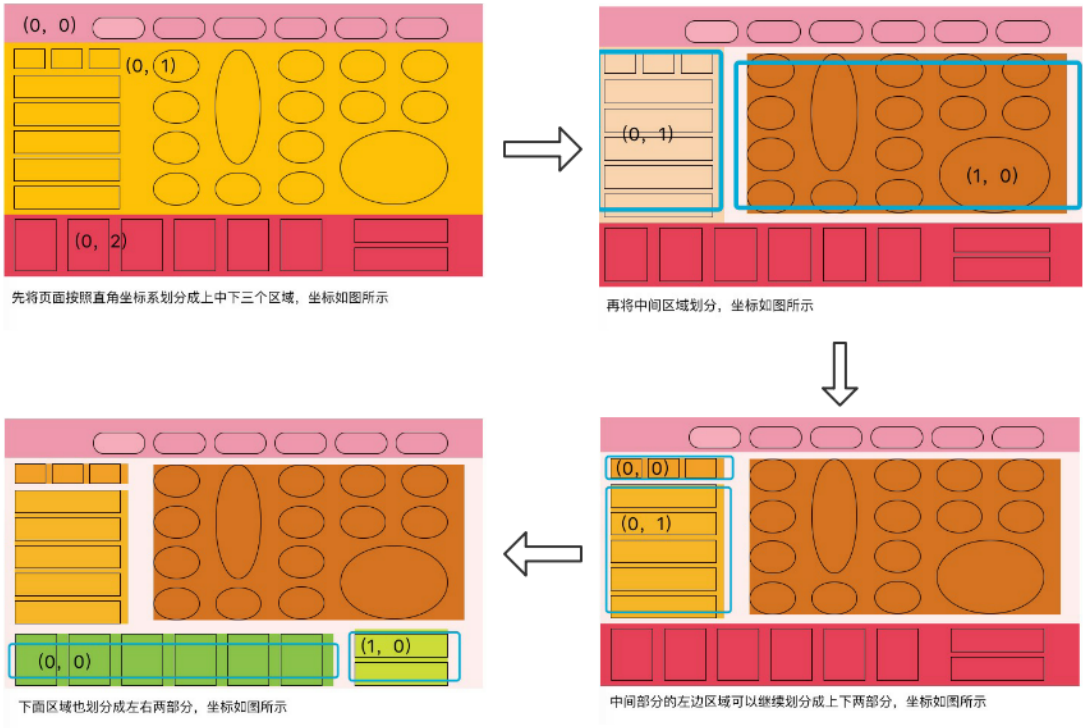
为什么要将光标节点分类呢?

我们先看一下这张图 a_1按钮节点的光标怎么移到b1_2节点上呢？对于这种无规律的页面结构，首先需要将有规律的节点合并成集合，直到页面节点区域有规律。

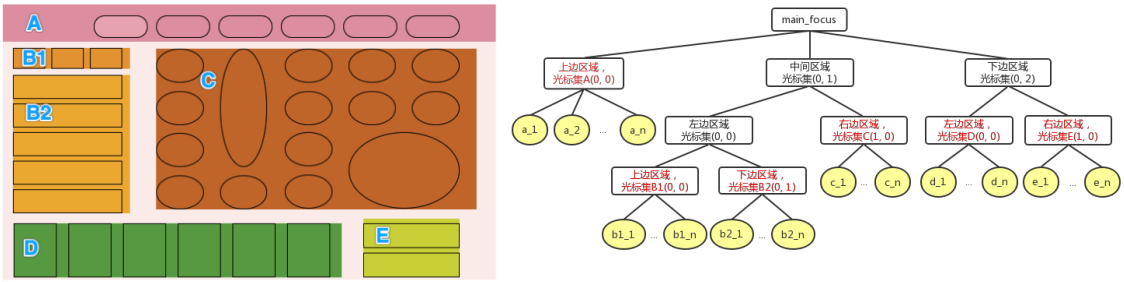


不过，比较推荐的方法是拿到页面结构后，从大往小划分。

最后，



得到的划分结果如下图:



光标按钮节点和光标集合节点的异同

现在我们知道了两种光标节点类型，那这两种类型之间有什么异同吗?

- 相同点：都是基于FocusNode
- 不同点：

    1. HTML创建DIV时，可以定义的属性不同。

    按钮节点：name、x、y、id、status、url、history、auth；

    集合节点：id、name、x、y、status、default、filter；

    2. 可以响应的事件不同。

    光标按钮节点可以响应的事件有：

    on : function () {...}, //光标落焦按钮时调用的方法

    lost : function () {...}, //光标离开按钮时调用的方法

    ok : function () {...}, //光标按钮点击ok时调用的方法

    集合按钮节点可以响应的事件有：

    enter : function () {...}, //光标进入集合时调用的方法

    leave : function () {...}, //光标离开集合时调用的方法

left : function () {...}, //光标在集合左边界再往左移时调用的方法

right : function () {...}, //光标在集合右边界再往右移时调用的方法

bottom : function () {...}, //光标在集合下边界再往下移时调用的方法

top : function () {...}, //光标在集合上边界再往上移时调用的方法

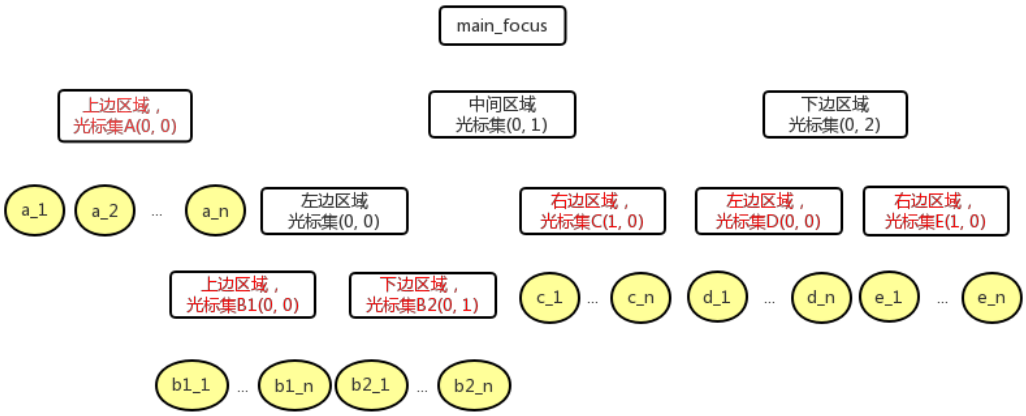
二、bjf.focus

bjf.focus是一个页面自动创建的控制和管理光标的对象。

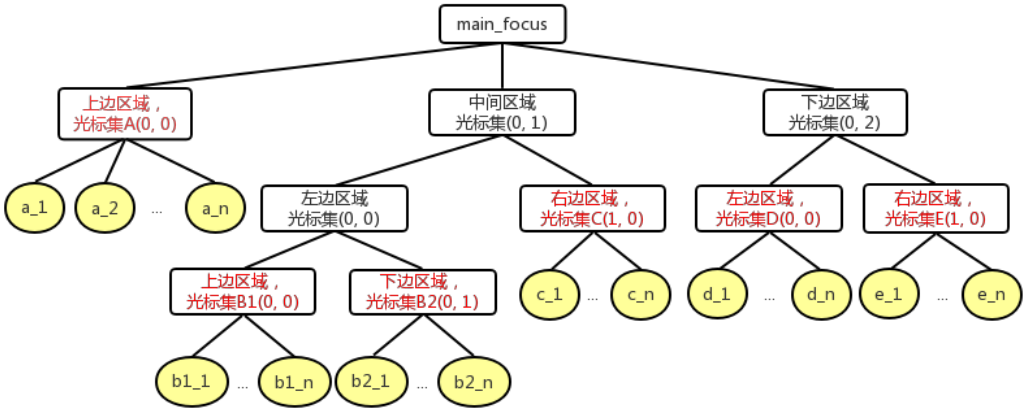
列举3个bjf.focus对象的主要功能。

- bjf.focus对象负责关联光标节点创建光标系统

  光标按钮节点和光标集合节点的层级类似于多叉树里的终端节点和非终端节点。还是拿上面的例子，这张图:



  我们用FocusNode创建好这些光标按钮节点和光标集合节点后，就需要用bjf.focus的build方法将这些节点联系起来。得到下面这张图，



- bjf.focus对象负责管理当前落焦节点。先看下bjf.focus对象的结构:

```
bjf.focus = {
        locked        : 0, //光标是否锁定，锁定后不可移动。手工change也无效
        node          : null, //当前激活的光标节点
        dir           : 0, //1上，2下，3左，4右
        DIR_UP        : 1,
        DIR_DOWN      : 2,
        DIR_LEFT      : 3,
```

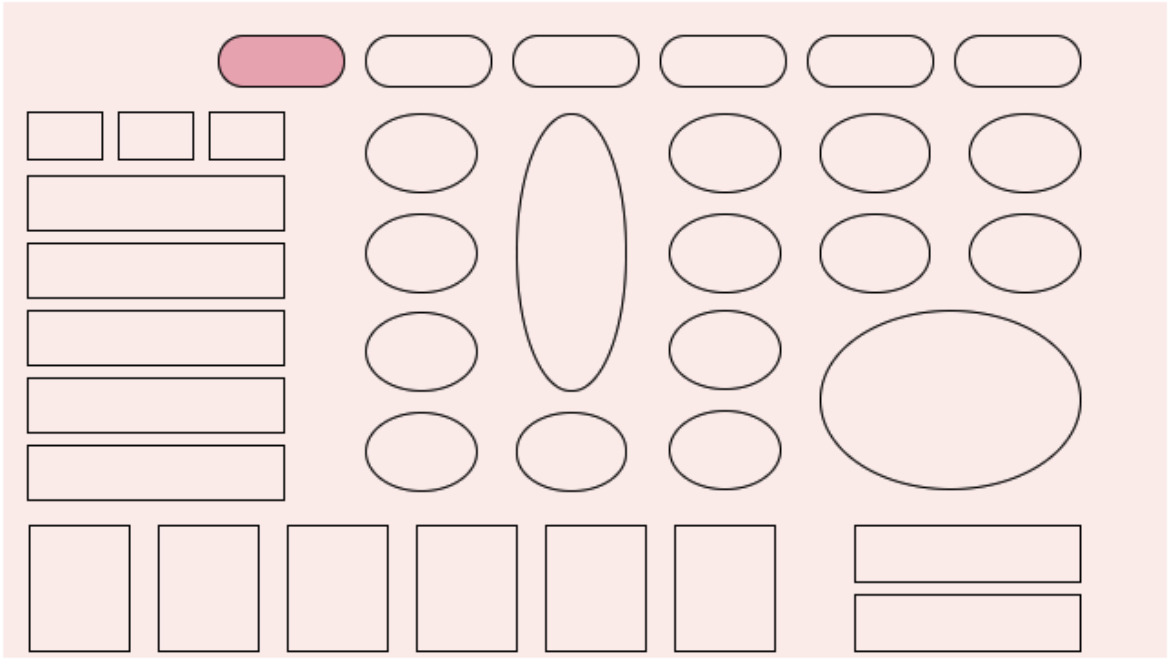
```
        DIR_RIGHT     : 4,
        eventDefault  : {}, //默认的光标事件响应
        eventRegistry : {}, //光标事件管理器
        init          : function (node_) {...}, //初始化光标节点，在一开始页面还
没有光标时，调用此方法
        initByhistory : function (main_node, history) {}, //根据历史记录，初始
化光标节点
        change        : function (next_, direction_) {...}, //光标移动，让上一
个按钮的光标消失，让下一个按钮的光标亮起
        up            : function () {...}, //光标上移
        down          : function () {...}, //光标下移
        left          : function () {...}, //光标左移
        right         : function () {...}, //光标右移
        ok            : function () {...}, //光标所在的按钮被点击
        getHistory    : function () {...}, //获取当前光标节点的路径，以供保存历史
记录
        addEventListener: function (id_, name_, listener_) {...}, //为光标集
或光标节点添加监听方法
        build         : function (ele_, parent_) {...} //为ele_下所有元素创建光
标
}
```

在页面上始终只有一个光标按钮节点是激活的（处于落焦状态，就是这个node记录的节点），bjf.focus对象就是管理哪个光标按钮节点是激活的，激活落焦时调用的方法，变成非激活时调用的方法。

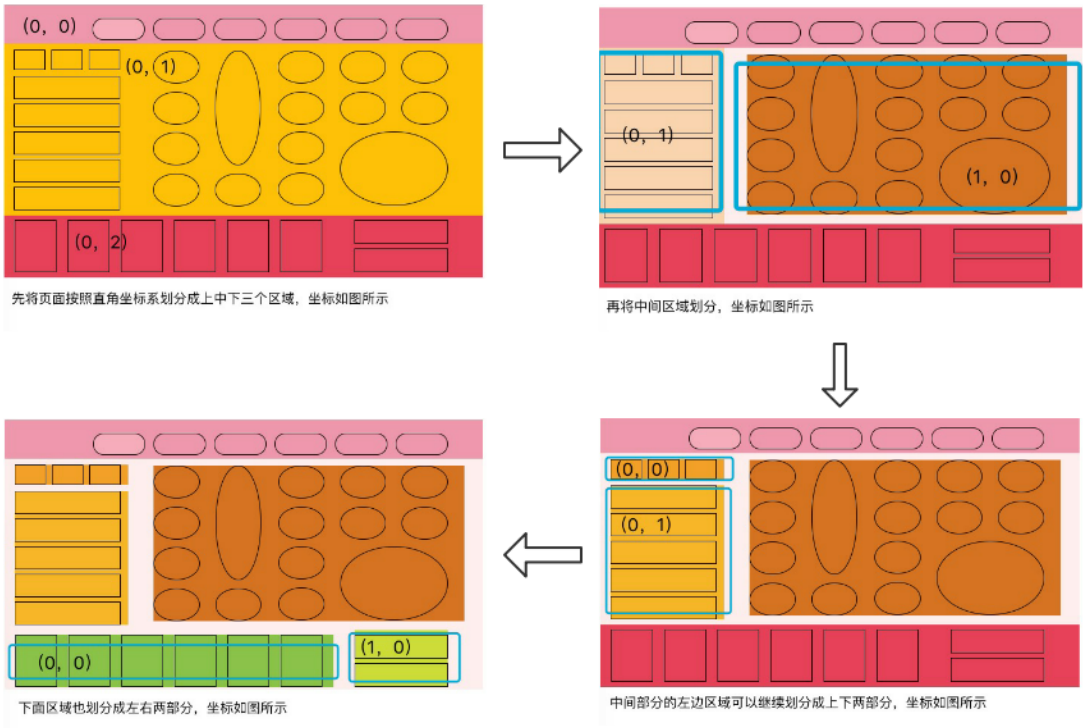
- bjf.focus对象负责初始化光标节点，指定第一次激活的光标节点。
- bjf.focus对象在光标上下左右移动时负责查找下一个落焦光标按钮节点。

三、示例1-创建基础光标系统

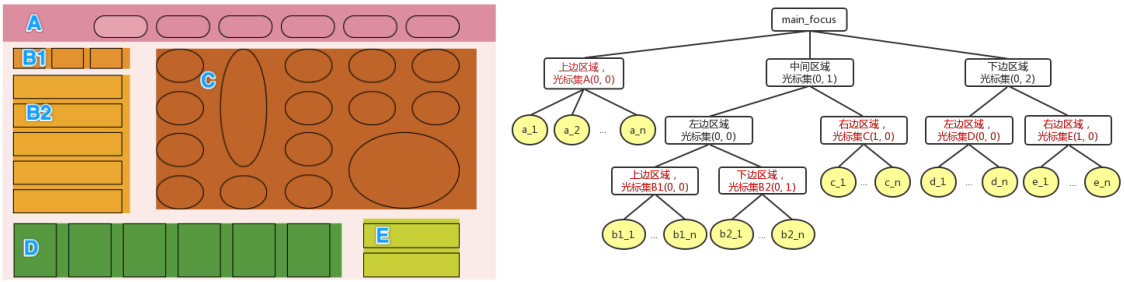
页面效果图：



我们还是拿之前页面举例，开发人员拿到效果图后，应先根据页面结构按照二维坐标系（x轴、y轴）将其递归划分到最小光标集合节点，也就是终端节点的上一层。



从大到小划分，最终得到这个结果：



接下来，就可以开始写HTML页面了，先将最基础的DIV元素和CSS写好，得到如下代码（因为css代码不是这次重点，这边先省略）：

```
<div class="main">
    <div class="area_A">
        <div class="a"></div>
        <div class="a"></div>
        <div class="a"></div>
        <div class="a"></div>
        <div class="a"></div>
        <div class="a"></div>
    </div>
    <div class="height458">
        <div class="area_B">
            <div class="area_B1">
                <div class="b1"></div>
                <div class="b1"></div>
                <div class="b1"></div>
            </div>
            <div class="area_B2">
                <div class="b2"></div>
                <div class="b2"></div>
```

```
                <div class="b2"></div>
                <div class="b2"></div>
                <div class="b2"></div>
            </div>
        </div>
        <div class="area_C">
            <div class="round8-h">
                <div class="round3-h">
                    <div class="c"></div>
                    <div class="c"></div>
                    <div class="c"></div>
                </div>
                <div class="round3-h">
                    <div class="c" style="width: 119px;height: 302px;"></div>
                </div>
                <div class="round2-w">
                    <div class="c"></div>
                    <div class="c"></div>
                </div>
            </div>
            <div class="round12-h">
                <div class="round6-w">
                    <div class="c"></div>
                    <div class="c"></div>
                    <div class="c"></div>
                    <div class="c"></div>
                    <div class="c"></div>
                    <div class="c"></div>
                </div>
                <div class="round6-w">
                    <div class="round2-h">
                        <div class="c"></div>
                        <div class="c"></div>
                    </div>
                    <div class="round4">
                        <div class="c" style="width: 285px;height: 193px;">
</div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="height166">
        <div class="area_D">
            <div class="d"></div>
            <div class="d"></div>
            <div class="d"></div>
            <div class="d"></div>
            <div class="d"></div>
            <div class="d"></div>
        </div>
        <div class="area_E">
            <div class="e"></div>
            <div class="e"></div>
        </div>
    </div>
</div>
```

到这里一个基础的页面已经完成，然后我们就要在div元素上添加光标节点的属性。先完成光标集合节点A区域，

```
<div class="main">
    <!--光标集合节点A-->
    <div class="area_A" id="A" name="fset-main_focus" x="0" y="0">
        <div class="a" name="fnode-A" x="0" y="0"></div>
        <div class="a" name="fnode-A" x="1" y="0"></div>
        <div class="a" name="fnode-A" x="2" y="0"></div>
        <div class="a" name="fnode-A" x="3" y="0"></div>
        <div class="a" name="fnode-A" x="4" y="0"></div>
        <div class="a" name="fnode-A" x="5" y="0"></div>
    </div>
    ... ...
</div>
```

可以看到，在class为"area_A"的div上增加了四个属性字段，这是**光标集合节点**必写的四个属性：

- "id"（光标集合节点的唯一标识），
- "name"（这个属性有填写格式，光标集合节点的格式："fset-"+父节点id，此处光标集合节点A的父节点为页面的根节点"main_focus"，也就是说"main_focus"为页面上最大的光标集合节点），
- "x"（x轴上的坐标值），
- "y"（y轴上的坐标值），

除此之外，光标集合节点还可以有三个可选属性：

- "status"（光标集合节点的状态，1为可用，0为不可用，不可用的光标集合节点在光标移动时将不可进入，默认为1），
- "default"（用于设定第一次进光标集合节点的默认高亮子结点，格式为'x,y'，默认为空），
- "filter"（用于将当前光标集合节点设定为不可用节点，1为不可用，0为可用，默认为0。该属性和'status'属性的区别在于，'status'为0的光标集合节点属于光标系统，'filter'为0的光标集合节点不属于光标系统）。

class为"a"的div是可以落焦的光标按钮节点，这里同样也添加了**光标节点**必写的三个属性：

- "name"（这个属性有填写格式，光标节点的格式："fnode-"+父节点id，此处光标节点的父节点id为"A"，所以得到"fnode-A"），
- "x"（x轴上的坐标值），
- "y"（y轴上的坐标值），

光标节点的其他五个可选属性有：

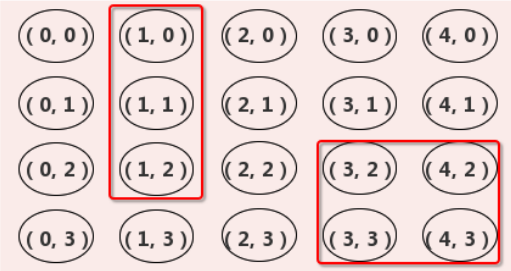
- "id"（光标节点的唯一标识，默认为空），
- "status"（光标节点的状态，1为可用，0为不可用，不可用的光标节点在光标移动时将不可落焦，默认为1），
- "url"（用于设定跳转的地址，默认为空），
- "history"（用于设定跳转时是否记住跳转历史，默认为空），
- "auth"（用于设定跳转播放时的鉴权控制，默认为空）。

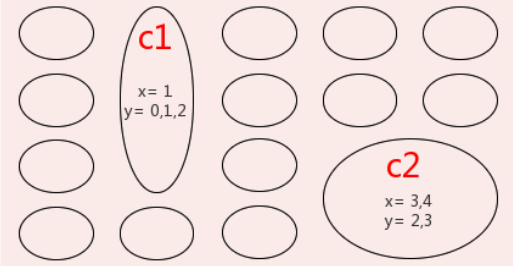
根据上述规则，再把光标集合节点B1、B2、D和E区域完善，得到如下代码：

```
<div class="main" id="main_focus">
    <!--光标集合节点A-->
    <div class="area_A" name="fset-main_focus" id="A" x="0" y="0">
        <div class="a" name="fnode-A" x="0" y="0"></div>
        <div class="a" name="fnode-A" x="1" y="0"></div>
```

```
        <div class="a" name="fnode-A" x="2" y="0"></div>
        <div class="a" name="fnode-A" x="3" y="0"></div>
        <div class="a" name="fnode-A" x="4" y="0"></div>
        <div class="a" name="fnode-A" x="5" y="0"></div>
    </div>
    <div class="height458" name="fset-main_focus" id="h485" x="0" y="1">
        <div class="area_B" name="fset-h485" id="B" x="0" y="0">
            <!--光标集合节点B1-->
            <div class="area_B1" name="fset-B" id="B1" x="0" y="0">
                <div class="b1" name="fnode-B1" x="0" y="0"></div>
                <div class="b1" name="fnode-B1" x="1" y="0"></div>
                <div class="b1" name="fnode-B1" x="2" y="0"></div>
            </div>
            <!--光标集合节点B2-->
            <div class="area_B2" name="fset-B" id="B2" x="0" y="1">
                <div class="b2" name="fnode-B2" x="0" y="0"></div>
                <div class="b2" name="fnode-B2" x="0" y="1"></div>
                <div class="b2" name="fnode-B2" x="0" y="2"></div>
                <div class="b2" name="fnode-B2" x="0" y="3"></div>
                <div class="b2" name="fnode-B2" x="0" y="4"></div>
            </div>
        </div>
        ... ...
    </div>
    <div class="height166" name="fset-main_focus" id="h166" x="0" y="2">
        <!--光标集合节点D-->
        <div class="area_D" name="fset-h166" id="D" x="0" y="0">
            <div class="d" name="fnode-D" x="0" y="0"></div>
            <div class="d" name="fnode-D" x="1" y="0"></div>
            <div class="d" name="fnode-D" x="2" y="0"></div>
            <div class="d" name="fnode-D" x="3" y="0"></div>
            <div class="d" name="fnode-D" x="4" y="0"></div>
            <div class="d" name="fnode-D" x="5" y="0"></div>
        </div>
        <!--光标集合节点E-->
        <div class="area_E" name="fset-h166" id="E" x="1" y="0">
            <div class="e" name="fnode-E" x="0" y="0"></div>
            <div class="e" name="fnode-E" x="0" y="1"></div>
        </div>
    </div>
</div>
```

最后还剩下光标集合节点C区域，区域C里面的光标节点并不能按照二维坐标系依序排列，但为什么还能将其划分成最小光标集合节点呢？先看下面的图片：





可以注意到，光标集合节点C区域可以看成是由5×4的20个有序元素组成，再将其中的个别元素合并后得到，所以我们可以将它看成是一个有规则的结构。

对于这样的光标节点，我们可以通过"x"属性和"y"属性在赋值时采用 'value1,value2,...' 的格式将横跨的光标节点进行合并。

上图光标节点c1的代码：

```
<div class="c" style="width: 119px;height: 302px;" name="fnode-C" x="1"
y="0,1,2"></div>
```

上图光标节点c2的代码：

```
<div class="c" style="width: 285px;height: 193px;" name="fnode-C" x="3,4"
y="2,3"></div>
```

光标集合节点C区域的完整代码：

```
<!--光标集合节点C-->
<div class="area_C" name="fset-h485" id="C" x="1" y="0">
    <div class="round8-h">
        <div class="round3-h">
            <div class="c" name="fnode-C" x="0" y="0"></div>
            <div class="c" name="fnode-C" x="0" y="1"></div>
            <div class="c" name="fnode-C" x="0" y="2"></div>
        </div>
        <div class="round3-h">
            <div class="c" name="fnode-C" style="width: 119px;height: 302px;"
x="1" y="0,1,2"></div>
        </div>
        <div class="round2-w">
            <div class="c" name="fnode-C" x="0" y="3"></div>
            <div class="c" name="fnode-C" x="1" y="3"></div>
        </div>
    </div>
    <div class="round12-h">
        <div class="round6-w">
            <div class="c" name="fnode-C" x="2" y="0"></div>
            <div class="c" name="fnode-C" x="3" y="0"></div>
            <div class="c" name="fnode-C" x="4" y="0"></div>
            <div class="c" name="fnode-C" x="2" y="1"></div>
            <div class="c" name="fnode-C" x="3" y="1"></div>
            <div class="c" name="fnode-C" x="4" y="1"></div>
        </div>
        <div class="round6-w">
            <div class="round2-h">
                <div class="c" name="fnode-C" x="2" y="2"></div>
                <div class="c" name="fnode-C" x="2" y="3"></div>
            </div>
            <div class="round4">
                <div class="c" name="fnode-C" style="width: 285px;height:
193px;" x="3,4" y="2,3"></div>
            </div>
        </div>
    </div>
</div>
```

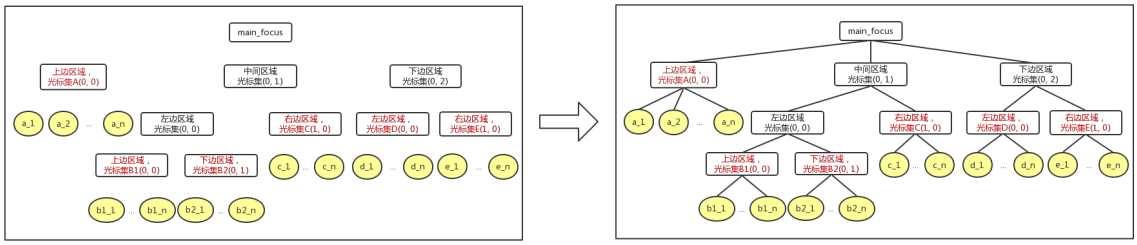
注意：在添加所需光标节点属性时，只需对相应的光标集合节点和光标按钮节点DIV添加属性即可，例如上面光标集合节点C区域里class为'round**'的元素作用只是为了更好的调整页面结构，并不是预先设计的光标节点，所以不需要添加光标节点属性。

下图是上述示例光标节点集合和按钮的name和id 字段列表：

节点类型	节点name属性	节点id属性	节点坐标
集合(页面根节点)		main_focus	
集合	fset-main_focus	A	(0, 0)
按钮	fnode-A		(0, 0)
··· 省略N个fnode-A			
集合	fset-main_focus	h485	(0, 1)
集合	fset-h485	B	(0, 0)
集合	fset-B	B1	(0, 0)
按钮	fnode-B1		(0, 0)
··· 省略N个fnode-B1			
集合	fset-B	B2	(0, 1)
按钮	fnode-B2		(0, 0)
··· 省略N个fnode-B2			
集合	fset-h485	C	(1, 0)
按钮	fnode-C		(0, 0)
··· 省略N个fnode-C			
集合	fset-main_focus	h166	(0, 2)
集合	fset-h166	D	(0, 0)
按钮	fnode-D		(0, 0)
··· 省略N个fnode-D			
集合	fset-h166	E	(1, 0)
按钮	fnode-E		(0, 0)
··· 省略N个fnode-E			

至此，页面HTML部分已经完成，接下来添加javascript脚本。

```
//【必写】引用bjf框架
<script src="bjf.all.1.0.1.js"></script>
//【必写】创建页面根节点，并将其id命名为"main_focus"
var main_focus = new FocusNode({id:'main_focus'});

//【必写】将class为"main"的DIV元素和"mian_focus"节点关联，并根据其下设定好的DIV元素创建光标系统
bjf.focus.build(bjf.$('.main'), main_focus);
```




```
//【必写】设定默认的光标节点落焦时响应事件
bjf.focus.eventDefault.on = function () {
    //【自定义】举例1.光标节点DIV的class列表添加on类
    this.data.ele.classList.add('on');
};

//【必写】设定默认的光标节点移出时的响应事件
bjf.focus.eventDefault.lost = function () {
    //【自定义】举例1.光标节点DIV的class列表删除on类
    this.data.ele.classList.remove('lost');
};

//【必写】设定默认的光标节点点击ok按钮时的响应事件
bjf.focus.eventDefault.ok = function () {
    //【自定义】举例1.光标节点DIV的class列表添加ok类
    this.data.ele.classList.add('ok');
};
```

这几行是设定光标的默认响应事件。

介绍2个相关知识点：

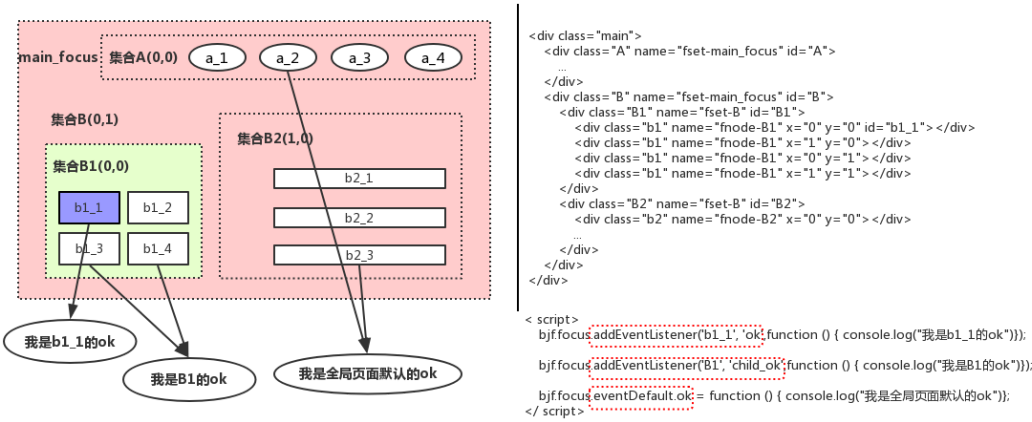
光标节点响应事件定义方法

之前有提到光标按钮节点可以响应的事件有on(光标移上落焦)、lost(光标移出)和ok(光标点击ok)，开发人员可以使用下面三个方法定义，拿ok事件举例：

- 方法一、光标按钮节点id对应响应事件名称，bjf.focus.addEventListener('光标按钮节点id','ok',function{...});
- 方法二、光标集合节点id对应加'child_'前缀的响应事件名称，表示给该光标集合节点下的所有光标按钮节点添加响应事件，bjf.focus.addEventListener('光标按钮节点的父节点id','child_ok',function{...});
- 方法三、设置全局默认光标按钮节点的响应时间，bjf.focus.eventDefault.ok = function () {...}

这三个方法的调用优先级依次为：方法一 > 方法二 > 方法三

举个例子：



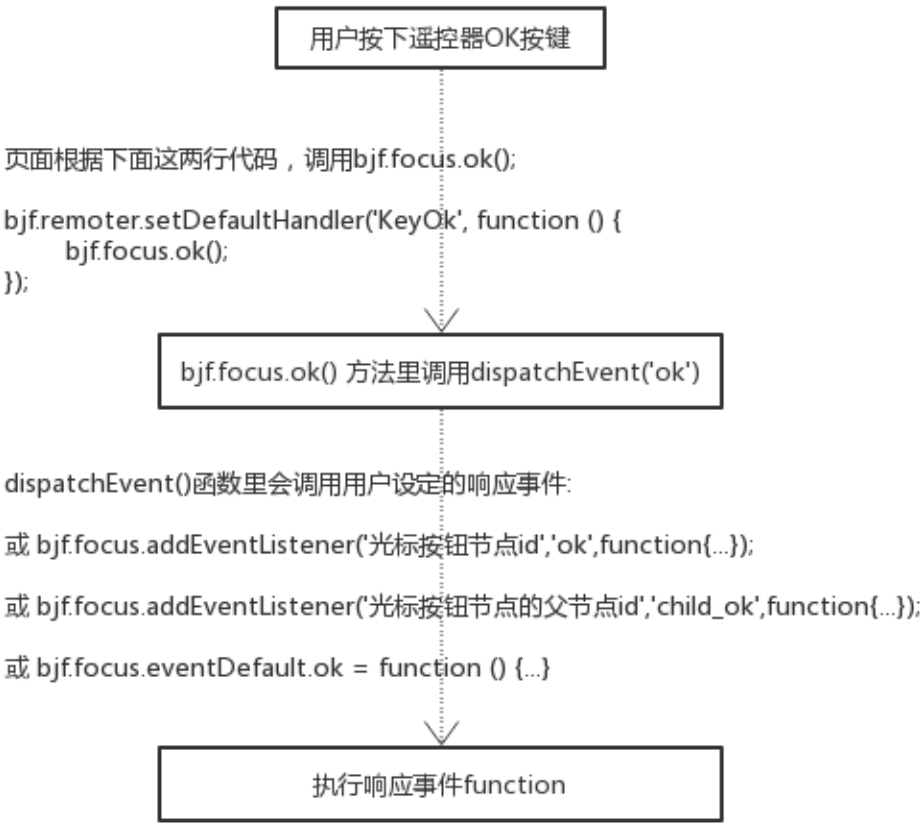
光标集合节点可以响应的事件有enter(光标移入集合)、leave(光标移出集合)、top(光标在集合上边界再向上移动时)、bottom(光标在集合下边界再向下移动时)、left(光标在集合左边界再向左移动时)和right(光标在集合右边界再向右移动时)，开发人员可以使用下面的方法定义，拿enter事件举例：

- 光标集合节点id对应响应事件名称，bjf.focus.addEventListener('光标集合节点id','enter',function{...});

关联遥控器按键

在focus模块中，定义了遥控器上下左右和ok事件默认监听函数，将遥控器按键和响应时间关联起来。

```
bjf.remoter.setDefaultHandler('KeyUp', function () {
    bjf.focus.up();
});
bjf.remoter.setDefaultHandler('KeyDown', function () {
    bjf.focus.down();
});
bjf.remoter.setDefaultHandler('KeyLeft', function () {
    bjf.focus.left();
});
bjf.remoter.setDefaultHandler('KeyRight', function () {
    bjf.focus.right();
});
bjf.remoter.setDefaultHandler('KeyOk', function () {
    bjf.focus.ok();
});
```



所以，这里有一点要注意，ok响应事件不是和遥控器的Ok按键必然关联的，开发人员也可以在方法里调用某光标按钮节点的ok方法来模拟按钮点击事件。其他光标集合节点的enter、leave、top事件等都是同个道理。

回到示例，最后添加`bjf.focus.init()`方法初始化光标，

```
//【必写】初始化光标系统，填写的参数'main_focus.getFirstChild()'表示设定的第一次落焦的光标节点
bjf.focus.init(main_focus.getFirstChild());
```

完整的javascript代码：

```
<script src="bjf.all.1.0.1.js"></script>
var main_focus = new FocusNode({id:'main_focus'});
bjf.focus.build(bjf.$('.main'), main_focus);
bjf.focus.eventDefault.on = function () {
    this.data.ele.classList.add('on');
};
bjf.focus.eventDefault.lost = function () {
    this.data.ele.classList.remove('on');
};
bjf.focus.eventDefault.ok = function () {
    this.data.ele.classList.add('ok');
};
bjf.focus.init(main_focus.getFirstChild());
```

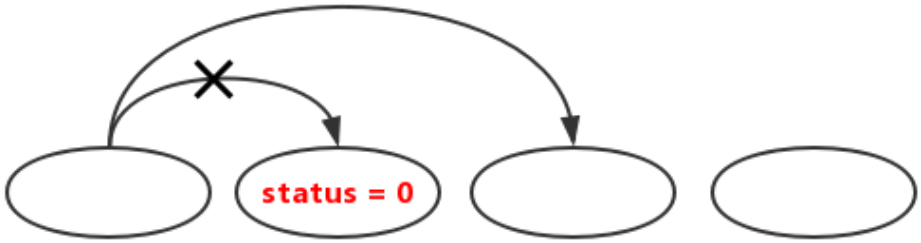
这样，一个基础的带光标系统的页面就完成了。

四、FocusNode属性和方法列表

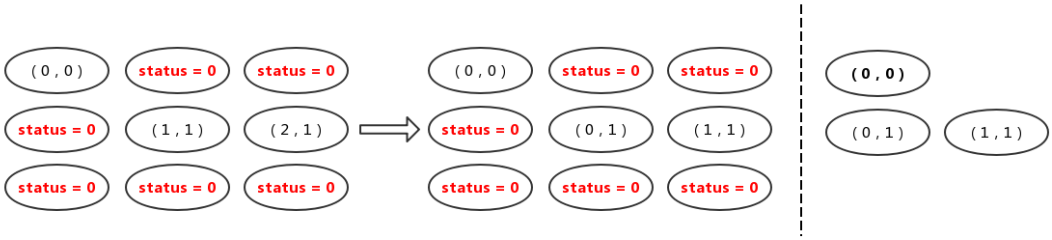
上面已经看过FocusNode的结构，我这里再结合使用场景和设置获取方法着重说明下几个比较常用的属性。

- FocusNode.status 这个属性是在DIV节点创建的时候定义的，表示光标状态是否可用。相关的方法有，

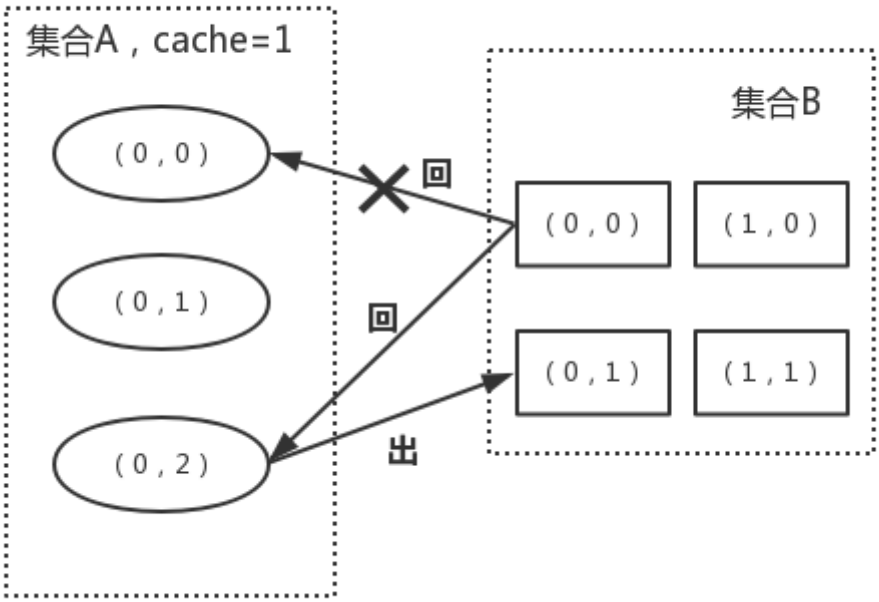
    1. getPosterity()，这个方法功能是根据偏移量来获取当前节点的兄弟结点，如果兄弟光标按钮节点节点的status为不可用，就会获得下一个兄弟光标按钮节点，



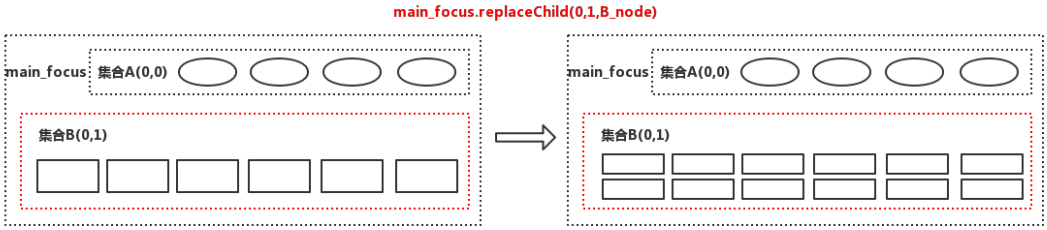
2.filterDisabled()，这个方法功能是过滤当前光标集下所有不可用的子结点，需要开发人员调用。看下面这个例子，



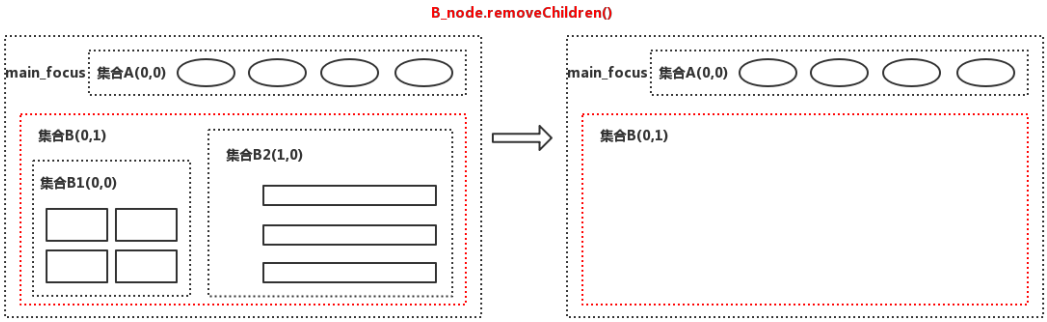
- FocusNode.cache 这个属性是在DIV节点创建的时候定义的(只对光标集合节点有用)，表示光标跳出该集合时，是否记住光标所在子结点的ID。



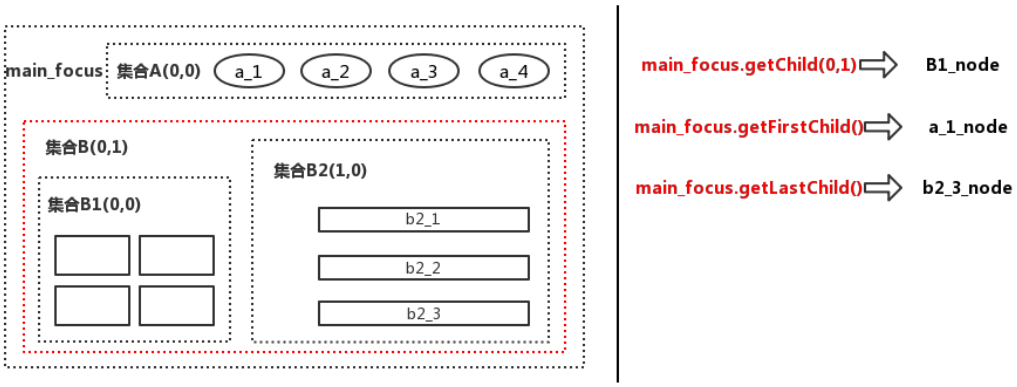
- FocusNode.replaceChild(x, y, node) 这个方法功能是替换指定位置的子结点，该位置原来的结点会和parent解除父子关系。



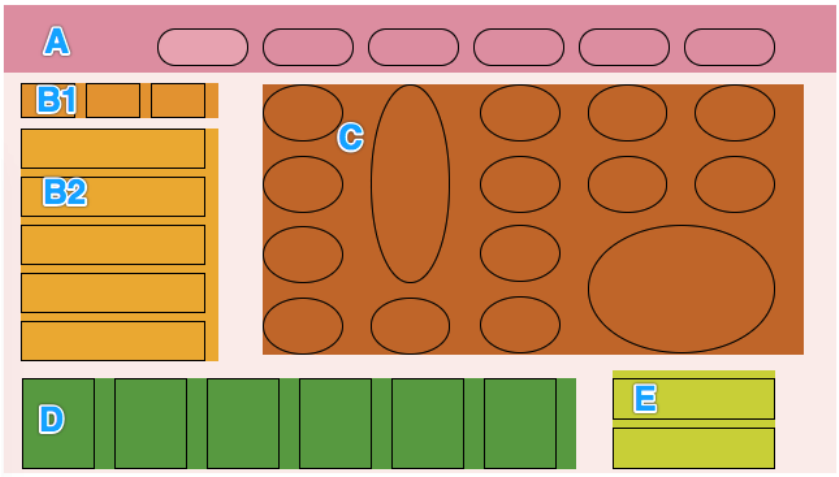
- FocusNode.removeChildren() 这个方法功能是移除所有子结点，并断开父子关系。

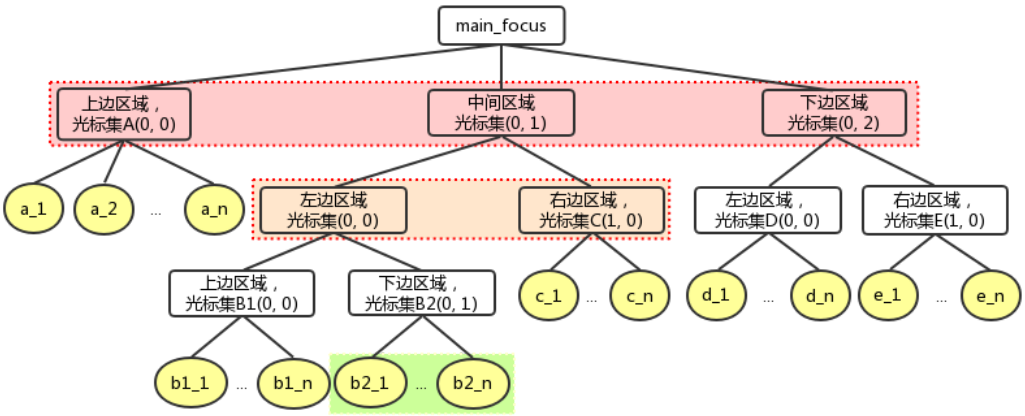


- FocusNode.getChild(x, y)；FocusNode.getFirstChild()；FocusNode.getLastChild() 三个方法都是获取子节点，但是有所区别



- FocusNode.getAbove()；FocusNode.getUnder()；FocusNode.getLeft()；FocusNode.getRight() 这四个方法分别是获取上下左右兄弟节点





五、bjf.focus对象属性和方法列表

- bjf.focus.locked 定义光标是否锁定，锁定后光标不能移动

- bjf.focus.node 表示当前亮起的光标按钮节点

- bjf.focus.getHistory() 在我们要离开当前页面的时候，有时候需要记住光标的位置，方便在回到该页面的时候，光标依然停留在原先的位置。 此时只需要调用bjf.focus.getHistory()，就可以把当前的光标位置以字符串的形式拼接出来。格式如下： "x-y:parent.x-parent.y....top.x-top.y" 如："0-2:0-0:0-1:1-1:0-0"

- bjf.focus.init(); bjf.focus.initByhistory() 这两个方法都是初始化页面光标的方法，不同的是initByhistory()是根据历史记录初始化页面。

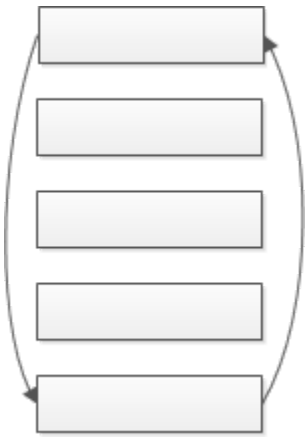
```
//离开时记录页面光标位置
bjf.storage.pageSet('focus-history', bjf.focus.getHistory());

//回来时按记录恢复
var region = bjf.storage.pageGet('focus-history');
function init() {
    if (region.length > 1) {
        bjf.focus.initByhistory(main_focus, region);
    }
    else {
        bjf.focus.init(menu_focus.getFirstChild());
    }
}
```

- bjf.focus.change() 替换亮起的光标节点

- bjf.focus.up(); bjf.focus.down(); bjf.focus.left(); bjf.focus.right() 上下左右移动光标

六、几种特殊情况举例

1、循环移动



如图所示，有些看吧有光标首尾循环的要求，这样的话，之前在上面提到，如果光标已经到达第一条时，再往上FocusNode会认为到达上边界。此时会触发一个onTopBorder的事件(同理到达下边界时，也会触发onBottomBorder事件，左右亦如此)。

看吧逻辑可使用这两个事件的回调函数来实现首尾相连。

例如：

```
<div class="c" id="west_focus" name="fnode-C" x="0" y="1"></div>

bjf.focus.addEventListener('west_focus', 'top', function () {
    Focus.change(this.getChild(0, this.children.length - 1));
    return false;
});

bjf.focus.addEventListener('west_focus', 'bottom', function () {
    Focus.change(this.getChild(0, 0));
    return false;
});
```

以上代码，在光标达到此区域上边界时，强制将Focus.node变换到了最下面一个。在下边界时，变换到了第一个。

此处要注意回调函数的返回值，如果看吧在回调函数里，干预了Focus.node，必须要return false 否则FocusNode还会执行默认逻辑(比如此处，光标会跳出，干预失效)

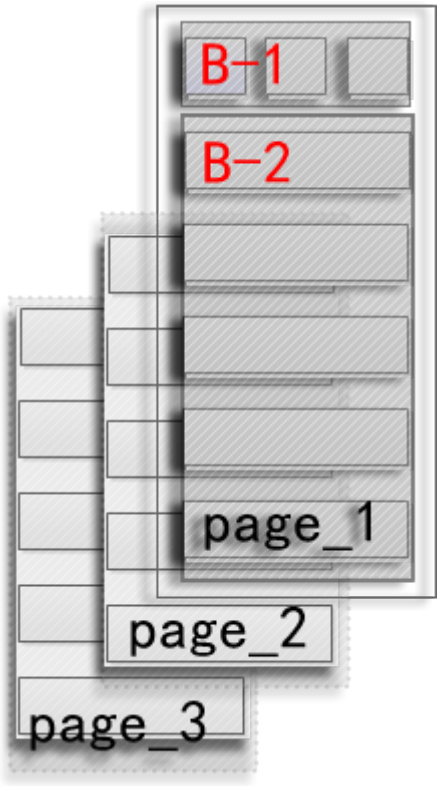
2、分页

分页在看吧里很常见。所以在设计FocusNode时，也充分考虑了分页需求。

比如，我们现在把B区，分解成为B-1和B-2两个子区域。其中B-2的列表是有分页需求的。

那么代码可以这么写

```
<div class="area_B" name="fset-h485" id="B" x="0" y="0">
    <div class="area_B1" name="fset-B" id="B1" x="0" y="0">
        <div class="b1" name="fnode-B1" x="0" y="0">b1-1</div>
        <div class="b1" name="fnode-B1" x="1" y="0">b1-2</div>
        <div class="b1" name="fnode-B1" x="2" y="0">b1-3</div>
    </div>
    <div class="area_B2" name="fset-B" id="B2" x="0" y="1">
        <div class="b2" name="fnode-B2" x="0" y="0">b2-1</div>
        <div class="b2" name="fnode-B2" x="0" y="1">b2-2</div>
        <div class="b2" name="fnode-B2" x="0" y="2">b2-3</div>
        <div class="b2" name="fnode-B2" x="0" y="3">b2-4</div>
        <div class="b2" name="fnode-B2" x="0" y="4">b2-5</div>
    </div>
</div>
```



如果此时，触发了分页逻辑(用户点击了遥控器下一页，或者光标移动到了最下面一条再往下)

那么我们再新建一个光标节点，结构和b_area_2一样的。

```
<div class="area_B2" name="fset-B" id="B2-1" x="0" y="1">
    <div class="b2" name="fnode-B2" x="0" y="0">b2-1</div>
    <div class="b2" name="fnode-B2" x="0" y="1">b2-2</div>
    <div class="b2" name="fnode-B2" x="0" y="2">b2-3</div>
    <div class="b2" name="fnode-B2" x="0" y="3">b2-4</div>
    <div class="b2" name="fnode-B2" x="0" y="4">b2-5</div>
</div>
```

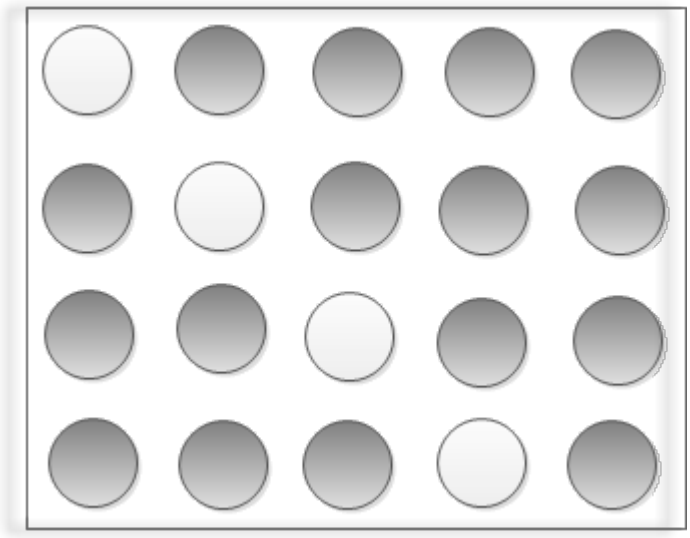
然后在执行一下语句：

```
middle_left.replaceChild(0, 1, B2-1);
```

那么在middle_left的坐标系x=0,y=1的位置，b_area_2_page_2替换掉了原先的b_area_2，b_area_2也和middle_left断绝了父子关系，即完成了分页。

3、可用与不可用二维混合排列

我们先看一张图：



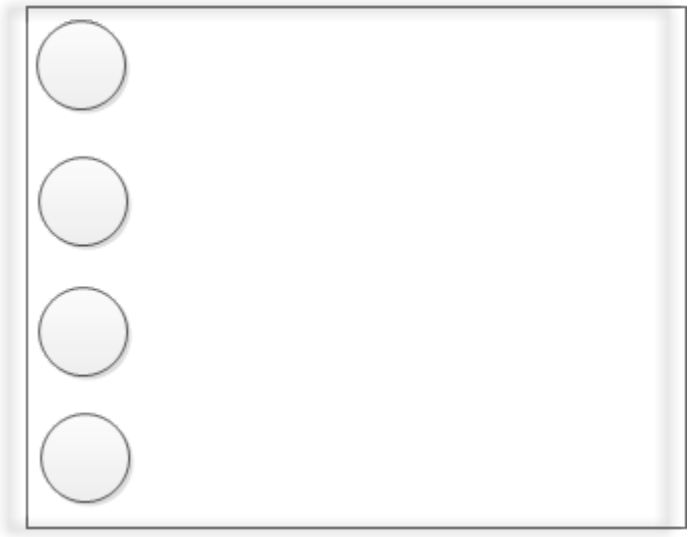
假设现在C区域内，暗色的按钮都不可点击，那么单光标进入到第一个按钮时，该怎么走啊，正常的逻辑，似乎永远也走不到二排二列那位兄弟上了(位移传递也到不了)。

其实在我做高清NBA的时候就碰到了这种情况(产品一般不会考虑这种极端情况，但是悲催的开发就得要考虑)。

现在普通的二维坐标系已经无法满足我们的要求了，于是我在FocusNode下面新增了一个方法：filterDisabled()。把不可用的干掉，用"可用按钮"重建新的坐标系。

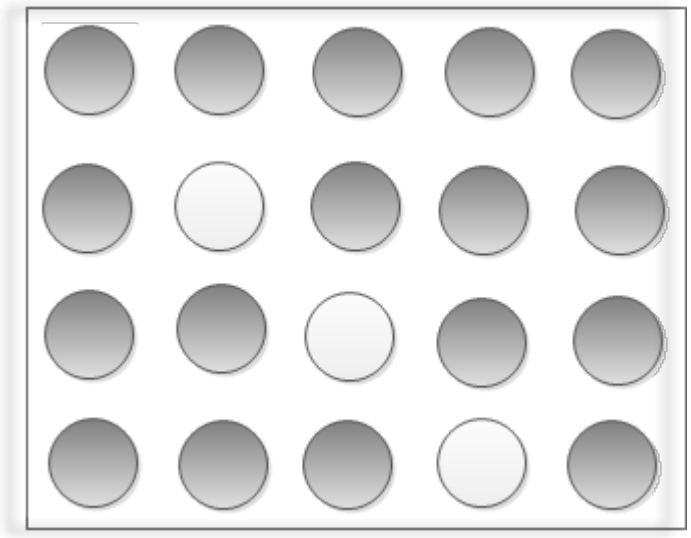
如果大家在做看吧时，碰到了可能会出现这类情况的时候，就需要在C区节点的子结点添加好后，执行一下此方法：middle_right.filterDisabled()

此方法会把可用的节点保存在FocusNode.alive_children里面，形成新的坐标系：如图所示:



FocusNode内部的位移方法，会在普通坐标系查找失败、alive_children存在的情况下，到健在的子结点里面去查找。

还有一种更变态的情况，如图所示：



C区的第一个子元素就是不可用的，按照正常逻辑，根本就跳不进来。但是有了alive_children之后，会通过新坐标系(启动异次元搜索)跳进来。

---

demo演示

http://10.61.13.147/vistools/bjf/focus/demo_creatSimplefocus.html