**介绍**

状态模式（State）允许一个对象在其内部状态改变的时候改变它的行为，对象看起来似乎修改了它的类。

**正文**

举个例子，就比如我们平时在下载东西，通常就会有好几个状态，比如准备状态（ReadyState）、下载状态（DownloadingState）、暂停状态（DownloadPausedState）、下载完毕状态（DownloadedState）、失败状态（DownloadFailedState），也就是说在每个状态都只可以做当前状态才可以做的事情，而不能做其它状态能做的事儿。

由于State模式描述了下载（Download）如何在每一种状态下表现出不同的行为。这一模式的关键思想就是引入了一个叫做State的抽象类（或JS里的函数）来表示下载状态，State函数（作为原型）为每个状态的子类（继承函数）声明了一些公共接口。其每个继承函数实现与特定状态相关的行为，比如DownloadingState和DownloadedState分别实现了正在下载和下载完毕的行为。这些行为可以通过Download来来维护。

让我们来实现一把，首先定义作为其他基础函数的原型的State函数：

var State = function () {  
  
};  
  
State.prototype.download = function () {  
 throw new Error("该方法必须被重载!");  
};  
  
State.prototype.pause = function () {  
 throw new Error("该方法必须被重载!");  
};  
  
State.prototype.fail = function () {  
 throw new Error("该方法必须被重载!");  
};  
  
State.prototype.finish = function () {  
 throw new Error("该方法必须被重载!");  
};

我们为State的原型定义了4个方法接口，分别对应着下载（download）、暂停（pause）、失败（fail）、结束（finish）以便子函数可以重写。

在编写子函数之前，我们先来编写一个ReadyState函数，以便可以将状态传递给第一个download状态：

var ReadyState = function (oDownload) {  
 State.apply(this);  
 this.oDownload = oDownload;  
};  
  
ReadyState.prototype = new State();  
  
ReadyState.prototype.download = function () {  
 this.oDownload.setState(this.oDownload.getDownloadingState());  
 // Ready以后，可以开始下载，所以设置了Download函数里的状态获取方法  
 console.log("Start Download!");  
};  
  
ReadyState.prototype.pause = function () {  
 throw new Error("还没开始下载，不能暂停!");  
};  
  
ReadyState.prototype.fail = function () {  
 throw new Error("文件还没开始下载，怎么能说失败呢!");  
};  
  
ReadyState.prototype.finish = function () {  
 throw new Error("文件还没开始下载，当然也不能结束了!");  
};

该函数接收了一个Download维护函数的实例作为参数，Download函数用于控制状态的改变和获取（类似于中央控制器，让外部调用），ReadyState重写了原型的download方法，以便开始进行下载。我们继续来看Download函数的主要功能：

var Download = function () {  
 this.oState = new ReadyState(this);  
};  
  
Download.prototype.setState = function (oState) {  
 this.oState = oState;  
};  
  
// 对外暴露的四个公共方法，以便外部调用  
  
Download.prototype.download = function () {  
 this.oState.download();  
};  
  
Download.prototype.pause = function () {  
 this.oState.pause();  
};  
  
Download.prototype.fail = function () {  
 this.oState.fail();  
};  
  
Download.prototype.finish = function () {  
 this.oState.finish();  
};  
  
//获取各种状态，传入当前this对象  
Download.prototype.getReadyState = function () {  
 return new ReadyState(this);  
};  
  
Download.prototype.getDownloadingState = function () {  
 return new DownloadingState(this);  
};  
  
Download.prototype.getDownloadPausedState = function () {  
 return new DownloadPausedState(this);  
};  
  
Download.prototype.getDownloadedState = function () {  
 return new DownloadedState(this);  
};  
  
Download.prototype.getDownloadedFailedState = function () {  
 return new DownloadFailedState(this);  
};

Download函数的原型提供了8个方法，4个是对用于下载状态的操作行为，另外4个是用于获取当前四个不同的状态，这4个方法都接收this作为参数，也就是将Download实例自身作为一个参数传递给处理该请求的状态对象（ReadyState 以及后面要实现的继承函数），这使得状态对象比必要的时候可以访问oDownlaod。

接下来，继续定义4个相关状态的函数：

var DownloadingState = function (oDownload) {  
 State.apply(this);  
 this.oDownload = oDownload;  
};  
  
DownloadingState.prototype = new State();  
  
DownloadingState.prototype.download = function () {  
 throw new Error("文件已经正在下载中了!");  
};  
  
DownloadingState.prototype.pause = function () { this.oDownload.setState(this.oDownload.getDownloadPausedState());  
 console.log("暂停下载!");  
};  
  
DownloadingState.prototype.fail = function () { this.oDownload.setState(this.oDownload.getDownloadedFailedState());  
 console.log("下载失败!");  
};  
  
DownloadingState.prototype.finish = function () {  
 this.oDownload.setState(this.oDownload.getDownloadedState());  
 console.log("下载完毕!");  
};

DownloadingState的主要注意事项就是已经正在下载的文件，不能再次开始下载了，其它的状态都可以连续进行。

var DownloadPausedState = function (oDownload) {  
 State.apply(this);  
 this.oDownload = oDownload;  
};  
  
DownloadPausedState.prototype = new State();  
  
DownloadPausedState.prototype.download = function () {  
 this.oDownload.setState(this.oDownload.getDownloadingState());  
 console.log("继续下载!");  
};  
  
DownloadPausedState.prototype.pause = function () {  
 throw new Error("已经暂停了，咋还要暂停呢!");  
};  
  
DownloadPausedState.prototype.fail = function () { this.oDownload.setState(this.oDownload.getDownloadedFailedState());  
 console.log("下载失败!");  
};  
  
DownloadPausedState.prototype.finish = function () {  
 this.oDownload.setState(this.oDownload.getDownloadedState());  
 console.log("下载完毕!");  
};

DownloadPausedState函数里要注意的是，已经暂停的下载，不能再次暂停。

var DownloadedState = function (oDownload) {  
 State.apply(this);  
 this.oDownload = oDownload;  
};  
  
DownloadedState.prototype = new State();  
  
DownloadedState.prototype.download = function () {  
 this.oDownload.setState(this.oDownload.getDownloadingState());  
 console.log("重新下载!");  
};  
  
DownloadedState.prototype.pause = function () {  
 throw new Error("对下载完了，还暂停啥？");  
};  
  
DownloadedState.prototype.fail = function () {  
 throw new Error("都下载成功了，咋会失败呢？");  
};  
  
DownloadedState.prototype.finish = function () {  
 throw new Error("下载成功了，不能再为成功了吧!");  
};

DownloadedState函数，同理成功下载以后，不能再设置finish了，只能设置重新下载状态。

var DownloadFailedState = function (oDownload) {  
 State.apply(this);  
 this.oDownload = oDownload;  
};  
  
DownloadFailedState.prototype = new State();  
  
DownloadFailedState.prototype.download = function () {  
 this.oDownload.setState(this.oDownload.getDownloadingState());  
 console.log("尝试重新下载!");  
};  
  
DownloadFailedState.prototype.pause = function () {  
 throw new Error("失败的下载，也不能暂停!");  
};  
  
DownloadFailedState.prototype.fail = function () {  
 throw new Error("都失败了，咋还失败呢!");  
};  
  
DownloadFailedState.prototype.finish = function () {  
 throw new Error("失败的下载，肯定也不会成功!");  
};同理，DownloadFailedState函数的失败状态，也不能再次失败，但可以和finished以后再次尝试重新下载。

调用测试代码，就非常简单了，我们在HTML里演示吧，首先是要了jquery，然后有3个按钮分别代表：开始下载、暂停、重新下载。（注意在Firefox里用firebug查看结果，因为用了 console.log方法）。

<html>  
<head>  
 <link type="text/css" rel="stylesheet" href="http://www.cnblogs.com/css/style.css" />  
 <title>State Pattern</title>  
 <script type="text/javascript" src="/jquery.js"></script>  
 <script type="text/javascript" src="Download.js"></script>  
 <script type="text/javascript" src="states/State.js"></script>  
 <script type="text/javascript" src="states/DownloadFailedState.js"></script>  
 <script type="text/javascript" src="states/DownloadPausedState.js"></script>  
 <script type="text/javascript" src="states/DownloadedState.js"></script>  
 <script type="text/javascript" src="states/DownloadingState.js"></script>  
 <script type="text/javascript" src="states/ReadyState.js"></script>  
</head>  
<body>  
 <input type="button" value="开始下载" id="download\_button" />  
 <input type="button" value="暂停" id="pause\_button" />  
 <input type="button" value="重新下载" id="resume\_button" />  
 <script type="text/javascript">  
 var oDownload = new Download();  
 $("#download\_button").click(function () {  
 oDownload.download();  
 });  
  
 $("#pause\_button").click(function () {  
 oDownload.pause();  
 });  
  
 $("#resume\_button").click(function () {  
 oDownload.download();  
 });  
 </script>  
</body>  
</html>

**总结**

状态模式的使用场景也特别明确，有如下两点：

1. 一个对象的行为取决于它的状态，并且它必须在运行时刻根据状态改变它的行为。
2. 一个操作中含有大量的分支语句，而且这些分支语句依赖于该对象的状态。状态通常为一个或多个枚举常量的表示。

参考：https://github.com/tcorral/Design-Patterns-in-Javascript/blob/master/State/1/index.html