**介绍**

在软件系统中，有时候面临着“一个复杂对象”的创建工作，其通常由各个部分的子对象用一定的算法构成；由于需求的变化，这个复杂对象的各个部分经常面临着剧烈的变化，但是将它们组合在一起的算法确相对稳定。如何应对这种变化？如何提供一种“封装机制”来隔离出“复杂对象的各个部分”的变化，从而保持系统中的“稳定构建算法”不随着需求改变而改变？这就是要说的建造者模式。

建造者模式可以将一个复杂对象的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。也就是说如果我们用了建造者模式，那么用户就需要指定需要建造的类型就可以得到它们，而具体建造的过程和细节就不需要知道了。

**正文**

这个模式相对来说比较简单，先上代码，然后再解释

function getBeerById(id, callback) {  
 // 使用ID来请求数据，然后返回数据.  
 asyncRequest('GET', 'beer.uri?id=' + id, function (resp) {  
 // callback调用 response  
 callback(resp.responseText);  
 });  
}  
  
var el = document.querySelector('#test');  
el.addEventListener('click', getBeerByIdBridge, false);  
  
function getBeerByIdBridge(e) {  
 getBeerById(this.id, function (beer) {  
 console.log('Requested Beer: ' + beer);  
 });  
}

根据建造者的定义，表相即是回调，也就是说获取数据以后如何显示和处理取决于回调函数，相应地回调函数在处理数据的时候不需要关注是如何获取数据的，同样的例子也可以在jquery的ajax方法里看到，有很多回调函数（比如success, error回调等），主要目的就是职责分离。

同样再来一个jQuery的例子：

$('<div class= "foo"> bar </div>');

我们只需要传入要生成的HTML字符，而不需要关系具体的HTML对象是如何生产的。

**总结**

建造者模式主要用于“分步骤构建一个复杂的对象”，在这其中“分步骤”是一个稳定的算法，而复杂对象的各个部分则经常变化，其优点是：建造者模式的“加工工艺”是暴露的，这样使得建造者模式更加灵活，并且建造者模式解耦了组装过程和创建具体部件，使得我们不用去关心每个部件是如何组装的。

参考：http://www.cnblogs.com/feipeng/archive/2007/03/12/671932.html