前端开发理论热点面对面

从怎么看，到怎么做?

kejun

1. 前端开发约等于客户端开发
   - 桌面、手机、浏览器、其他console

2. 完全受限于客户端运行时环境
   -渲染引擎、JS引擎、设备API

3. 很深的标准诉求
   - 历史问题: Microsoft vs. W3C
   - W3C: HTML 4.01, CSS 2.1, DOM Level 2, ...
   - ECMA-262 Edition 5, javascript 1.5, 1.6, ...

4. 独特的文化
   - 兼容性：CSSHack，Fallback
   - 安全性：Hack同源策略，XSS/CSRF

5. 三种语言(HTML/CSS/Javascript)协同开发

6. 不同的开发思维（很分裂）

HTML　没逻辑，没表现， 纯结构，所以有模式可循
CSS　　没逻辑， 全表现，谙熟显示原理，层叠和继承
JS　　强逻辑， 离不开DOM

7. 亦繁亦简，亦正亦邪

- SPA(Single-Page-Application), WebApp
- 效果、验证、请求
- XSS

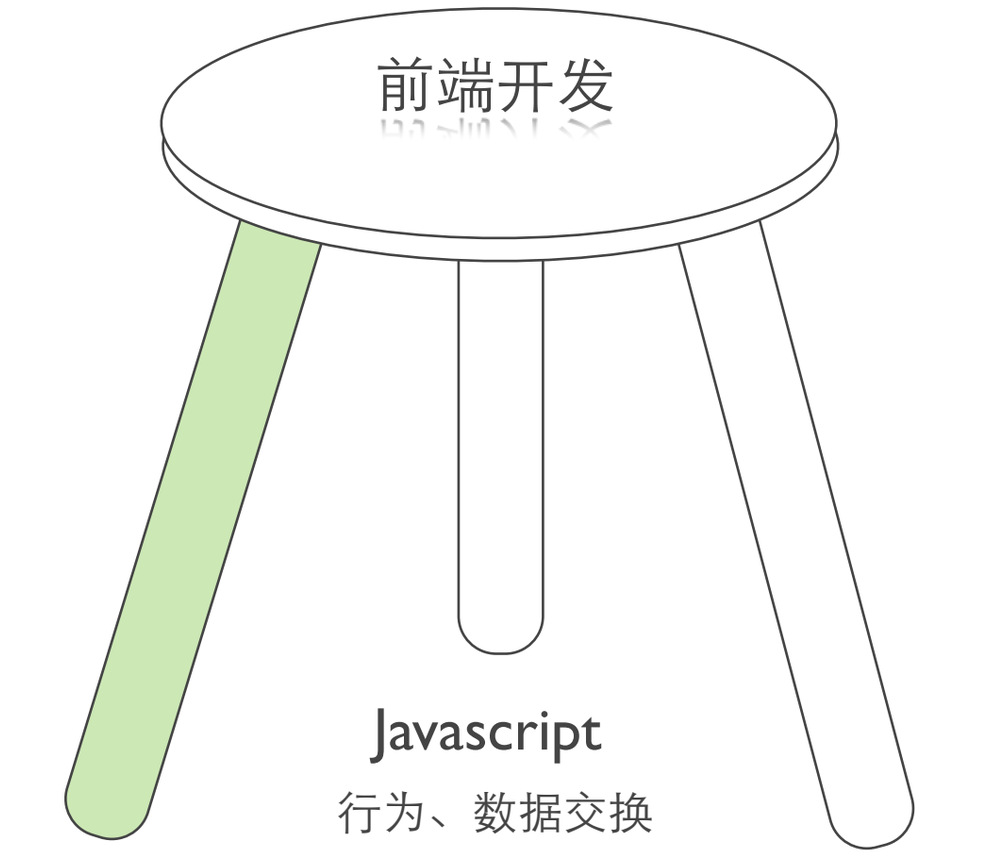
8. 用户敏感

- 性能
- 可用性

9. 分级支持 / 向前兼容 / 向后退化

10. 不断学习

1. HTML设计思想
2. CSS基本原理
3. Javascript开发方式
4. jQuery最佳实践
5. 开发规范


前端开发
Javascript
行为、数据交换
HTML
信息结构
CSS
显示效果

1. 关注结构设计

2. 结构设计中的概念：元素、结构、容器、模块、语义化

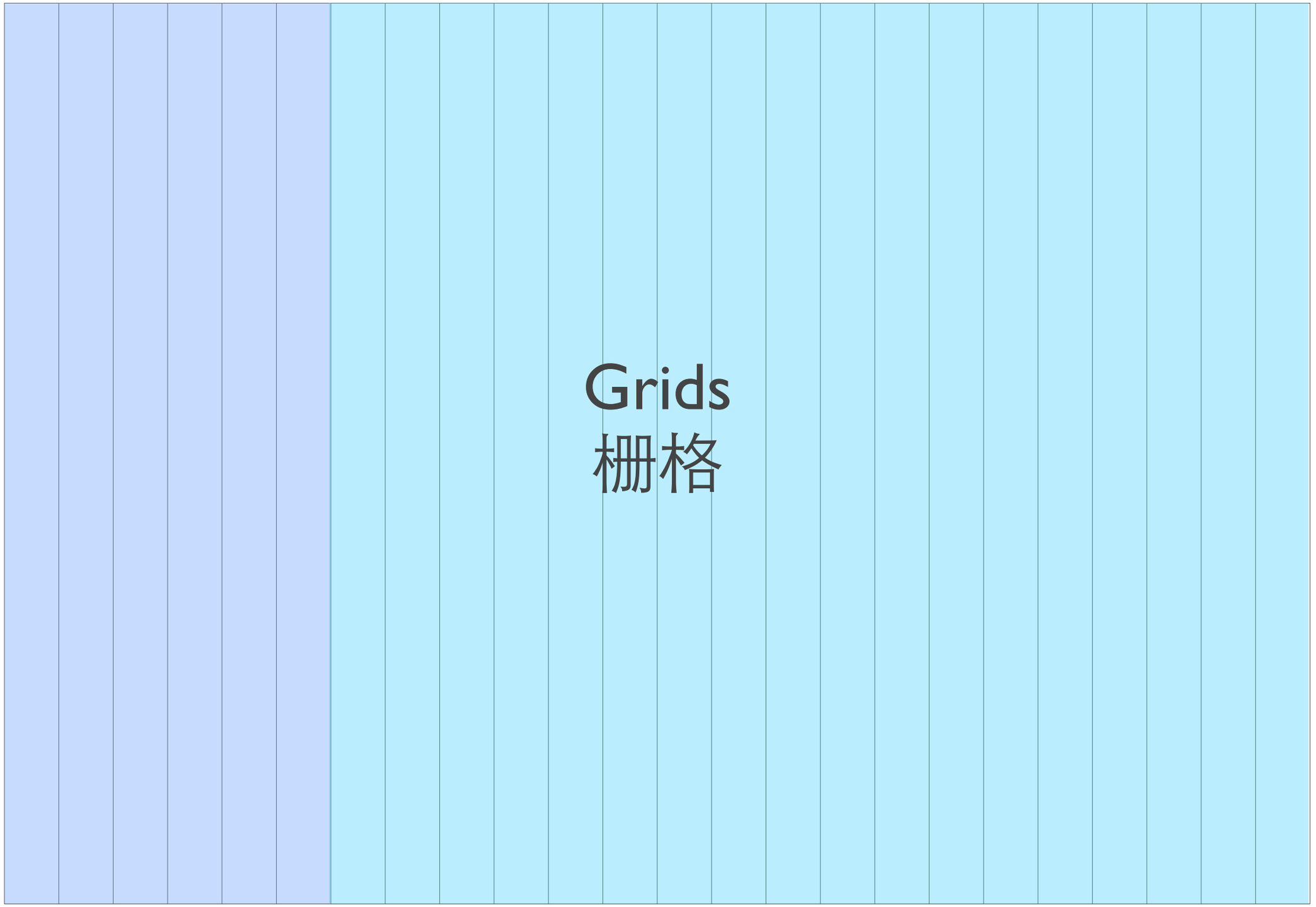
3. 设计步骤（或逆向）
   step1: 页面布局 (固定尺寸、栅格原理)
   step2: 模块化装配
   step3: 模块结构
   step4: 语义化内容

100%

950px

590px

310px



http://developer.yahoo.com/yui/3/cssgrids/

http://www.douban.com/css/grids.css

装配模块

有8个活动邀请你参加，需要你处理
有2个线上活动邀请你参加，需要你处理
有20个人关注你
你有8个提醒

```
有 <a href="/request/#event">8个活动</a>邀请你参加，需要你处理
有 <a href="/request/#online">2个线上活动</a>邀请你参加，需要你处理
有 <a href="/contacts/rlist">20个人关注你</a>
你有<a href="/notification/">8个提醒</a>
```

img, a, object, form, input, button ...

```
<ul>
<li>有 <a href="/request/#event">8个活动</a>邀请你参加，需要你处理</li>
<li>有 <a href="/request/#online">2个线上活动</a>邀请你参加，需要你处理</li>
<li>有 <a href="/contacts/rlist">20个人关注你</a></li>
<li>你有<a href="/notification/">8个提醒</a></li>
</ul>
```

ul-li, ol-li, dl-dt-dd, table, p, blockquote...

标明这是一个叫 ”Infobox” 的对象

```
<div class=”infobox”>
<ul>
<li>有 <a href="/request/#event">8个活动</a>邀请你参加，需要你处理</li>
<li>有 <a href="/request/#online">2个线上活动</a>邀请你参加，需要你处理</li>
<li>有 <a href="/contacts/rlist">20个人关注你</a></li>
<li>你有<a href="/notification/">8个提醒</a></li>
</ul>
</div>
```

div, span...

有 8个活动邀请你参加，需要你处理

有 2个线上活动邀请你参加，需要你处理

有 20个人关注你

你有8个提醒

有 8个活动邀请你参加，需要你处理

有 2个线上活动邀请你参加，需要你处理

有 20个人关注你

你有8个提醒

有 8个活动邀请你参加，需要你处理

有 2个线上活动邀请你参加，需要你处理

有 20个人关注你

你有8个提醒

有 8个活动邀请你参加，需要你处理

有 2个线上活动邀请你参加，需要你处理

有 20个人关注你

你有8个提醒

标签、CLASS&ID命名、通用模式

1. 标签表示一个元素
2. 按性质划分：Block-Level和Inline-Level
3. 按语义划分：

- 通用文档构成 h1-h6, p, em, strong, abbr
- 引用和参考 blockquote, q, cite
- 联系方式 address
- 表单 form, input, button, label, fieldset, legend, select(optgroup, option), textarea
- 程序 code, kbd, pre, samp, var
- 表格类数据 table(tr, td, tbody, th, col, colgroup, ...)
- 定义或解释 dfn, dl(dt, dd)
- 媒体 img, object, embed
- 列表 ol(li), ul(li)
- 通用容器 div, span

4. 不推荐使用的标签：

big, hr, small, tt, font, center, dir

5. 不标准的标签：

blink, marguee

http://hikejun.com/work/qq/spec/html_reference.html

HTML中“类”和“对象”的概念

```
<div id=”db-tribe-members” class=”mod”>...</div>
<ul class=”list member-list”>...</div>
```

类： div.mod, ul.list, ul.member-list
对象： div#db-tribe-members

1. 结构化的东西都有“模式”（模式：解决某一类问题的通用方法）

2. 抽象信息对象
   页面布局、列表、图文混排，.....

3. Y!经典的模块结构

```
<div id="db-xxx" class="mod">
    <div class="hd">
        标题
    </div>
    <div class="bd">
        内容
    </div>
    <div class="ft">
        更多
    </div>
</div>
```

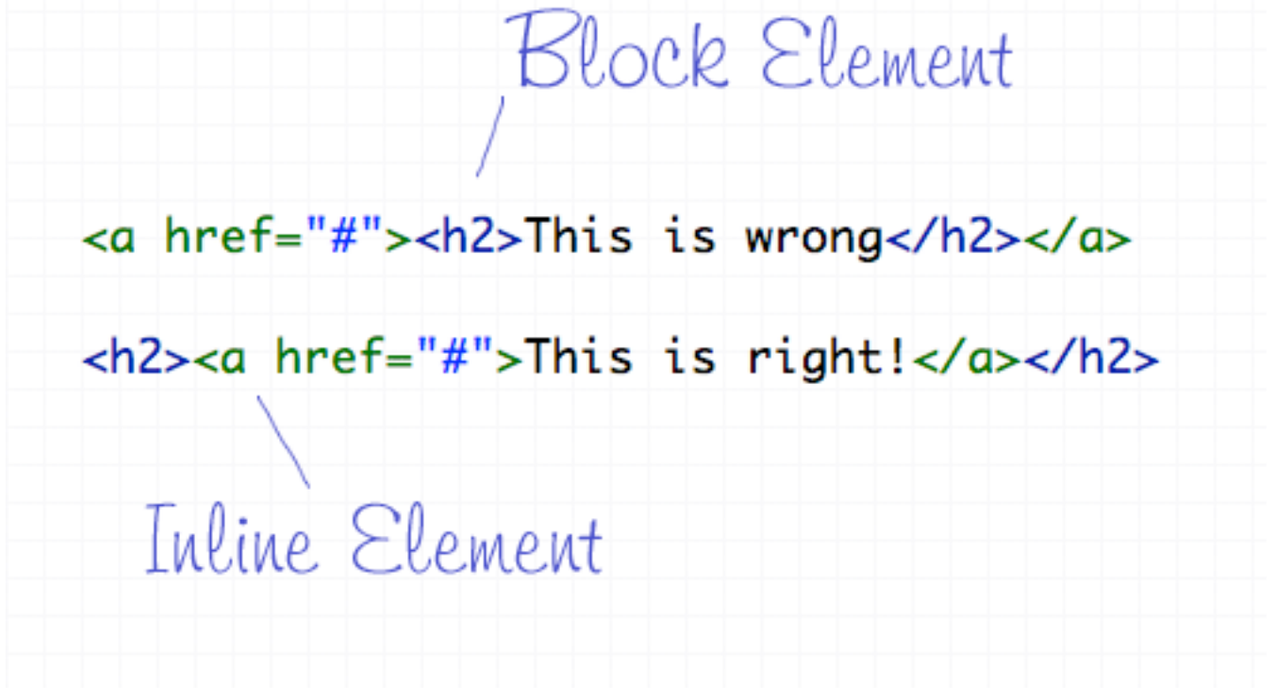
```
<!DOCTYPE html>
<html lang="zh-CN" class="${client_class(request)}">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

    <title>${self.title(True)}</title>
       ${self.doc_head()}
 </head>
<body>
    ${self.top_navigation()}
    <div id="wrapper">
        ${self.header()}
        ${self.mbody()}
        ${self.footer()}
    </div>
    ${self.bottom_script()}
    ${self.stat()}
</body>
</html>
```

base模板

避免易犯错误

http://line25.com/articles/10-html-tag-crimes-you-really-shouldnt-commit

错误的嵌套会引发怪异的渲染问题



HTML 4 strict/XHTML 1.0 strict嵌套规则

block嵌套block或inline
a嵌套inline, 不能嵌套a
label不能嵌套label
button不能嵌套a, input, select, textarea, label, button, form, fieldset
form不能嵌套form
pre嵌套inline, 不能嵌套img, object, sup, sub
address嵌套inline
object嵌套block或inline
img, br, input, hr为空标签
必须组合使用

- ol-li, ul-li, dl-dt+dd
- table-caption+colgroup-col+col+thead+tbody+tfoot-tr-td+th
- select-option+optgroup-option
- object-param

提高可访问性：替换文字

No ALT attribute

```
<img src="panthro.jpg" />

<img src="panthro.jpg" alt="Panthro from the Thundercats" />
```

考虑语义和扩展性

```
<p>
    Lion-O<br />
    Cheetara<br />
    Panthro<br />
    Tygra<br />
    Mumm-Ra<br />
</p>
```

Use a List

,

```
<ul>
    <li>Lion-O</li>
    <li>Cheetara</li>
    <li>Panthro</li>
    <li>Tygra</li>
    <li>Mumm-Ra</li>
</ul>
```

标签选择从语义出发而非样式

```
<b>This is bold</b>,
and <strong>so is this (but better!)</strong>
```

所有样式由CSS控制而非HTML

```
Feel the magic, hear the roar
<br />
<br />
Thundercats are loose
```

Don't use multiple
 tags

```
<p>Feel the magic, hear the roar</p>

<p>Thundercats are loose</p>
```

同样用 控制缩进和间距也是同样的问题

使用语义化标签

<s> and <strike> are now deprecated

```
<strike>Lion-O</strike> Panthro is the coolest Thundercat.
```

<del> and <ins> are the new tags on the block

```
<del>Lion-O</del> <ins>Panthro</ins> is the coolest Thundercat.
```

所有样式由CSS控制，style属性打破优先级

所有样式由CSS控制，不要用改变样式的属性

```
<img src="#" alt="" border="0" />
```

Put it in the stylesheet

正确用H1-H6结构化文档

```
<p><strong>Thundercats</strong></p>
<p>Thundercats are awesome</p>
```

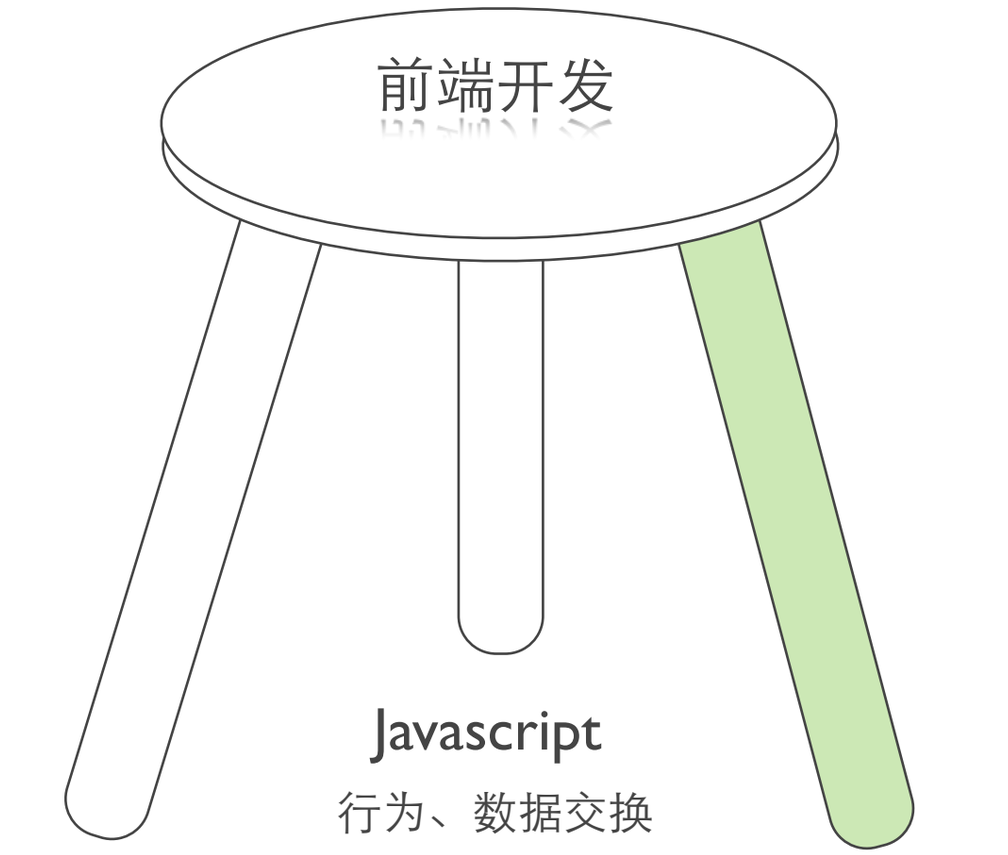
Use Headers where appropriate

```
<h3>Thundercats</h3>
<p>Thundercats are awesome</p>
```

不要用非标准和不推荐使用的标签

用样的还有marguee

名词区分：DHTML / XHTML / HTML / HTML5


前端开发
Javascript
行为、数据交换
HTML
信息结构
CSS
显示效果

1. CSS - Cascading Style Sheets，千变万化的CSS http://www.mezzoblue.com/zengarden/alldesigns/
2. CSS - 实现排版和视觉效果
3. CSS 3 - 更灵活的布局、更丰富的视觉效果和动态效果
4. CSS描述界面是如何呈现的
5. 独特的CSSHack

```
/* 语法超简单，越简单越灵活 */
selector { property: value; }

/* CSS的At-rules用法 */
@import url(xxx.css);
@media print { ... }
@charset  “UTF-8”;
@font-face { ... }
```

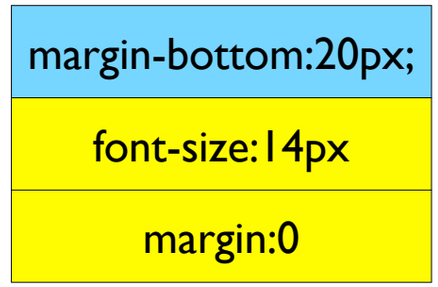
解析顺序： selector部分从右到左，声明部分从左到右

CSS必搞清楚的概念：

1. 层叠和继承
2. 优先级
3. 盒模型
4. 格式化
5. 定位

```
<div class="content note-content">
    <p>正文</p>
</div>
```

```
.content p { margin:0;font-size:14px; }
.note-content  p { margin:0 0 20px 0; }
```

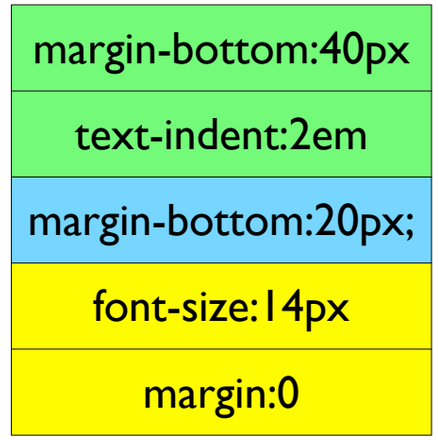



```
<div class="content comment-content">
    <p>正文</p>
</div>
```

```
.comment-content  p { font-size:12px;color:#666; }
```

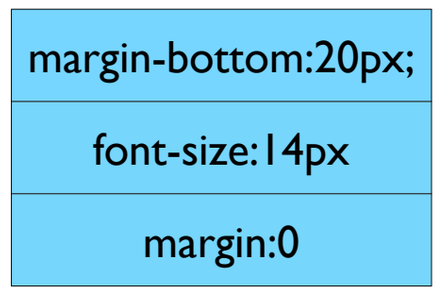

```
<div class="content note-content note-full">
    <p>正文</p>
</div>
```

```
.note-full  p { text-indent:2em;margin-bottom:40px;  }
```




```
<div class="note-content">
    <p>正文</p>
</div>
```

```
.note-content  p,
.content p { margin:0;font-size:14px; }
.note-content  p { margin:0 0 20px 0; }
```



特定样式

局部通用样式

产品全局样式

Reset默认样式

CSS开发全部继承浏览器的默认样式

http://www.iecss.com/

```
<div id=”db-site-note” class=”content note-content”>
    <p>正文</p>
</div>
```

```
.content p { color:blue; }
.content { color:red; }
.note-content p { color:yellow; }
```

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
0	0	0	1	1

```
div.content p { color:#000; }
```

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
0	0	0	1	0

```
#db-site-note p { color:#999; }
```

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
0	0	0	1	2

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
0	0	1	0	1

```
<div id="db-site-note" class="content note-content" style="color:green">
    <p>正文</p>
</div>
```

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
0	1	0	0	0

```
p { color:#000 !important; }
```

!important	Style属性	#ID	.Class/ 伪类/ 属性选择器	标签
1	0	0	0	0

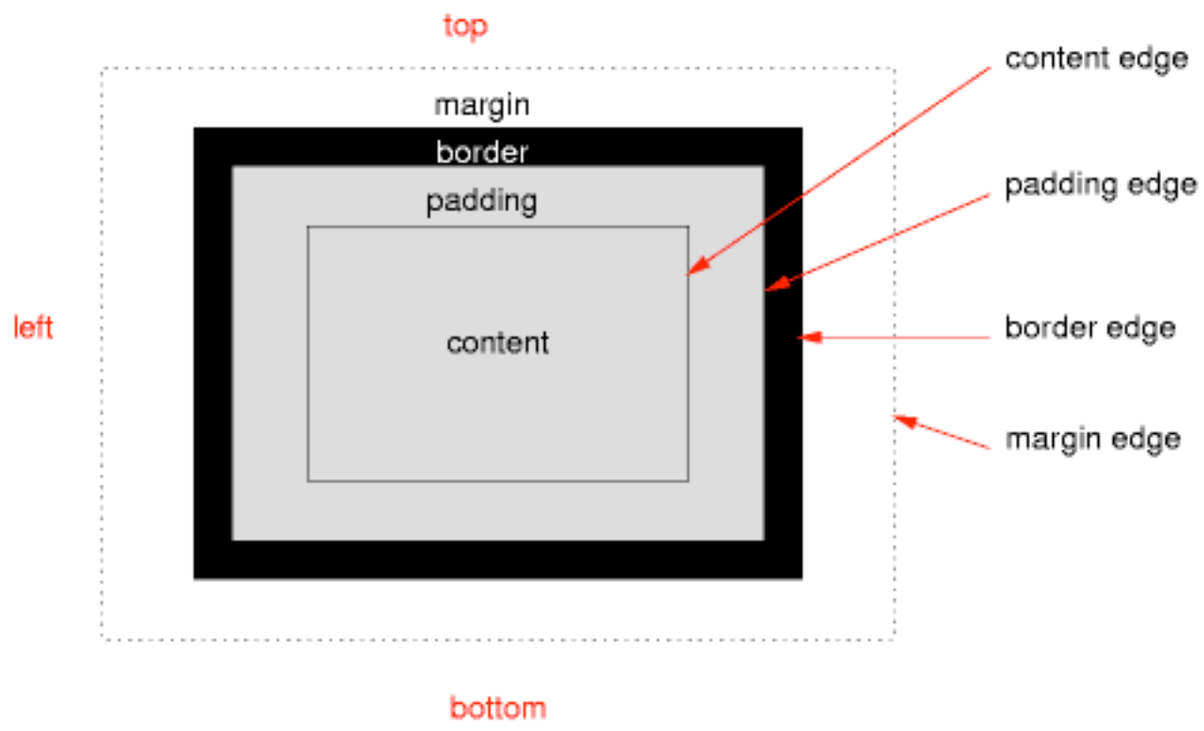
优先级关系被打乱，慎用！！！

1. 简单说一个box对应一个HTML元素

2. box的性质由display属性指定
   inline - inline box
   block - block box
   inline-block - box内部按block box格式化，外部按inline box
   none - box及内部的box均不可见
   .....（CSS3）

3. 基本的box模型描述的就是block-level box在文档流中的布局方式

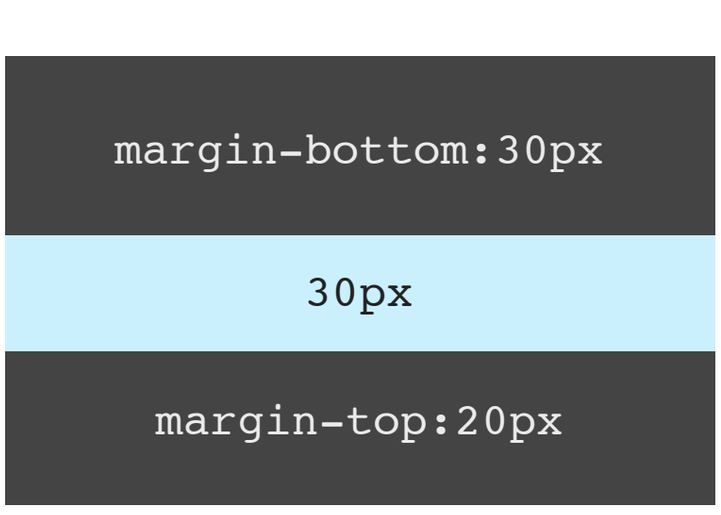
基本的box模型

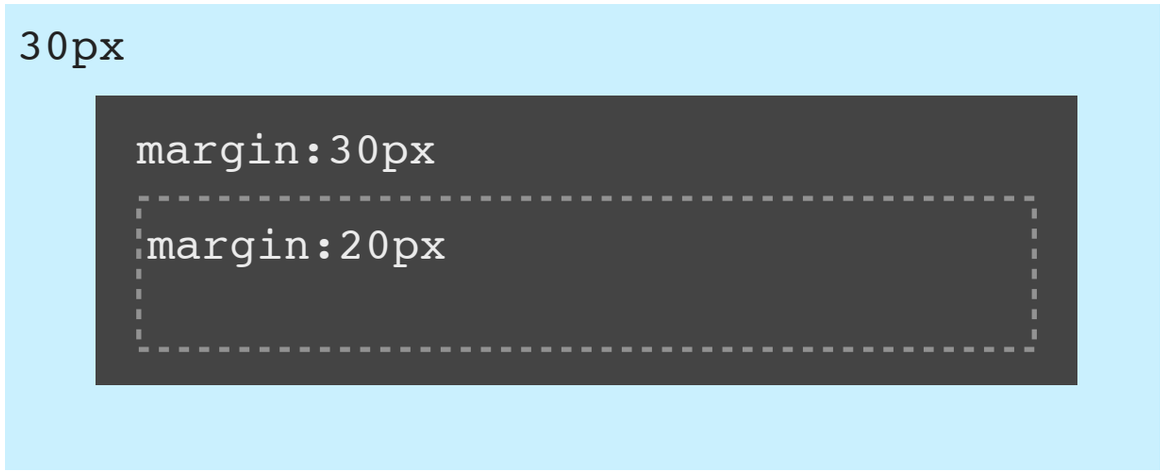


小心：width/height和padding/margin/border同时用

block格式化

1. 在一个block格式化上下文中(block formatting context)，box是垂直布局的
2. 垂直两个box的间距通过margin控制
3. collapsing margins问题
   3-1. block box和block box之间
   3-2. block box嵌套block box






```
<div style="margin-bottom:30px;"></div>
<div style="margin-top:20px;"></div>
```

```
<div style="margin-bottom:30px;">
   <div style="margin-top:20px;"></div>
</div>
```

inline格式化

1. 在一个inline格式化上下文中(inline formatting context)，box是水平布局的
2. 水平的margin、padding、border有效，垂直无效
3. 不能指定宽高

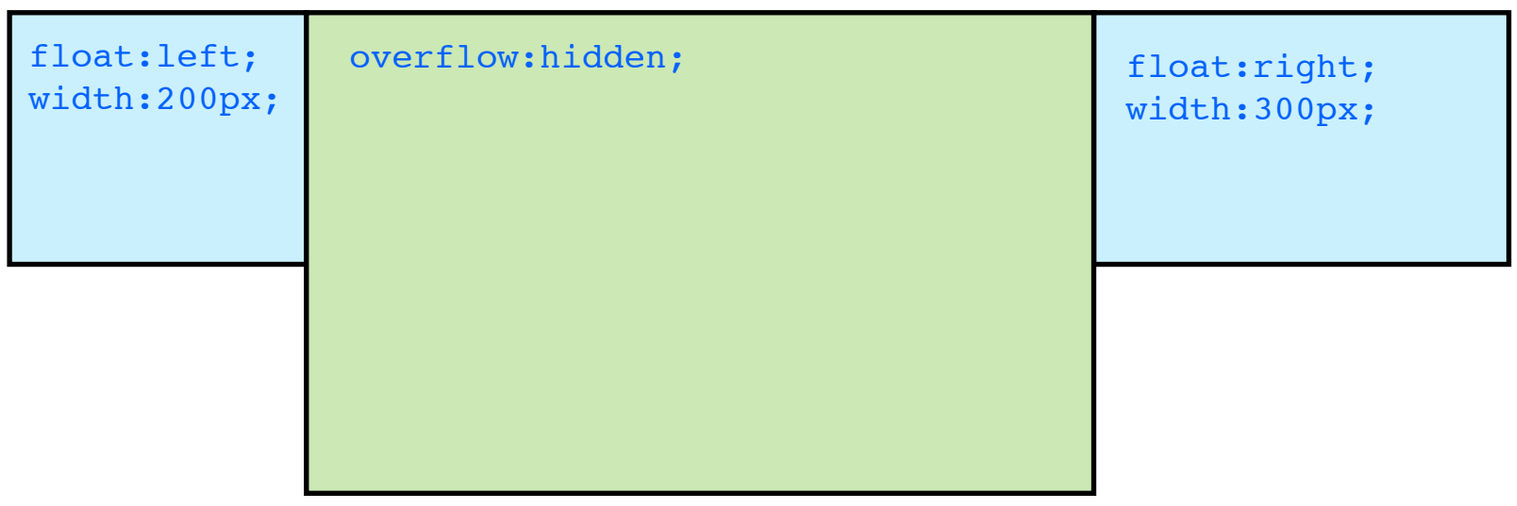
block格式化上下文 (block formatting context)

满足下面条件就会形成一个block格式化上下文

1. float不为none
2. overflow不为visible
3. display设为‘table-cell’, ‘table-caption’, 或‘inline-block’
4. position既不是static也不是relative
5. IE的hasLayout特性会建立一个新的block formatting context

block formatting context的特性：

1. 不存在collapsing margins问题
2. 边缘不会和float box重叠。常用来清浮动和布局



IE的hasLayout特性

http://www.satzansatz.de/cssd/onhavinglayout.html

1. 最大用处是解决IE的渲染bug

2. 下面的属性可以触发hasLayout:
   position: absolute
   float: left|right
   display: inline-block (IE6下跟标准不同)
   width: value (非auto)
   height: value (非auto)
   zoom: value (非normal)
   writing-mode: tb-rl

   IE 7下:
   overflow: auto|hidden|scroll
   overflow-x|-y: auto|hidden|scroll
   position: fixed
   min-|max-width: value
   min-|max-height: value

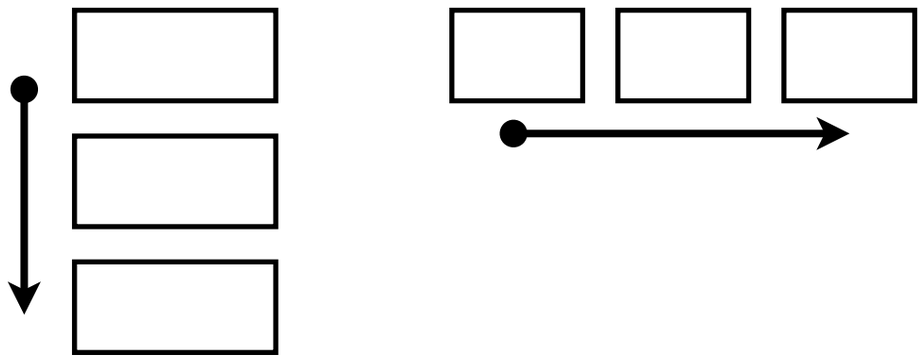
注：利 width/height 触发inline-level box的hasLayout仅在quirks mode有效。
通常用 zoom:1

CSS2.1规定了3种定位方案：

1. Normal flow
2. Floats
3. Absolute position

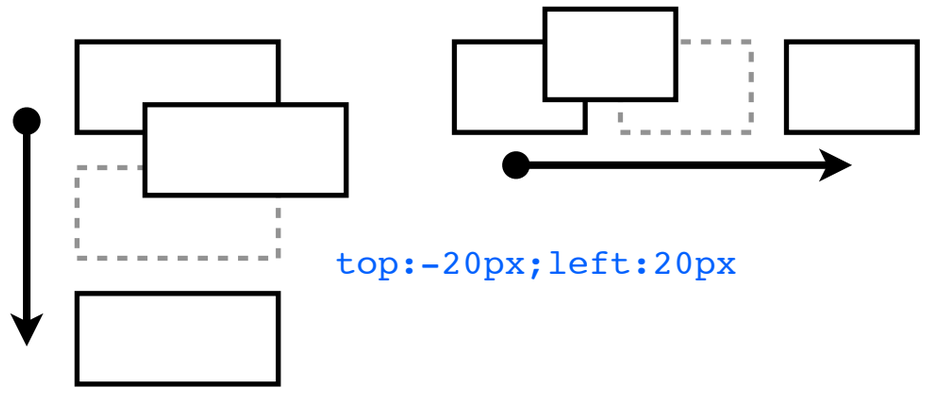
normal flow - 普通流

position:static(默认) - 包含block格式化、inline格式化，top/left/right/bottom, z-index无效
position:relative - 符合normal flow布局，top/left/right/bottom, z-index有效
仍然在原有的格式化上下文中，同时不会改变box的性质



position:static

top:-10px;left:-20px



position:relative

Floats - 浮动

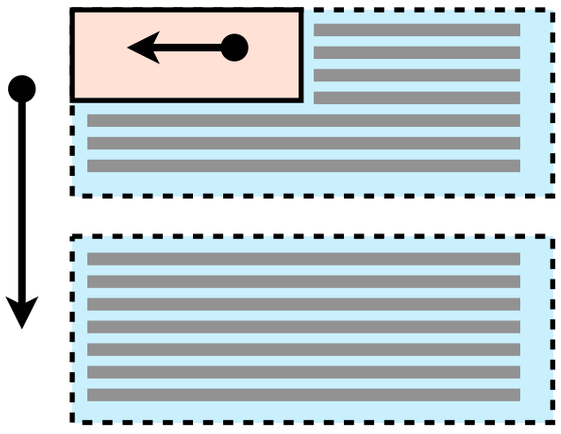
1. 脱离当前的文档流，变换到容器的边缘，或是另一个浮动box的边缘，若空间不够再向下移动直到可以放下

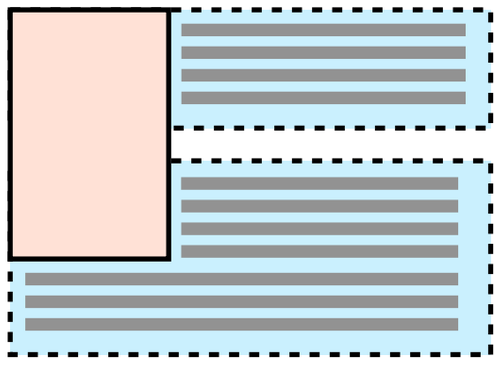
2. 文档流中的line box, inline box将围绕float box

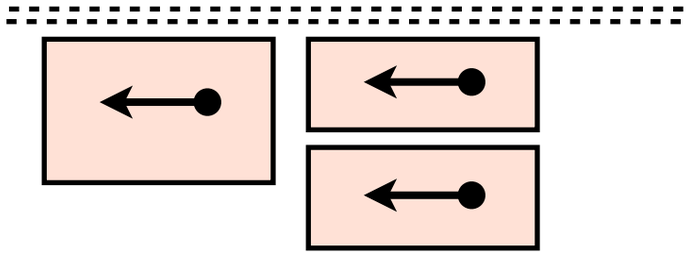
3. float box若不设宽度将是它实际内容的宽度

4. 把box都变成block-level box，形成一个新的block formatting context

5. top/left/right/bottom, z-index无效

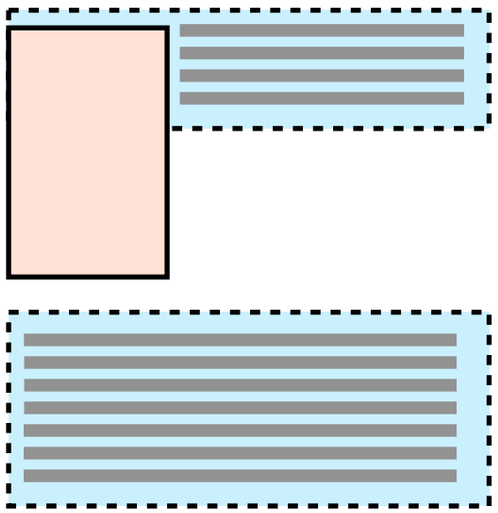




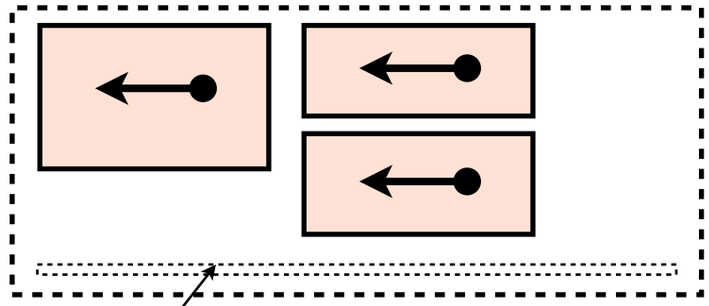


清浮动

1. clear属性 - clear:left|right|both;



clear:both;

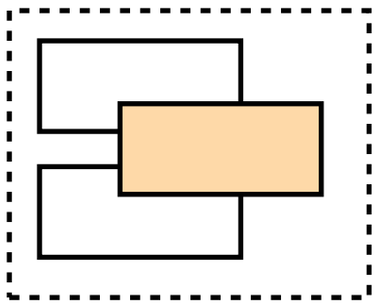


display:block;clear:both;

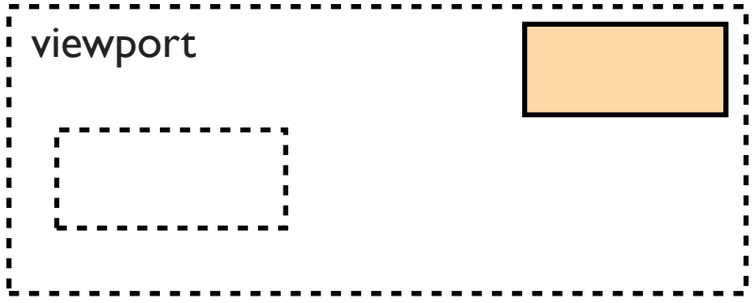
2. 触发形成一个新的block formatting context

Absolute Position - 绝对定位

1. 完全脱离文档流
2. 不会被任何box围绕
3. 若不设宽度将是它实际内容的宽度
4. 把box都变成block-level box，形成一个新的block formatting context
5. 设为position:absolute后，float失效
6. 向上寻找最近的position为relative或absolute的父容器定位直到根结点
7. position:fixed是absolute定位的子类。相对viewport的绝对定位，不受容器限制
8. IE6/Mobile webkit(iOS 3.5除外)不支持position:fixed



```
position:absolute;
top:100px;
left:100px
```



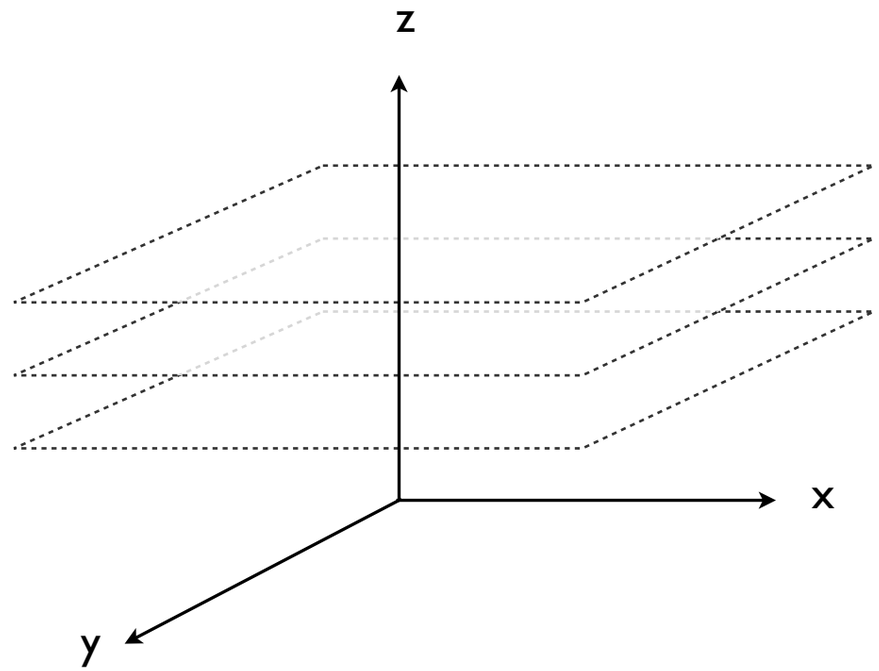

```
position:fixed;
top:10px;
right:10px
```

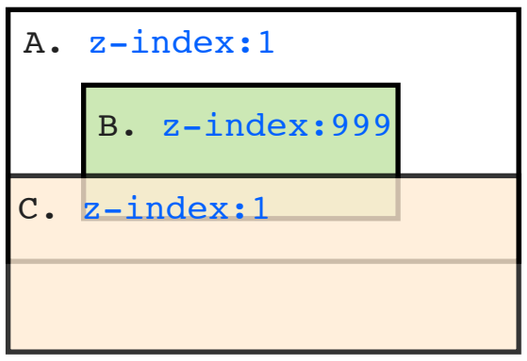
z-index

position: absolute|relative|fixed有效

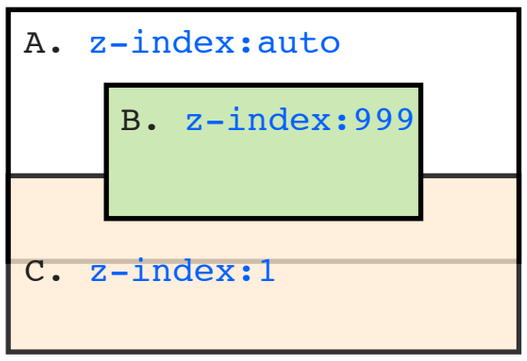
Stack Level：正数或负数

Stacking Context：
z-index值不为auto将建立一个新的stacking context
z-index值为auto，该box的stack level将跟它的父级一致 (注： IE6/7下会不同)





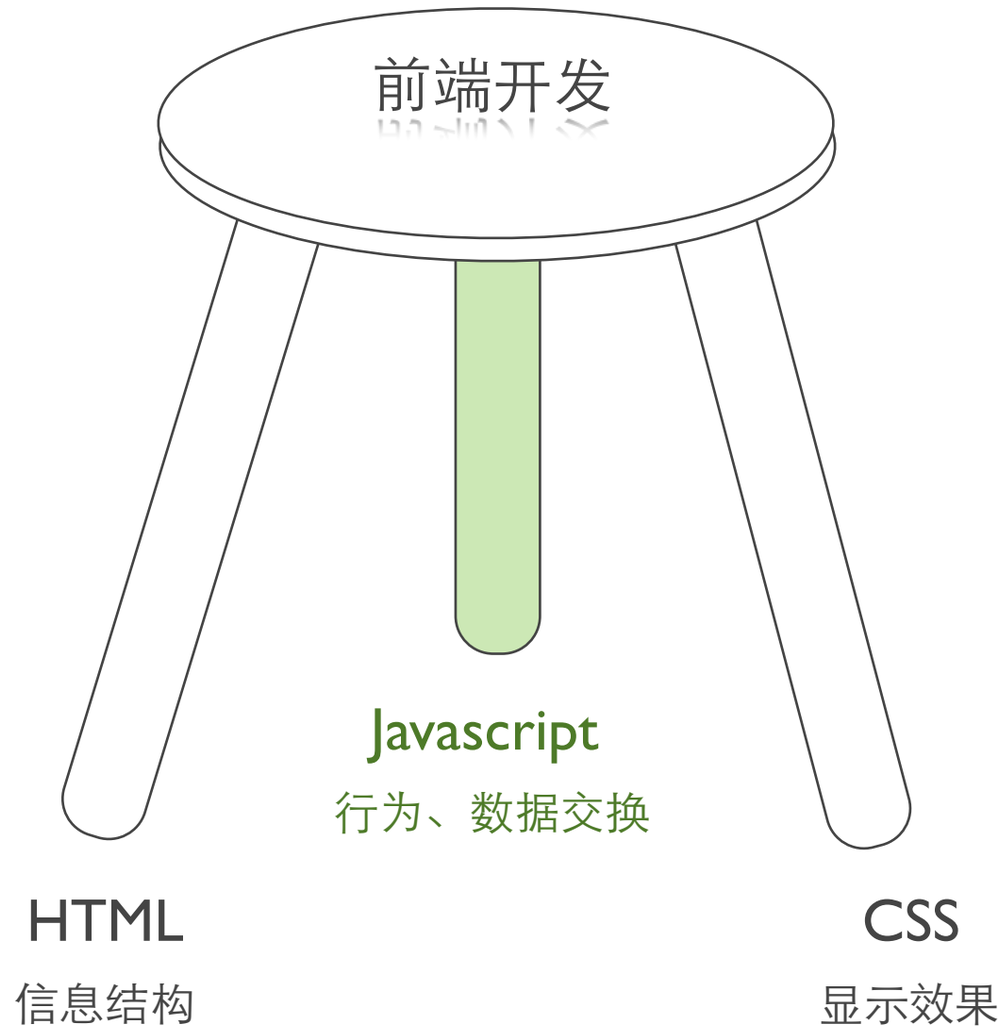
A，C为平行的stacking context。同值，后者居上
A是B的父级，B的z值再大也覆盖不了C



A的z值为auto，变换为和原父级一样的stack level
B也就和A平行，B的z值大于C的，就可以覆盖C
遗憾的是IE6/7不是这样处理的

CSS品质校验工具

http://csslint.net/


前端开发
Javascript
行为、数据交换
HTML
信息结构
CSS
显示效果

语言

数据类型

变量

表达式和运算符

控制语句

Function

继承

闭包

通用类库

数据类型

Number

String

Boolean

Object

- Function
- Array
- Date
- RexExp

null

undefined

数据类型

Number

String

Boolean

Primitive type

Object

- Function
- Array
- Date
- RexExp

null

undefined

数据类型

Number String Boolean	Primitive type
Object - Function - Array - Date - RexExp	Reference type
null	
undefined	

1. 弱类型

```
3*'3'
> 9

3+'3'
> "33"

+'3'
> 3

+new Date
> 1314181420537

'010'|0
> 10

parseInt('010')
> 8

parseInt('010',10)
> 10
```

2. 数值型是"double-precision 64-bit format IEEE 754 values"

```
0.1+0.2
> 0.30000000000000004
```

3. 一切皆对象

```
true.toString()
> "true"
(2).toString()
> "2"

'test'.split('')
> ["t", "e", "s", "t"]
'test'.charAt(0)
> "t"

Number.prototype.length = function(){ return (this + '').length; }
(10086).length()
> 5
```

4. 布尔类型特点

0,“”, NaN, null, undefined为false，其它皆true

```
!!null
> false
!!undefined
> false
!![]
> true
!!{}
> true
```

5. reference类型的特点

```
var obj1 = {a:1};
var obj2 = obj1;
obj2.a
> 1
obj1.a = 2
obj2.a
> 2

var obj1 = [1,2,3];
var obj2 = obj1;
obj1[0] = 99;
obj2
> [99, 2, 3]
```

6. 对象是prototype-base实现继承

```
Object.prototype.newMethod = function() {
  console.log('被污染了');
}
(2).newMethod()
> “被污染了”
(2).hasOwnProperty(newMethod)
> false
```

变量

全局变量

1. 皆是window的成员
2. 没用var声明的变量会被转换为全局变量
3. 页面unload后被GC回收
4. 全局变量是evil

局部变量

1. 定义在函数体内
2. 函数执行完若未被其它对象引用便会被GC回收

Scope chains (作用域链)

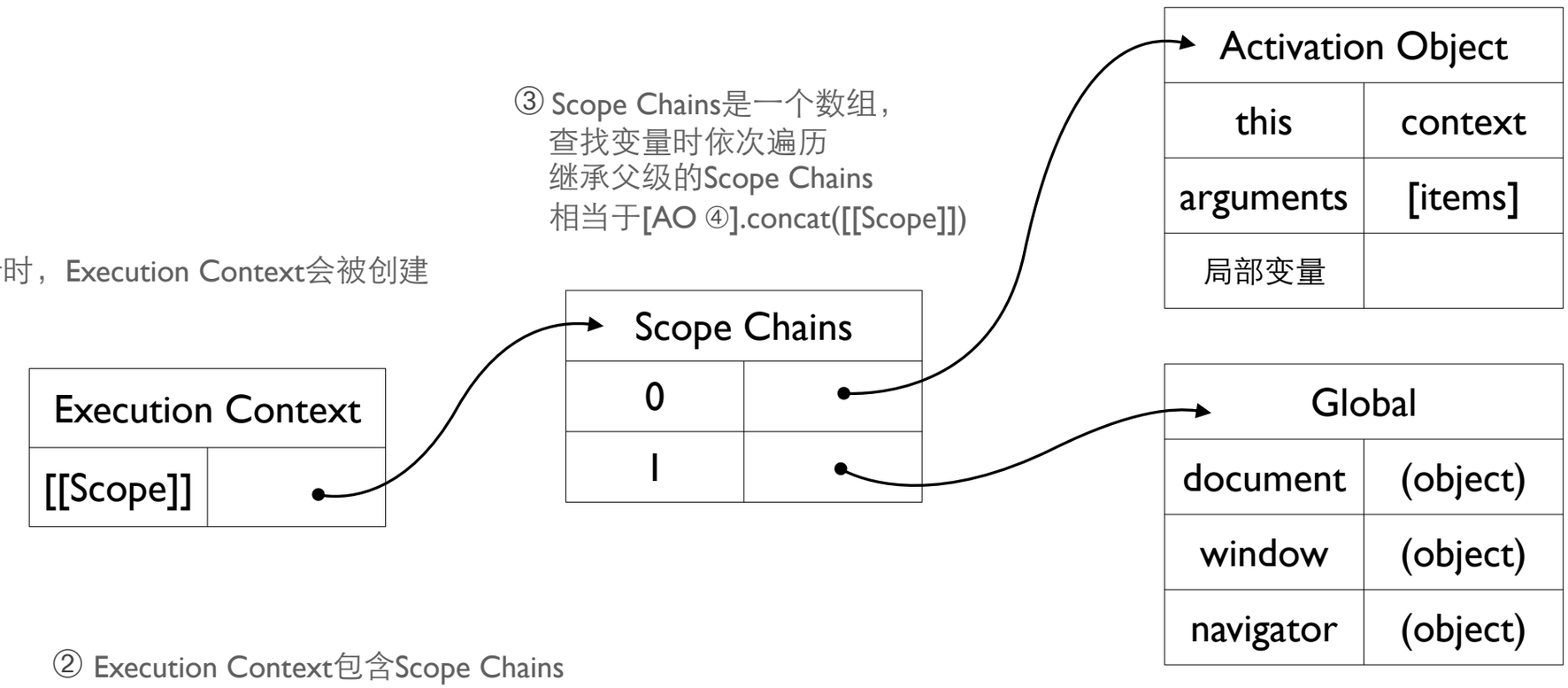
- 影响执行效率的重要因素
- 实现Javascript动态特性的基础机制

Scope Chains

④ Activation Object (激活对象)
是在函数执行时被创建，放在Scope Chains的顶端

③ Scope Chains是一个数组，
查找变量时依次遍历
继承父级的Scope Chains
相当于[AO ④].concat([[Scope]])

① 当函数执行时，Execution Context会被创建



② Execution Context包含Scope Chains

⑤ Global Variable Object所有全局对象

⑥ 没找到抛出ReferenceError异常

详细说明：http://dmitrysoshnikov.com/ecmascript/chapter-4-scope-chain/

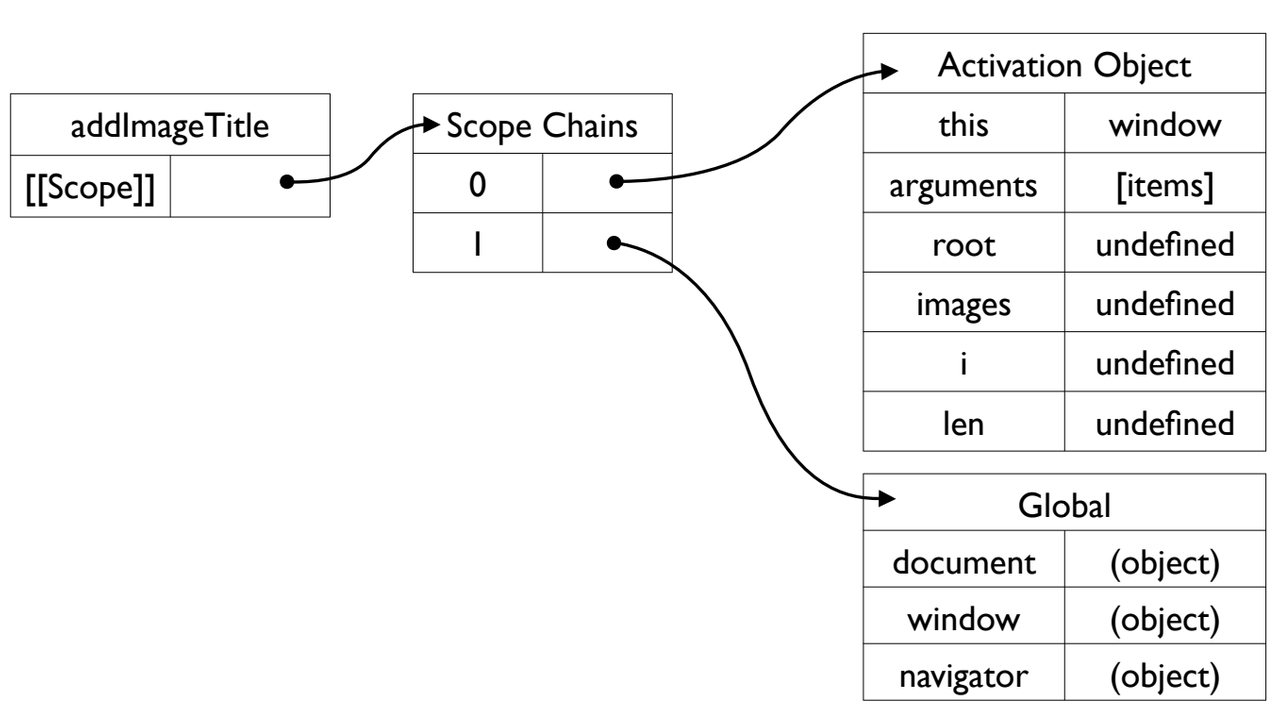
理解从词法环境下的父级继承Scope Chains

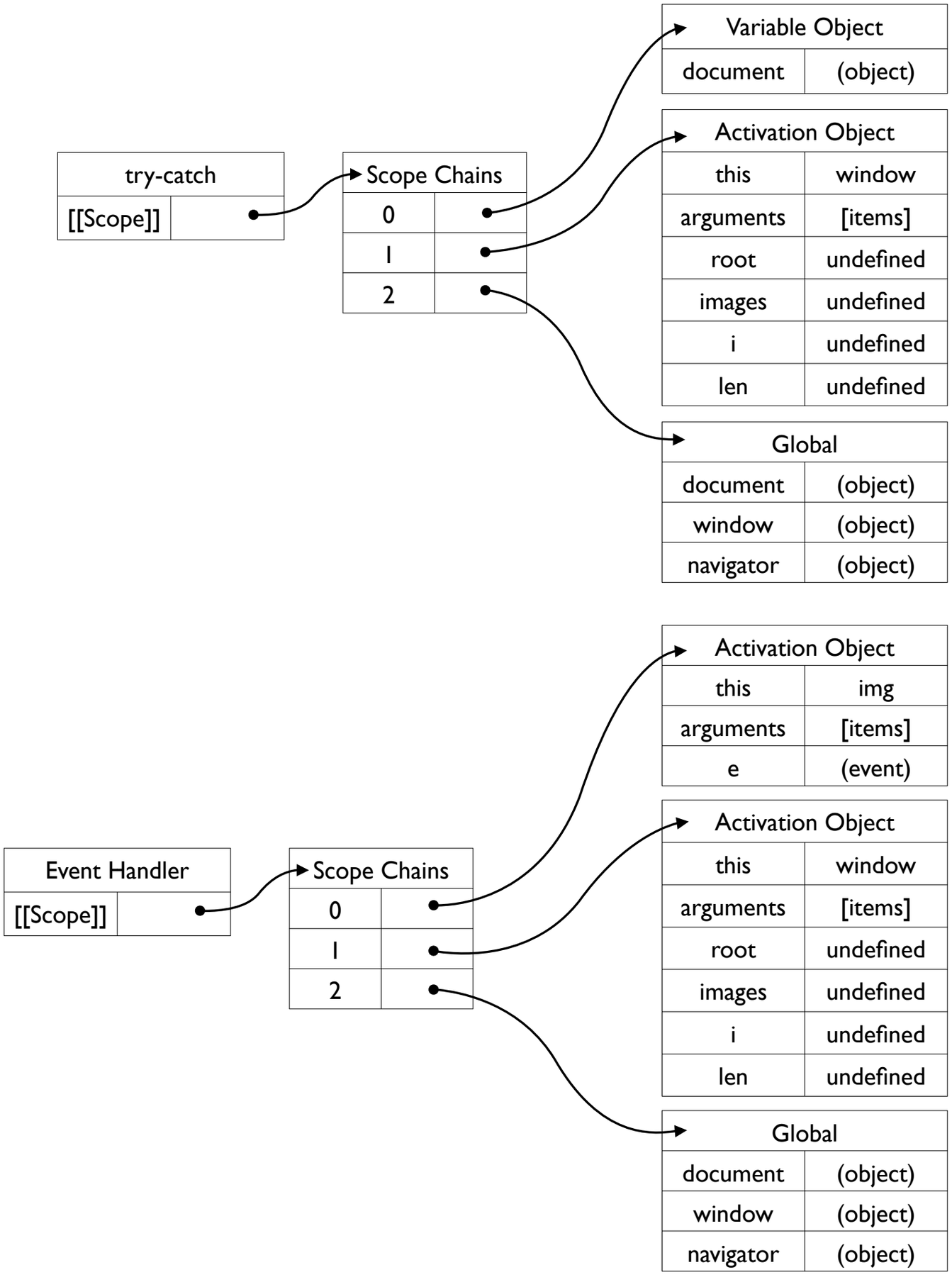
```
var x = 10;

function foo() {
  alert(x);
}

(function () {
  var x = 20;
  foo();
})();
```

```
var x = 10;

(function () {
  var x = 20;

  function foo() {
    alert(x);
  }

  foo();
})();
```

注： with, try-catch均会在scope chains的前面增加一个新的对象

```
function addImageTitle(options) {
  try {
    var root = document.getElementById('content'),
    images = root.getElementsByTagName('img');
    for (var i = 0, len = images.length; i < len; i++) {
       images[i].addEventListener('mouseover', function(e){
         this.title = this.src;
       }, false);
    }
  catch(error) {
    log(error);
  }
}
```




try-catch
[[Scope]]
Scope Chains
0
1
2
Variable Object
document
(object)
Activation Object
this
window
arguments
[items]
root
undefined
images
undefined
i
undefined
len
undefined
Global
document
(object)
window
(object)
navigator
(object)
Event Handler
[[Scope]]
Scope Chains
0
1
2
Activation Object
this
img
arguments
[items]
e
(event)
Activation Object
this
window
arguments
[items]
root
undefined
images
undefined
i
undefined
len
undefined
Global
document
(object)
window
(object)
navigator
(object)

Functions

函数类型：

- 声明型
- 表达式型
- 匿名函数

‘类’和OOP

声明型函数

1. 有特定名称
2. 代码的位置要么在程序级的全局上下文上
   要么在另一个函数体内
3. 在进入上下文阶段创建
4. 影响Variable Object
5. 函数作为对象

```
function foo(e) {
    return e;
}

foo.property = value;

foo.length
> 1

foo.name
> "foo"
```

声明型函数

1. 有特定名称
2. 代码的位置要么在程序级的全局上下文上
   要么在另一个函数体内
3. 在进入上下文阶段创建
4. 影响Variable Object
5. 函数作为对象

①

```
function foo() {
  console.log('foo');
}
```

```
function foo(e) {
    return e;
}

foo.property = value;

foo.length
> 1

foo.name
> "foo"
```

声明型函数

1. 有特定名称
2. 代码的位置要么在程序级的全局上下文上
   要么在另一个函数体内
3. 在进入上下文阶段创建
4. 影响Variable Object
5. 函数作为对象

```
function foo(e) {
    return e;
}

foo.property = value;

foo.length
> 1

foo.name
> "foo"
```

①

```
function foo() {
  console.log('foo');
}
```

②

```
function foo() {
  console.log('foo');
  function foo1() {
    console.log('foo1');
  }
}


if (0) {
  function foo1() {
    console.log('foo1');
  }
}
foo1();
```

声明型函数

1. 有特定名称
2. 代码的位置要么在程序级的全局上下文上
   要么在另一个函数体内
3. 在进入上下文阶段创建
4. 影响Variable Object
5. 函数作为对象

```
function foo(e) {
    return e;
}

foo.property = value;

foo.length
> 1

foo.name
> "foo"
```

①

```
function foo() {
  console.log('foo');
}
```

②

```
function foo() {
  console.log('foo');
  function foo1() {
    console.log('foo1');
  }
}


if (0) {
  function foo1() {
    console.log('foo1');
  }
}
foo1();
```

③ ④

```
foo();
...
function foo() {
  console.log('foo');
}
```

表达式型函数

1. 在源码中出现在表达式的位置上
2. 函数名可选
3. 不影响Variable Object
4. 在代码执行阶段创建

表达式型函数

1. 在源码中出现在表达式的位置上
2. 函数名可选
3. 不影响Variable Object
4. 在代码执行阶段创建

①

```
var foo = function() {
  console.log('foo');
};
```

表达式型函数

1. 在源码中出现在表达式的位置上
2. 函数名可选
3. 不影响Variable Object
4. 在代码执行阶段创建

①

```
var foo = function() {
  console.log('foo');
};
```

②

```
var foo = function _foo() {
  console.log(typeof _foo); //function
};
_foo(); //ReferenceError: Can't find variable: _foo
foo.name
> “_foo”
```

表达式型函数

1. 在源码中出现在表达式的位置上
2. 函数名可选
3. 不影响Variable Object
4. 在代码执行阶段创建

①

```
var foo = function() {
  console.log('foo');
};
```

②

```
var foo = function _foo() {
  console.log(typeof _foo); //function
};
_foo(); //ReferenceError: Can't find variable: _foo
foo.name
> “_foo”
```

③ ④

```
var foo = function(callback) {
  callback();
};

foo(function _foo1(){
   console.log();
});

var bar = (foo % 2 == 0 ?
             function() { alert(0); } :
             function() { alert(1); }
);
bar();
```

④

```
foo();
var foo = function() {
  console.log(‘foo’);
};
```

```
function foo(){
  console.log('foo');
}();


function(){
  console.log('foo');
}();
```

```
function foo(){
  console.log('foo');
}();
```

```
function(){
  console.log('foo');
}();
```

```
(function foo(){
  console.log('foo');
}());
```

```
1,function(){
   console.log('foo');
}();
```

```
!function(){
   console.log('foo');
}();
```

```
+function(){
   console.log('foo');
}();
```

```
 function foo(){
   console.log('foo');
 }();
```

```
function(){
   console.log('foo');
 }();

> SyntaxError: Parse error
```

```
(function foo(){
  console.log('foo');
}());

> foo

1,function(){
   console.log('foo');
}();

> foo

!function(){
   console.log('foo');
}();

> foo

+function(){
   console.log('foo');
}();

 > foo
```

声明型函数不能立即执行
声明型函数变换为表达式型，并可以立即执行

```
 function foo(){
   console.log('foo');
 }();
```

```
function(){
   console.log('foo');
 }();

> SyntaxError: Parse error
```

```
(function foo(){
  console.log('foo');
}());

> foo

1,function(){
   console.log('foo');
}();

> foo

!function(){
   console.log('foo');
}();

> foo

+function(){
   console.log('foo');
}();

 > foo
```

匿名函数

作为对象出现在表达式上

```
condition ?
  function(){ console.log(1); }() :
  function(){ console.log(2); }();
```

避免全局变量

```
(function(){
  var i = 1;
})();
```

作为参数

```
bottom.addEventListener(‘click’, function(e){}, false);
```

作为闭包

```
var add = function() {
   var i = 1;
   return function(e) {
      i = i + e;
      return i;
   }
}();
```

“类”和OO

1. 类是对象方法和属性的封装，实现代码重用。
   一个类可以实例化多个对象
2. OO的特点：封装、 继承、 扩展、多态
3. Javascript没有类，但完全可以实现OO的所有特点

多态

1. 当函数在一个对象的上下文中被调用，
   this指向这个对象

2. 执行函数的call和apply方法，
   将在给定对象的上下文中调用函数

多态

1. 当函数在一个对象的上下文中被调用，this指向这个对象

2. 执行函数的call和apply方法，将在给定对象的上下文中调用函数

```
sample 1:

var obj = {
  input: function() {
      return this.handle(arguments[0], arguments[1]);
  },
  handle: function(inp1, inp2){
    return inp1 + inp2;
  }
};

var obj1 = {
  handle: function(inp1, inp2) {
    return inp1 * inp2;
  }
}

alert(obj.input(10,20));
alert(obj.input.call(obj1, 10,20));
```

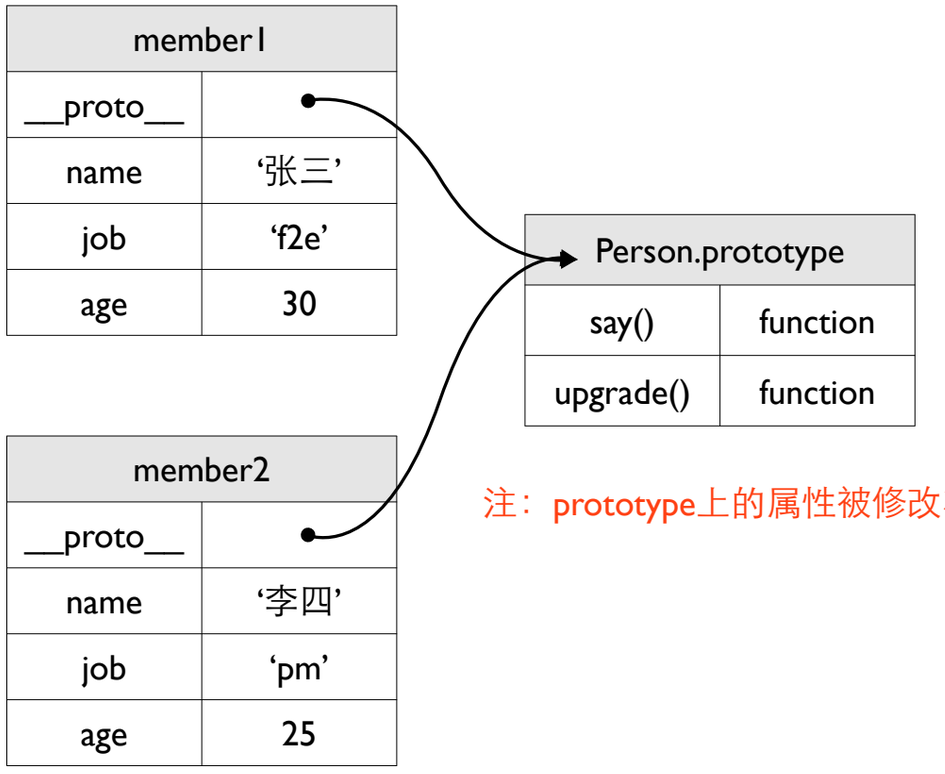
```
sample 2:

function A(){
  this.method = function() { alert(1); };
}

function B(){
  A.call(this, arguments);
  this.method = function() { alert(2); };
}

function C(){
  A.call(this, arguments);
  this.method = function() { alert(3); };
}

(new A).method();
(new B).method();
(new C).method();
```

封装

```
function Person(options) {
  // 私有属性
  var grade = 1;

  // 公共属性
  this.name = options.name;
  this.job = options.job;
  this.age = options.age;

  // 公共方法
  this.say = function(s) {
    alert(s);
  };

  this.upgrade = function() {
    grade++;
    return grade;
  };
}


// 实例化对象
var member1 = new Person({
      name: '张三',
      job: 'f2e',
      age: 30
    });

member1.say('Hello');
alert(member1.get('name'));
alert(member1.upgrade());
```

member1	
name	‘张三’
job	‘f2e’
age	30
say()	function
upgrade()	function

```
function Person(options) {
  // 公共属性
  this.grade = 1;
  this.name = options.name;
  this.job = options.job;
  this.age = options.age;
}
// 公共方法
Person.prototype = {
  say: function(s) {
    alert(s);
  },
  upgrade: function() {
    this.grade++;
    return this.grade;
  }
};


var member1 = new Person({
      name: '张三',
      job: 'f2e',
      age: 30
    });

var member2 = new Person({
      name: '李四',
      job: 'pm',
      age: 25
    });
```



注：prototype上的属性被修改将会影响所有实例对象

多重继承

不推荐使用，问题：

1. 增加复杂度
2. 影响效率
3. 原型链上的属性被误改，影响范围广

建议：用mixin方式替代

```
function Person(options) {
  this.grade = 1;
  this.name = options.name;
  this.job = options.job;
  this.age = options.age;
}

Person.prototype.say = function(s) {
  alert(s);
};
Person.prototype.upgrade = function() {
  this.grade++;
  return this.grade;
};

function Engineer(options) {
  Person.call(this, options);
  this.skill = options.skill;
}

Engineer.prototype = Person.prototype;

Engineer.prototype.coding = function(code) {
  alert(code);
};


var member1 = new Engineer({
      name: '张三',
      job: 'f2e',
      age: 30,
      skill: 'html/css/js'
    });

member1.coding('<h1>');
member1.say('Hello');
alert(member1.name);
alert(member1.skill);
alert(member1.upgrade());
```

扩展

mixin方式

```
function extend(target, source) {
  for (var i in source) {
    if (source.hasOwnProperty(i)) {
      target[i] = source[i];
    }
  }
}

function Person(options) {
  this.grade = 1;
  this.name = options.name;
  this.job = options.job;
  this.age = options.age;
}

Person.prototype = {
  say: function(s) {
    alert(s);
  },
  upgrade: function(s) {
    this.grade++;
    return this.grade;
  }
};

function PM(options) {
  Person.call(this, options);
  this.skill = options.skill;
}

PM.prototype.prd = function(prd) {
  alert(prd);
};

function Engineer(options) {
  Person.call(this, options);
  this.skill = options.skill;
}

Engineer.prototype.coding = function(code) {
  alert(code);
};

extend(Engineer.prototype, Person.prototype);
extend(Engineer.prototype, PM.prototype);

var member1 = new Engineer({
      name: '张三',
      job: 'f2e',
      age: 30,
      skill: 'html/css/js'
});
```

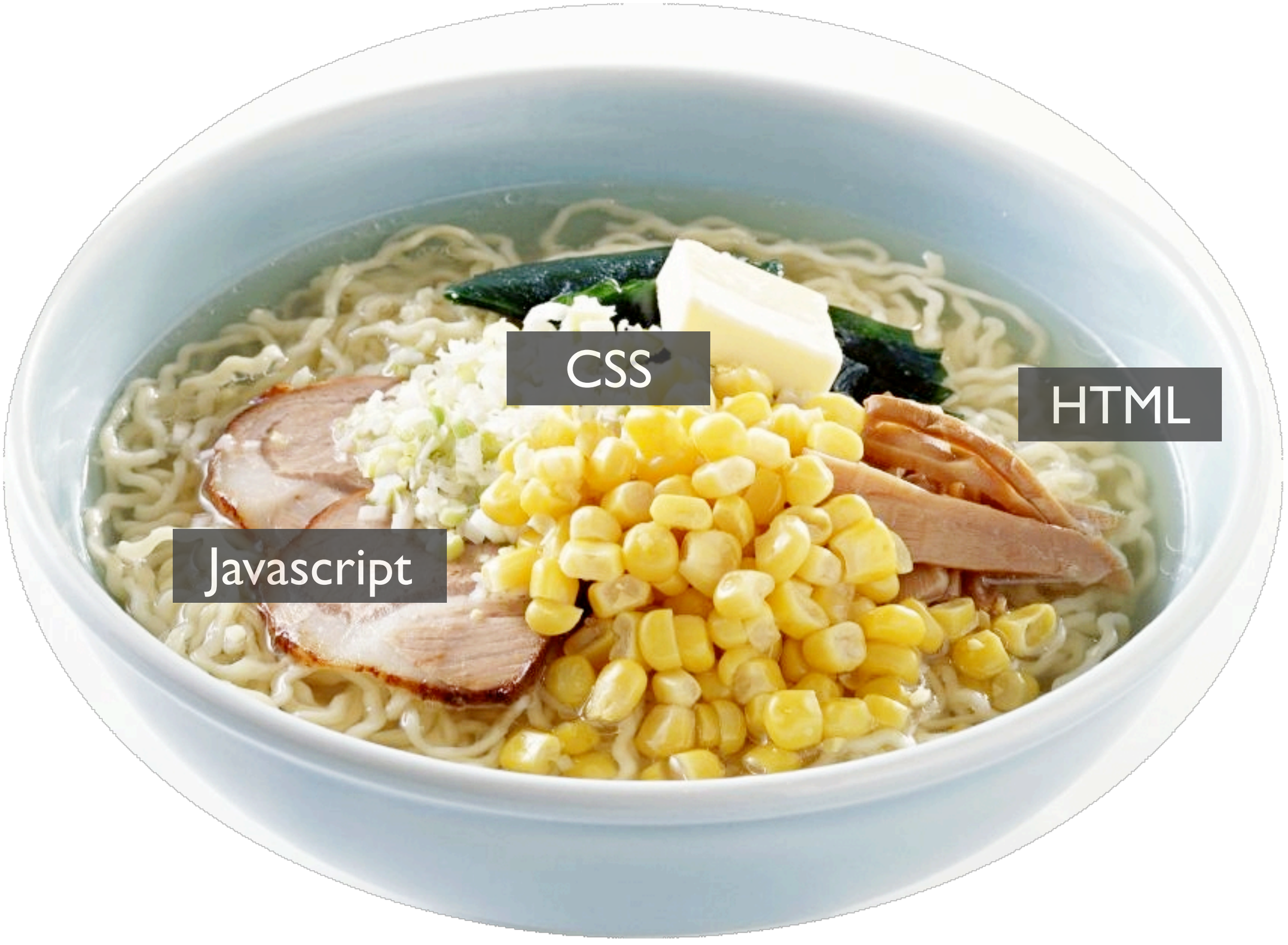
```
member1.coding('<h1>');
member1.prd('update');
member1.say('Hello');
alert(member1.name);
alert(member1.skill);
alert(member1.upgrade());

Person.prototype.say = function() {};
member1.say('hi'); //不受影响
```

闭包 (Closure)

闭包是可以访问一个函数局部变量的函数。

```
var add = function(){
  var i = 0;
  return function(){
    return ++i;
  }
}();
```

```
$(window).bind('scroll', function(){
  var t;
    return function(e) {
      if (t) {
        clearTimeout(t);
      }
      t = setTimeout(function(){ handle(e); }, 20);
    };
}());
```


CSS
HTML
Javascript

1. 遵循统一的开发规范
   - 统一代码风格Guideline (bit.ly/douban-javascript, bit.ly/douban-css)

2. 建立静态资源文件管理系统
   - 模块化
   - 自动化
   - 开发环境下代码对开发者友好
     生产环境下代码对用户友好

3. 品质检验工具化
   - douban-jslint, csslint
   - 自动化测试

4. 合理的架构
   - 建立通用管理机制、类库
   - 分离业务逻辑和通用类库
   - 合理利用代码设计模式

5. 三层分离和模块化开发
   - 利用好模板系统，提高重用
   - 合理组织JS/CSS，尤其inline代码

谢谢!

skype: kejunz
mail/GTalk: listenpro@gmail.com