[S.O.L.I.D五大原则之单一职责SRP](http://www.cnblogs.com/TomXu/archive/2012/01/06/2305513.html)

**前言**

五大原则分别是：

1. The Single Responsibility Principle（单一职责SRP）
2. The Open/Closed Principle（开闭原则OCP）
3. The Liskov Substitution Principle（里氏替换原则LSP）
4. The Interface Segregation Principle（接口分离原则ISP）
5. The Dependency Inversion Principle（依赖反转原则DIP）

五大原则，相对于JavaScript这种以原型为base的动态类型语言来说还为数不多，英文原文：http://freshbrewedcode.com/derekgreer/2011/12/08/solid-javascript-single-responsibility-principle/

**单一职责**

单一职责的描述如下：

A class should have only one reason to change  
类发生更改的原因应该只有一个

一个类（JavaScript下应该是一个对象）应该有一组紧密相关的行为的意思是什么？遵守单一职责的好处是可以让我们很容易地来维护这个对象，当一个对象封装了很多职责的话，一旦一个职责需要修改，势必会影响该对象想的其它职责代码。通过解耦可以让每个职责工更加有弹性地变化。

不过，我们如何知道一个对象的多个行为构造多个职责还是单个职责？我们可以通过参考[Object Design: Roles, Responsibilies, and Collaborations](http://www.amazon.com/Object-Design-Roles-Responsibilities-Collaborations/dp/0201379430)一书提出的Role Stereotypes概念来决定，该书提出了如下Role Stereotypes来区分职责：

1. Information holder – 该对象设计为存储对象并提供对象信息给其它对象。
2. Structurer – 该对象设计为维护对象和信息之间的关系
3. Service provider – 该对象设计为处理工作并提供服务给其它对象
4. Controller – 该对象设计为控制决策一系列负责的任务处理
5. Coordinator – 该对象不做任何决策处理工作，只是delegate工作到其它对象上
6. Interfacer – 该对象设计为在系统的各个部分转化信息（或请求）

一旦你知道了这些概念，那就狠容易知道你的代码到底是多职责还是单一职责了。

**实例代码**

该实例代码演示的是将商品添加到购物车，代码非常糟糕，代码如下：

function Product(id, description) {  
 this.getId = function () {  
 return id;  
 };  
 this.getDescription = function () {  
 return description;  
 };  
}  
  
function Cart(eventAggregator) {  
 var items = [];  
  
 this.addItem = function (item) {  
 items.push(item);  
 };  
}  
  
(function () {  
 var products = [new Product(1, "Star Wars Lego Ship"),  
 new Product(2, "Barbie Doll"),  
 new Product(3, "Remote Control Airplane")],  
cart = new Cart();  
  
 function addToCart() {  
 var productId = $(this).attr('id');  
 var product = $.grep(products, function (x) {  
 return x.getId() == productId;  
 })[0];  
 cart.addItem(product);  
  
 var newItem = $('<li></li>').html(product.getDescription()).attr('id-cart', product.getId()).appendTo("#cart");  
 }  
  
 products.forEach(function (product) {  
 var newItem = $('<li></li>').html(product.getDescription())  
 .attr('id', product.getId())  
 .dblclick(addToCart)  
 .appendTo("#products");  
 });  
})();

该代码声明了2个function分别用来描述product和cart，而匿名函数的职责是更新屏幕和用户交互，这还不是一个很复杂的例子，但匿名函数里却包含了很多不相关的职责，让我们来看看到底有多少职责：

1. 首先，有product的集合的声明
2. 其次，有一个将product集合绑定到#product元素的代码，而且还附件了一个添加到购物车的事件处理
3. 第三，有Cart购物车的展示功能
4. 第四，有添加product item到购物车并显示的功能

**重构代码**

让我们来分解一下，以便代码各自存放到各自的对象里，为此，我们参考了martinfowler的事件聚合（[Event Aggregator](http://martinfowler.com/eaaDev/EventAggregator.html)）理论在处理代码以便各对象之间进行通信。

首先我们先来实现事件聚合的功能，该功能分为2部分，1个是Event，用于Handler回调的代码，1个是EventAggregator用来订阅和发布Event，代码如下：

function Event(name) {  
 var handlers = [];  
  
 this.getName = function () {  
 return name;  
 };  
  
 this.addHandler = function (handler) {  
 handlers.push(handler);  
 };  
  
 this.removeHandler = function (handler) {  
 for (var i = 0; i < handlers.length; i++) {  
 if (handlers[i] == handler) {  
 handlers.splice(i, 1);  
 break;  
 }  
 }  
 };  
  
 this.fire = function (eventArgs) {  
 handlers.forEach(function (h) {  
 h(eventArgs);  
 });  
 };  
 }  
  
 function EventAggregator() {  
 var events = [];  
  
 function getEvent(eventName) {  
 return $.grep(events, function (event) {  
 return event.getName() === eventName;  
 })[0];  
 }  
  
 this.publish = function (eventName, eventArgs) {  
 var event = getEvent(eventName);  
  
 if (!event) {  
 event = new Event(eventName);  
 events.push(event);  
 }  
 event.fire(eventArgs);  
 };  
  
 this.subscribe = function (eventName, handler) {  
 var event = getEvent(eventName);  
  
 if (!event) {  
 event = new Event(eventName);  
 events.push(event);  
 }  
  
 event.addHandler(handler);  
 };  
 }

然后，我们来声明Product对象，代码如下：

function Product(id, description) {  
 this.getId = function () {  
 return id;  
 };  
 this.getDescription = function () {  
 return description;  
 };  
}

接着来声明Cart对象，该对象的addItem的function里我们要触发发布一个事件itemAdded，然后将item作为参数传进去。

function Cart(eventAggregator) {  
 var items = [];  
  
 this.addItem = function (item) {  
 items.push(item);  
 eventAggregator.publish("itemAdded", item);  
 };  
}

CartController主要是接受cart对象和事件聚合器，通过订阅itemAdded来增加一个li元素节点，通过订阅productSelected事件来添加*product。*

function CartController(cart, eventAggregator) {  
 eventAggregator.subscribe("itemAdded", function (eventArgs) {  
 var newItem = $('<li></li>').html(eventArgs.getDescription()).attr('id-cart', eventArgs.getId()).appendTo("#cart");  
 });  
  
 eventAggregator.subscribe("productSelected", function (eventArgs) {  
 cart.addItem(eventArgs.product);  
 });  
}

Repository的目的是为了获取数据（可以从ajax里获取），然后暴露get数据的方法。

function ProductRepository() {  
 var products = [new Product(1, "Star Wars Lego Ship"),  
 new Product(2, "Barbie Doll"),  
 new Product(3, "Remote Control Airplane")];  
  
 this.getProducts = function () {  
 return products;  
 }  
}

ProductController里定义了一个onProductSelect方法，主要是发布触发productSelected事件，forEach主要是用于绑定数据到产品列表上，代码如下：

function ProductController(eventAggregator, productRepository) {  
 var products = productRepository.getProducts();  
  
 function onProductSelected() {  
 var productId = $(this).attr('id');  
 var product = $.grep(products, function (x) {  
 return x.getId() == productId;  
 })[0];  
 eventAggregator.publish("productSelected", {  
 product: product  
 });  
 }  
  
 products.forEach(function (product) {  
 var newItem = $('<li></li>').html(product.getDescription())  
 .attr('id', product.getId())  
 .dblclick(onProductSelected)  
 .appendTo("#products");  
 });  
}

最后声明匿名函数（需要确保HTML都加载完了才能执行这段代码，比如放在jQuery的ready方法里）：

(function () {  
 var eventAggregator = new EventAggregator(),  
cart = new Cart(eventAggregator),  
cartController = new CartController(cart, eventAggregator),  
productRepository = new ProductRepository(),  
productController = new ProductController(eventAggregator, productRepository);  
})();

可以看到匿名函数的代码减少了很多，主要是一个对象的实例化代码，代码里我们介绍了Controller的概念，他接受信息然后传递到action，我们也介绍了Repository的概念，主要是用来处理product的展示，重构的结果就是写了一大堆的对象声明，但是好处是每个对象有了自己明确的职责，该展示数据的展示数据，改处理集合的处理集合，这样耦合度就非常低了。

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

function Event(name) {  
 var handlers = [];  
  
 this.getName = function () {  
 return name;  
 };  
  
 this.addHandler = function (handler) {  
 handlers.push(handler);  
 };  
  
 this.removeHandler = function (handler) {  
 for (var i = 0; i < handlers.length; i++) {  
 if (handlers[i] == handler) {  
 handlers.splice(i, 1);  
 break;  
 }  
 }  
 };  
  
 this.fire = function (eventArgs) {  
 handlers.forEach(function (h) {  
 h(eventArgs);  
 });  
 };  
 }  
  
 function EventAggregator() {  
 var events = [];  
  
 function getEvent(eventName) {  
 return $.grep(events, function (event) {  
 return event.getName() === eventName;  
 })[0];  
 }  
  
 this.publish = function (eventName, eventArgs) {  
 var event = getEvent(eventName);  
  
 if (!event) {  
 event = new Event(eventName);  
 events.push(event);  
 }  
 event.fire(eventArgs);  
 };  
  
 this.subscribe = function (eventName, handler) {  
 var event = getEvent(eventName);  
  
 if (!event) {  
 event = new Event(eventName);  
 events.push(event);  
 }  
  
 event.addHandler(handler);  
 };  
 }  
  
 function Product(id, description) {  
 this.getId = function () {  
 return id;  
 };  
 this.getDescription = function () {  
 return description;  
 };  
 }  
  
 function Cart(eventAggregator) {  
 var items = [];  
  
 this.addItem = function (item) {  
 items.push(item);  
 eventAggregator.publish("itemAdded", item);  
 };  
 }  
  
 function CartController(cart, eventAggregator) {  
 eventAggregator.subscribe("itemAdded", function (eventArgs) {  
 var newItem = $('<li></li>').html(eventArgs.getDescription()).attr('id-cart', eventArgs.getId()).appendTo("#cart");  
 });  
  
 eventAggregator.subscribe("productSelected", function (eventArgs) {  
 cart.addItem(eventArgs.product);  
 });  
 }  
  
 function ProductRepository() {  
 var products = [new Product(1, "Star Wars Lego Ship"),  
 new Product(2, "Barbie Doll"),  
 new Product(3, "Remote Control Airplane")];  
  
 this.getProducts = function () {  
 return products;  
 }  
 }  
  
 function ProductController(eventAggregator, productRepository) {  
 var products = productRepository.getProducts();  
  
 function onProductSelected() {  
 var productId = $(this).attr('id');  
 var product = $.grep(products, function (x) {  
 return x.getId() == productId;  
 })[0];  
 eventAggregator.publish("productSelected", {  
 product: product  
 });  
 }  
  
 products.forEach(function (product) {  
 var newItem = $('<li></li>').html(product.getDescription())  
 .attr('id', product.getId())  
 .dblclick(onProductSelected)  
 .appendTo("#products");  
 });  
 }  
  
 (function () {  
 var eventAggregator = new EventAggregator(),  
 cart = new Cart(eventAggregator),  
 cartController = new CartController(cart, eventAggregator),  
 productRepository = new ProductRepository(),  
 productController = new ProductController(eventAggregator, productRepository);  
 })();

[复制代码](javascript:void(0);)

**总结**

看到这个重构结果，真的有必要做这么复杂么？我只能说：要不要这么做取决于你项目的情况。  
  
如果你的项目是个是个非常小的项目，代码也不是很多，那其实是没有必要重构得这么复杂，但如果你的项目是个很复杂的大型项目，或者你的小项目将来可能增长得很快的话，那就在前期就得考虑SRP原则进行职责分离了，这样才有利于以后的维护。