[设计模式之享元模式](http://www.cnblogs.com/TomXu/archive/2012/04/09/2379774.html)

**介绍**

享元模式（Flyweight），运行共享技术有效地支持大量细粒度的对象，避免大量拥有相同内容的小类的开销(如耗费内存)，使大家共享一个类(元类)。

享元模式可以避免大量非常相似类的开销，在程序设计中，有时需要生产大量细粒度的类实例来表示数据，如果能发现这些实例除了几个参数以外，开销基本相同的 话，就可以大幅度较少需要实例化的类的数量。如果能把那些参数移动到类实例的外面，在方法调用的时候将他们传递进来，就可以通过共享大幅度第减少单个实例 的数目。

那么如果在JavaScript中应用享元模式呢？有两种方式，第一种是应用在数据层上，主要是应用在内存里大量相似的对象上；第二种是应用在DOM层上，享元可以用在中央事件管理器上用来避免给父容器里的每个子元素都附加事件句柄。

**享元与数据层**

Flyweight中有两个重要概念--内部状态intrinsic和外部状态extrinsic之分，内部状态就是在对象里通过内部方法管理，而外部信息可以在通过外部删除或者保存。

说白点,就是先捏一个的原始模型，然后随着不同场合和环境,再产生各具特征的具体模型，很显然,在这里需要产生不同的新对象，所以Flyweight模式中常出现Factory模式，Flyweight的内部状态是用来共享的，Flyweight factory负责维护一个Flyweight pool(模式池)来存放内部状态的对象。

**使用享元模式**

让我们来演示一下如果通过一个类库让系统来管理所有的书籍，每个书籍的元数据暂定为如下内容：

ID  
Title  
Author  
Genre  
Page count  
Publisher ID  
ISBN

我们还需要定义每本书被借出去的时间和借书人，以及退书日期和是否可用状态：

checkoutDate  
checkoutMember  
dueReturnDate  
availability

因为book对象设置成如下代码，注意该代码还未被优化：

var Book = function( id, title, author, genre, pageCount,publisherID, ISBN, checkoutDate, checkoutMember, dueReturnDate,availability ){  
 this.id = id;  
 this.title = title;  
 this.author = author;  
 this.genre = genre;  
 this.pageCount = pageCount;  
 this.publisherID = publisherID;  
 this.ISBN = ISBN;  
 this.checkoutDate = checkoutDate;  
 this.checkoutMember = checkoutMember;  
 this.dueReturnDate = dueReturnDate;  
 this.availability = availability;  
};  
Book.prototype = {  
 getTitle:function(){  
 return this.title;  
 },  
 getAuthor: function(){  
 return this.author;  
 },  
 getISBN: function(){  
 return this.ISBN;  
 },  
/\*其它get方法在这里就不显示了\*/  
  
// 更新借出状态  
updateCheckoutStatus: function(bookID, newStatus, checkoutDate,checkoutMember, newReturnDate){  
 this.id = bookID;  
 this.availability = newStatus;  
 this.checkoutDate = checkoutDate;  
 this.checkoutMember = checkoutMember;  
 this.dueReturnDate = newReturnDate;  
},  
//续借  
extendCheckoutPeriod: function(bookID, newReturnDate){  
 this.id = bookID;  
 this.dueReturnDate = newReturnDate;  
},  
//是否到期  
isPastDue: function(bookID){  
 var currentDate = new Date();  
 return currentDate.getTime() > Date.parse(this.dueReturnDate);  
 }  
};

程序刚开始可能没问题，但是随着时间的增加，图书可能大批量增加，并且每种图书都有不同的版本和数量，你将会发现系统变得越来越慢。几千个book对象在内存里可想而知，我们需要用享元模式来优化。

我们可以将数据分成内部和外部两种数据，和book对象相关的数据（title, author 等）可以归结为内部属性，而（checkoutMember, dueReturnDate等）可以归结为外部属性。这样，如下代码就可以在同一本书里共享同一个对象了，因为不管谁借的书，只要书是同一本书，基本信息是一样的：

/\*享元模式优化代码\*/  
var Book = function(title, author, genre, pageCount, publisherID, ISBN){  
 this.title = title;  
 this.author = author;  
 this.genre = genre;  
 this.pageCount = pageCount;  
 this.publisherID = publisherID;  
 this.ISBN = ISBN;  
};

**定义基本工厂**

让我们来定义一个基本工厂，用来检查之前是否创建该book的对象，如果有就返回，没有就重新创建并存储以便后面可以继续访问，这确保我们为每一种书只创建一个对象：

/\* Book工厂 单例 \*/  
var BookFactory = (function(){  
 var existingBooks = {};  
 return{  
 createBook: function(title, author, genre,pageCount,publisherID,ISBN){  
 /\*查找之前是否创建\*/  
 var existingBook = existingBooks[ISBN];  
 if(existingBook){  
 return existingBook;  
 }else{  
 /\* 如果没有，就创建一个，然后保存\*/  
 var book = new Book(title, author, genre,pageCount,publisherID,ISBN);  
 existingBooks[ISBN] = book;  
 return book;  
 }  
 }  
 }  
});

**管理外部状态**

外部状态，相对就简单了，除了我们封装好的book，其它都需要在这里管理：

/\*BookRecordManager 借书管理类 单例\*/  
var BookRecordManager = (function(){  
 var bookRecordDatabase = {};  
 return{  
 /\*添加借书记录\*/  
 addBookRecord: function(id, title, author, genre,pageCount,publisherID,ISBN, checkoutDate, checkoutMember, dueReturnDate, availability){  
 var book = bookFactory.createBook(title, author, genre,pageCount,publisherID,ISBN);  
 bookRecordDatabase[id] ={  
 checkoutMember: checkoutMember,  
 checkoutDate: checkoutDate,  
 dueReturnDate: dueReturnDate,  
 availability: availability,  
 book: book;  
  
 };  
 },  
 updateCheckoutStatus: function(bookID, newStatus, checkoutDate, checkoutMember, newReturnDate){  
 var record = bookRecordDatabase[bookID];  
 record.availability = newStatus;  
 record.checkoutDate = checkoutDate;  
 record.checkoutMember = checkoutMember;  
 record.dueReturnDate = newReturnDate;  
 },  
 extendCheckoutPeriod: function(bookID, newReturnDate){  
 bookRecordDatabase[bookID].dueReturnDate = newReturnDate;  
 },  
 isPastDue: function(bookID){  
 var currentDate = new Date();  
 return currentDate.getTime() > Date.parse(bookRecordDatabase[bookID].dueReturnDate);  
 }  
 };  
});

通过这种方式，我们做到了将同一种图书的相同信息保存在一个bookmanager对象里，而且只保存一份；相比之前的代码，就可以发现节约了很多内存。

**享元模式与DOM**

关于DOM的事件冒泡，在这里就不多说了，相信大家都已经知道了，我们举两个例子。

**例1：事件集中管理**

举例来说，如果我们又很多相似类型的元素或者结构（比如菜单，或者ul里的多个li）都需要监控他的click事件的话，那就需要多每个元素进行事件绑定，如果元素有非常非常多，那性能就可想而知了，而结合冒泡的知识，任何一个子元素有事件触发的话，那触发以后事件将冒泡到上一级元素，所以利用这个特性，我们可以使用享元模式，我们可以对这些相似元素的父级元素进行事件监控，然后再判断里面哪个子元素有事件触发了，再进行进一步的操作。

在这里我们结合一下jQuery的bind/unbind方法来举例。

**HTML**:

<div id="container">  
 <div class="toggle" href="#">更多信息 (地址)  
 <span class="info">  
 这里是更多信息  
 </span></div>  
 <div class="toggle" href="#">更多信息 (地图)  
 <span class="info">  
 <iframe src="http://www.map-generator.net/extmap.php?name=London&amp;address=london%2C%20england&amp;width=500...gt;"</iframe>  
 </span>  
 </div>  
</div>

**JavaScript**:

stateManager = {  
 fly: function(){  
 var self = this;  
 $('#container').unbind().bind("click", function(e){  
 var target = $(e.originalTarget || e.srcElement);  
 // 判断是哪一个子元素  
 if(target.is("div.toggle")){  
 self.handleClick(target);  
 }  
 });  
 },  
  
 handleClick: function(elem){  
 elem.find('span').toggle('slow');  
 }  
});

**例2：应用享元模式提升性能**

另外一个例子，依然和jQuery有关，一般我们在事件的回调函数里使用元素对象是会后，经常会用到$(this)这种形式，其实它重复创建了新对象，因为本身回调函数里的this已经是DOM元素自身了，我们必要必要使用如下这样的代码：

$('div').bind('click', function(){  
 console.log('You clicked: ' + $(this).attr('id'));  
});  
// 上面的代码，要避免使用，避免再次对DOM元素进行生成jQuery对象，因为这里可以直接使用DOM元素自身了。  
$('div').bind('click', function(){  
 console.log('You clicked: ' + this.id);  
});

其实，如果非要用$(this)这样的形式，我们也可以实现自己版本的单实例模式，比如我们来实现一个jQuery.signle(this)这样的函数以便返回DOM元素自身：

jQuery.single = (function(o){  
  
 var collection = jQuery([1]);  
 return function(element) {  
  
 // 将元素放到集合里  
 collection[0] = element;  
  
 // 返回集合  
 return collection;  
  
 };  
 });

使用方法：

$('div').bind('click', function(){  
 var html = jQuery.single(this).next().html();  
 console.log(html);  
 });

这样，就是原样返回DOM元素自身了，而且不进行jQuery对象的创建。

**总结**

Flyweight模式是一个提高程序效率和性能的模式,会大大加快程序的运行速度.应用场合很多:比如你要从一个数据库中读取一系列字符串,这些字符串中有许多是重复的,那么我们可以将这些字符串储存在Flyweight池(pool)中。

如果一个应用程序使用了大量的对象，而这些大量的对象造成了很大的存储开心时就应该考虑使用享元模式；还有就是对象的大多数状态可以外部状态，如果删除对象的外部状态，那么就可以用相对较少的共享对象取代很多组对象，此时可以考虑使用享元模式。

参考地址：http://www.addyosmani.com/resources/essentialjsdesignpatterns/book/#detailflyweight