VisComposer 代码手册

# 引言

## 程序基础

VisComposer由JavaScript语言编写，架设在服务器上，可由浏览器访问运行。

## 运行环境

Chrome 39.0.1916以上

Apache

运行环境搭建教程

## 编写与调试工具

Sublime Text

Chrome Developer Tools

# 基本组成

## 代码目录结构

* css CSS文件
* js JS文件
  + core 核心机制类
  + data 数据管理类
  + scenegraph Scenegraph相关类
  + ui 界面类
    - domobject 动态对象（需要按情况生成的部分，如各种节点等）
    - static 静态窗口
    - manager.js UI管理、调用接口
  + util
  + vendor
  + workflow Workflow相关类
  + viscomposer.js include/程序入口（window.load）
* resource **服务器端**资源

## 模块组成

* 主体结构
* UI

Link（）

Port（）

workflow的输出

input

defines

input

ports

Module

Module

（）

Scnegraph（）

分发等定义

（）

children

root

Node（）

Workflow

Node

Workflow

Node

Workflow

* 绘制

## 其他组成部分

# 代码设计

## JavaScript编写技巧

JavaScript: The Good Parts

JavaScript高级程序设计

W3CSchool

### 面向对象编程

#### 类的定义

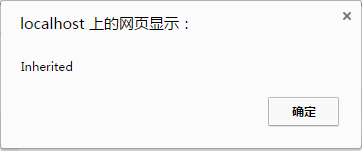
JavaScript中，类的定义非常简单。任意一个function在以new进行调用时即可作为构造函数实例化一个类。此外，通过this.可以定义/访问成员变量、通过对构造函数的prototype的成员赋值可以定义成员函数。

|  |
| --- |
| var Class=function(){//类的构造函数  this.x=1;//定义成员变量并赋值  };  Class.prototype.foo=function(){//定义成员函数  alert(this.x);//调用成员变量  };  Class.prototype.bar=function(){  this.foo();//调用成员函数  };  var c=new Class();//实例化  c.bar(); |

#### 继承与多态

JavaScript中实现类的继承有多种方式，这里使用其中一种：

|  |
| --- |
| var Base=function(){//基类  this.init();//可能会调用到衍生类的init函数  ...  };  Base.prototype.init=function(){  ...  alert("Base");  };  var Inherited=function(){//衍生类  Base.call(this);//调用基类构造函数  ...  };  Inherited.prototype=Object.create(Base.prototype);//构建原型链  Inherited.prototype.constructor=Inherited;//指定构造函数  Inherited.prototype.init=function(){  ...  alert("Inherited");  };  ...  var a=new Inherited(); |



由于JavaScript原型链的工作方式，可以看成所有函数都是虚函数。值得注意的是，在基类构造函数中以this.调用自身成员函数可能会调用到衍生类的对应函数，这种情况会发生在衍生类调用基类构造函数时（如下方图，上方代码中的init函数调用的是衍生类的版本）。在一定的设计意图下这可能是被期望的，但也可能造成错误。如需要强制确保调用基类函数，可以使用如下写法：

|  |
| --- |
| var Base=function(){//基类  Base.prototype.init.call(this);  ...  };  Base.prototype.init=function(){  ...  }; |

或者

|  |
| --- |
| var Base=function(){  baseInit.call(this);  ...  };  var baseInit=Base.prototype.init=function(){  ...  }; |

#### 控制成员变量访问权限

JavaScript中所有成员变量均为public，但可以通过一些方法实现类似public/private的区分：

|  |
| --- |
| var Class=(function(){  var a=0;//static private  return function(){//构造函数  this.b=1;//public  };  })();  Class.c=2;//public static  var c=new Class();  alert(Class.a);//undefined  alert(c.b);//1  alert(Class.c);//2 |

非static的private相对复杂一些，可以有如下实现：

|  |
| --- |
| var Class=(function(){  return function(){//构造函数  var a=1;//private  this.getA=function(){return a;}//拥有访问权限的方法  };  })();  var c=new Class();  alert(c.a);//undefined  alert(c.getA());//1 |

如使用setter和getter还能产生更加灵活的访问方式：

|  |
| --- |
| var Class=(function(){  var a=0,b=1;//private  var Class=function(){//构造函数  };  Class.prototype={  get a(){//可读不可写  return a;  },  set b(newB){//可写不可读  b=newB;  alert(b);  }  };  return Class;  })();  var c=new Class();  alert(c.a);//0  c.a=2;  alert(c.a);//0  c.b=3;//3 |

另一种定义setter和getter的方法：

|  |
| --- |
| Object.defineProperty(Class.prototype,"a",{  get:function(){return a;},  set:function(newA){a=newA;},  }); |

## 核心机制

### VObject

VisComposer中所有关键类的基类，简单来说，提供了一个uuid作为识别，同时提供一个继承机制以方便类的管理、存储和导入。

类工厂与继承机制细节

VObject类中定义了define方法，用于生成继承VObject的类（VObject本身的构造函数是private的，原则上这是继承VObject的唯一方法）。



其中的参数：

* name

类的名称，这个名称会被注册用于继续继承、模板化等。

* baseName

继承的基类名称，可以是"VObject"或者被注册的其他name。

* constructor

构造函数，但实际的构造函数会包括一些其他部分，比如调用基类构造函数等（所以构造函数不需要写出调用基类构造函数的语句）。

* members

注册成员变量，只有members中包含的变量在模板化存储和读取时会被存储和读取，其他均被视为运行时生成而不进行存储。关于存储和读取的其他细节见（）。

下面是一个简单的样例：

|  |
| --- |
| viscomposer.ui.DOMObject=(function(){  var DOMObject=  viscomposer.VObject.define("DOMObject",null,function(){  this.obj=null;  this.dom=null;  this.container=null;  this.parentDom=null;  update.call(this);  },["obj","dom","container","parentDom"]);  var prototype=DOMObject.prototype;  var update=prototype.update=function(){  //TODO  };  prototype.appendTo=function(parentDom){  $(this.dom).appendTo(parentDom);  this.parentDom=parentDom;  };  return DOMObject;  })(); |

### 代码生成

#### 变量名

变量名可以是自动生成的、由用户指定的以及代码中需要用到的临时变量。为了使生成代码时变量名不冲突，需要使用viscomposer.util.processConfilt函数，具体使用方法如下：

|  |
| --- |
| var varNames={};  var actualInput=viscomposer.util.processConflict('input',varNames);  var actualTmp=viscomposer.util.processConflict('tmp',varNames);  var actualI=viscomposer.util.processConflict('i',varNames); |

#### 代码编译

文本形式的代码需要通过viscomposer.Environment.build()进行编译。build函数本质就是一个拥有一个闭包环境（和其中的私有变量、函数）、且将参数直接转接到eval函数。这样做可以为代码提供一些额外的便利，如可以使用一些常用函数min/max等，能简化用户编写代码时的代码量（否则可能需要用类似viscomposer.Math.min这样的形式）;另外还能控制代码运行所处的环境（如屏蔽掉可能造成不良影响的全局变量）。

## 设计原则

### 其他

* 所有语句行结束时需加上分号。虽然不一定会影响运行，但在进行诸如代码自动压缩等步骤时会导致软件识别出现问题。容易漏掉的主要是一些函数定义之后的分号（因为紧跟在大括号之后）。

# 参考