

# 基于 Cairngorm 的 Flex 应用程序设计

TRANSLATED BY

Dreamer

1. 本文档的英文版版权归[www.digimmersion.com](http://www.digimmersion.com)所有
2. 文档中文翻译已经得到[www.digimmersion.com](http://www.digimmersion.com)的授权
3. 译文版权归Dreamer ([www.zhuoqun.net](http://www.zhuoqun.net)) 所有, 未经同意, 谢绝转载
4. 此文档系个人作品, 仅供学习交流之用, 可自由传播和打印, 但不得用于商业用途
5. 个人水平所限, 如果发现翻译有误, 烦请告知译者 (E-mail: [dreamer@zhuoqun.net](mailto:dreamer@zhuoqun.net))

## 概述

这个文档将会使开发者了解一个简单的基于Cairngorm 的Flex程序的设计过程，本程序展示了现实中的一个业务需求。本文档的目标读者是这样的开发者：他至少有点熟悉Flex，开发过一些Flex应用程序，有点熟悉Cairngorm 微型架构，并且希望学习更多关于如何从一开始就正确合理地设计一个Cairngorm 应用程序。

尽管本文档展示了可视化模型并推荐使用Digimmersion Flex 2 RIA Visio stencil，但是我们并没有打算将其作为单独的产品销售；进一步来说，考虑到Cairngorm 应用程序的快速开发，我们打算将此文档用来传授设计的实践。很显然，我们鼓励使用Digimmersion Visio stencil，但是当然也存在其他方式来创建可视化模型。不管你选择什么工具来创建模型，预先可视化程序都是程序开发的一个重要部分。

如果你对此文档或者文档中所讨论的任何主题有问题，你可以通过Digimmersion上的 Contact Us 页面来联系到作者。如果你对 Cairngorm , Flex 或者 ActionScript 有特别的或者细节方面的问题，你可以加入 FlexCoders Yahoo! Group 或者访问 Labs.Adobe.com 上的 Cairngorm Wiki。

## 业务问题

这个现实中的业务问题是目录服务（directory services）中的。多数大公司都会维护一个 Active、Novell 或者 LDAP 目录，里面包含了该网络的用户帐号。我的“白天的工作”是为一个机构工作，它拥有超过 6000 个雇员和超过 2000 个雇员，而且所有人都需要访问我们的网络。网络帐号被集中在我们的 Novell eDirectory (eDir)，它也可以通过 LDAP 访问到。

雇员帐号被放在我们的Employee容器中并通过dirXML驱动连接到PeopleSoft HR 系统，这样一来PeopleSoft中的诸如“请假”和“终止”等雇佣状态的更改就会自动在eDir中更新，而且帐号的特权也会适当地被激活或者被禁止。Affiliates 容器中包含了那些没有在PeopleSoft系统中的用户的帐号。下属办公室职员，志愿者，卖主以及其他任何人都可能需要访问网络，但是这并不是由我们的HR部门管理或者控制。

由于子公司控制他们自己的雇佣和解雇，而且我们的eDir没有连接到任何外部的HR系统，是手工操作的，所以当外部组织中的职工发生变化时eDir并不总能适当

地更新。当一个新的雇佣需要一个新的网络帐号的时候它们可以很快通知我们，但是当某人离开了他们的机构的时候并不能及时通知我们。这里的这个过程并不能自动处理，所以那些不再为外部子公司工作的人在eDir中依然会有活动的帐号。那些没有除去的帐号不只是安全隐患，而且它们还增加了eDir的大小，这就增加了开销并且需要额外的管理资源。

## 解决方案

创建一个由两部分组成的应用程序：

- 第一部分：一个计划任务（可执行的网页、服务等），每天晚上运行，遍历我们的 eDirectory 树并收集有关问题帐号、部门经理以及最后登录日期/时间等的信息，然后将这些信息储存在一个小型的简单 SQL 数据库中。
- 第二部分：一个用来与数据库交互的基于网页的客户端，有向经理报告问题帐号的功能，并且可以让经理为每个问题账号指定一个状态（请假，度假，终止等）。

当下属公司无法向我们通知它们的职员状态变动时，这个应用程序会提供一个安全网络。

既然我们想要让客户端可以被远程访问，那么就需要一个基于网页的解决方案，而这个客户端的技术我们选择使用 Adobe Flex。

## 软件需求描述

第一件事情就是要确定程序将要做什么以及如何工作。你和你的公司可以遵循任意的标准需求收集步骤来确定这个程序的需求，但是为了我们的目的我们需要使其保持简单。我们的程序将会依赖一个简单的数据库，而这个数据库每晚会自动收集信息。在这里我们不过多地讨论它是如何自动运行的，而只是把需求本身和实际客户端的界面联系起来：

- 登录功能
- 两种会用到的用户角色：Manager 和 Admin
- 所有的用户都是Manager

- 某些用户可以是Administrator
- Administrators可以……
  - 查看所有问题帐号
  - 选择查看那些还没有指派给Manager的问题帐号
  - 选中一个或多个帐号
  - 指派一个选定的帐号给一个Manager
  - 对已指派给Manager的帐号取消当前指派
  - 切换为Manager身份
- Manager可以……
  - 查看所有已指派给该Manager的帐号
  - 选择查看还没有指派状态的帐号
  - 选中一个或多个问题帐号
  - 选中一个状态
  - 为某些的状态选择一个回返日期
  - 为选中的帐号指派一个状态（以及日期，如果适用的话）
  - 取消选定帐号的状态指派
  - 切换为Admin身份，如果拥有Admin特权的话
- 对所有界面功能的帮助和用法说明
- 退出功能

## Cairngorm 概览

虽然本文档尝试涵盖如何设计一个Cairngorm应用程序的基础，但是并不会深入讲述框架的一些细节。更多信息和代码示例，你可以看Steven Webster的blog、Adobe Labs中的Cairngorm Wiki或者Yahoo! FlexCoders group。

什么是Cairngorm? Cairngorm从根本上来说是将程序代码按照逻辑功能（按照数据、用户视图、以及起控制作用的代码）分块的一种方法论。这个方法论被归纳为MVC，或者说是Model（模型），View（视图），和Control（控制）。

### Cairngorm的各部分

- **Model Locator:** 在一个地方存储程序中所有的值对象（Value Objects, 数据）并共享变量。它与HTTP Session对象很相似，不过它存储在Flex客户端，而不是存储在一个中间层程序服务器的服务器端。
- **View（视图）:** 一个或多个Flex组件（button, panel, combo box, Tile 等等）绑定到一起形成的一个特定的个体，使用Model Locator中的数据，并且针对用户的交互动作（点击，鼠标滑过，拖拽等）产生自定义的Cairngorm Events。

- **Front Controller**（前端控制器）：接收Cairngorm Events并且将它们映射到Cairngorm Commands。
- **Command**（命令）：处理业务逻辑，调用Cairngorm Delegates 和/或 其它的Commands，以及更新Model Locator中存储的值对象和变量。
- **Delegate**（委托）：由一个Command创建，它将远程过程调用（HTTP，Web Services等）实例化并且将结果返回给该Command。
- **Service**（服务）：定义连接到远程数据库的远程过程调用（HTTP，Web Services等）

## 各部分如何组织在一起

Cairngorm的工作流程大体上是这样：客户端界面是由View（视图）组成的。View使用Flex的binding（绑定）来显示Model Locator中包含的数据。View根据诸如鼠标点击，按钮按下以及拖拽之类的用户动作产生Event。这些Event被Front Controller“广播”并“监听”，Front Controller会将Event映射到Command。Command包括业务逻辑，创建所需Delegate，处理Delegate的相应，以及更新存储在Model Locator中的数据。由于View是绑定到Model Locator中的数据上的，所以当Model Locator中的数据改变的时候View也会自动更新。Delegate调用Service并且将结果提交给Command，这一步是可选的，但是推荐这么做。Service调用远程数据然后将结果提交给Delegate。

下一页的图表给出了一个Cairngorm应用程序处理流程的一个概貌。（译注：该图表可以在这里下载：[http://flexheads.com/cairngorm/cairngorm2\\_rpc.swf](http://flexheads.com/cairngorm/cairngorm2_rpc.swf)）

## Cairngorm 2 Microarchitecture – Server RPC

From 'Developing Flex RIAs with Cairngorm Microarchitecture' by Steven Webster  
Copyright: Adobe.com - [http://www.adobe.com/devnet/flex/articles/cairngorm\\_pt1.html](http://www.adobe.com/devnet/flex/articles/cairngorm_pt1.html)

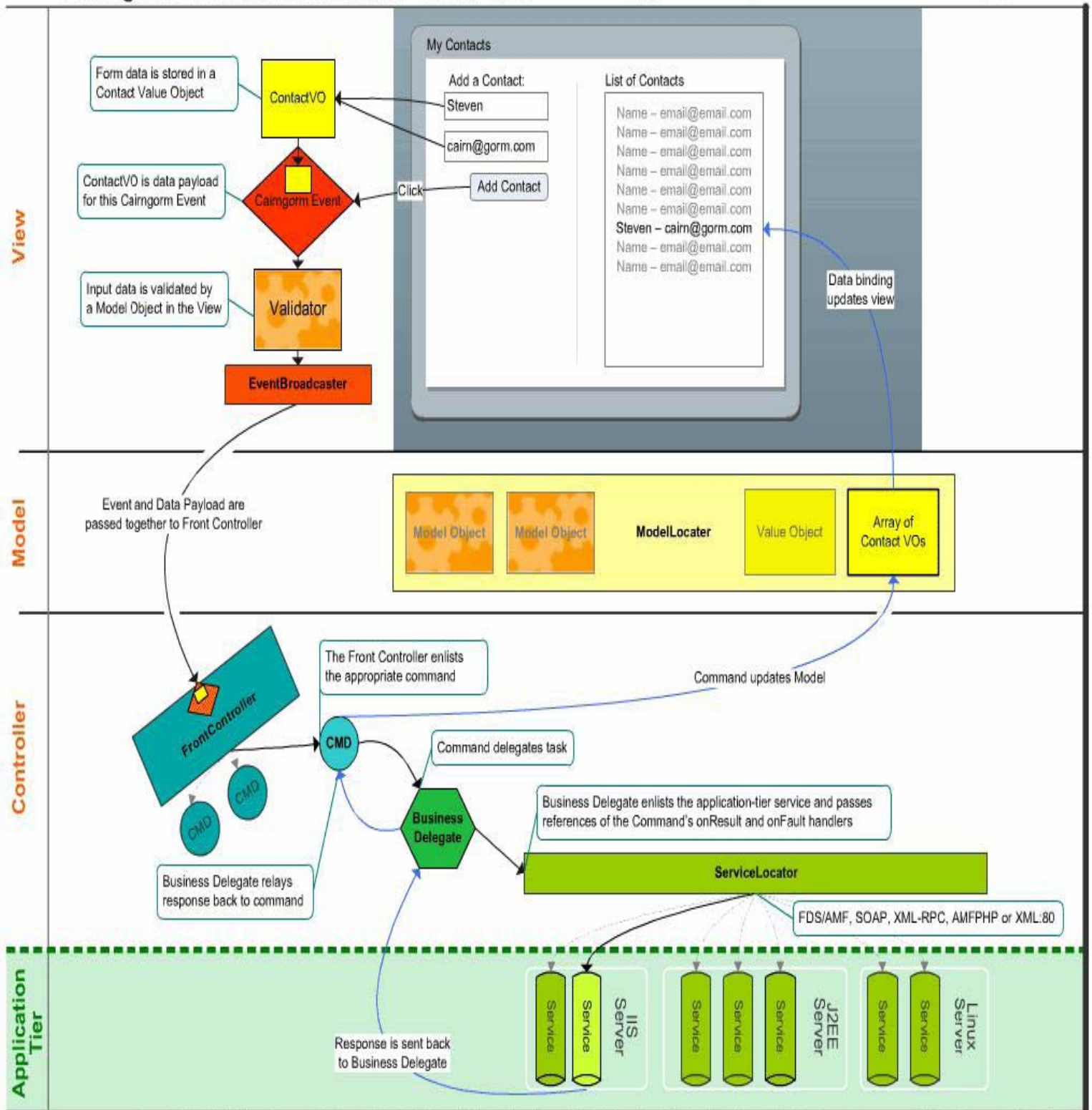


diagram by Evan Gifford (evan@usa.com) with special thanks to Steven Webster, Alex Uhlmann, Russell Munro, Jesse Warden, Dan Nielsen, Tom Chiverton and everyone on the flexcoders group – flexcoders.org



立足于Cairngorm的这几部分，现在我们对如何创建我们的程序应该有些了解了。

- 首先我们要了解我们的数据
- 我们将使用数据来定义View
- 我们将使用View来定义可能的用户动作
- 用户动作将会转化（转变）为Cairngorm Events。
- Event将会被映射到Command
- Command可能会被映射到Delegate
- Delegate映射到Service
- Command将使用Service传回的结果更新Model Locator

## 通过模型来了解Cairngorm的各个组件

我们的View需要显示夜间收集的数据，所以至少需要定义一个小的数据集并以此开始然后才能知道如何处理View。由于Cairngorm是模块化的，所以我们在发现需要跟踪并存储的数据的时候，总是可以向数据模型中添加。我们将以Model Locator里的一些简单数据开始。

### Cairngorm: Model Locator

在一些系统的某个地方我们会创建一个计划任务，它会自动遍历eDir树查找那些在一定时间内从未登录过的还在活动的被激活的用户帐号，我们将这些帐号叫做问题帐号；我们也会遍历这棵树来收集经理的帐号，我们将这些帐号叫做经理帐号。我们的程序需要显示这两种帐号，但是是在不同的View里显示。在收集完毕之后我们将把所有这些帐号存储在Model Locator里。

我们可以尝试着创建一个通用的AcctVO值对象来保存任一种类型的帐号，但是这两种帐号很不一样所以我们将使用两种不同类型的值对象：QAcctVO代表问题帐号，MAcctVO代表经理帐号。每一种帐号类型都有多个帐号，所以我们将这些帐号存储在两个列表里，一个存储经理帐号——MVOList，另一个QVOList存储问题帐号。这两个列表将会在Model Locator里定义。View将会绑定到这两个列表中的数据上。在程序中我们还需要跟踪其他的信息，比如登录信息和一些状态信息，所以我们将创建一个LoginVO对象来跟踪用户的信息，而且程序状态将会作为公有变量存储。如果程序很大很复杂我们可能会将程序状态和常量存储到它们自己的数组中或者值对象中来保持其组织性，但是在现在这种情况下我们只需要简单地创建一些相互独立的公有变量和常量。

\*注意：正因如此这里没有混淆，使用“状态”这个词表示比如“已登录”或“已初始化”等变量值，而不是<mx:State>标签的使用。对于一个或多个ViewStack，程序将主要比如selectedIndex使用状态变量来控制View。

## Cairngorm: Views

View可以以多种方式分解：通过要显示的数据、执行的函数、用户角色、可用性、状态等等。当设计用户界面的时候你会发现RIA几乎提供了无限的可能，然而我们不可能使用到Flex提供的的每一种用户设计模式、可用性、状态、动画、效果等等。这里为了保持程序的简单性，我们将同时通过要显示的数据以及状态来分解View，这里的状态是指View的状态，问题帐号需要一个View，但是Manager的问题帐号的View和Administrator的问题帐号的View有些不一样，所以这个View有两种状态。

由于我们想要保持敏捷性，所以这里对于如何分解View写了一个粗略的草稿，在以后的设计中我们会细化它……

- **LoginView** - 一个接受用户验证信息的对话框或者表格。
- **ControlView** - 一个允许用户在Manager角色和Admin角色之间来回切换（取决于特权）的菜单并提供退出或关闭的功能。
- **QAacctView** - 一个可以列出所有问题帐号的DataGrid。无论什么角色它都会显示相同的列，但是Administrator与Manager过滤DataGrid的方式会有所不同。
- **AssignmentView** - 在这里为问题帐号指定一个管理员或状态。当用户是一个Administrator的时候，这个View会允许他们指派给某个Manager一个或多个帐号；当用户是一个Manager的时候，这个View会允许它们将某个状态指派给一个或多个帐号。
- **HelpView** - 一个显示帮助文档的单独的区域。因为这个程序可能被组织中的任何人使用，使用者可能没有电脑经验，所以当用户将鼠标放在不同界面上的时候，一定要自动显示间接明了的帮助信息。

## 模型（Mockups）

正如你看到的那样，我们已经口头描述了一下程序界面，但是并不是全部都很清楚，而且没有详细到可以编码的程度。我们需要一些可以快速创建的可视化模块，下面列出了原因：

- 尽管这不是一个特别复杂的程序，但是如果现在就开始编码我们将不可避免地遗漏一些东西或者不了解我们所需要的东西。如果我们需要回过头去添加或者替换大量代码甚至是程序的某一部分的话，那才真的是浪费时间。敏捷是好的，但是一开始的时候做一点计划也是聪明的做法。
- Flex程序支持多层嵌套组件和布局；例如，一个Panel里有一个VBox，这



个VBox里有一个HBox，HBox里又有另外一个Panel，这个Panel里有一个Canvas，如此类推。如果只是口头描述程序，我们的开发过程很有可能是编码，测试，再编码，测试……尝试50次来获得我们想要的合理的完全符合程序需求分析的Flex布局。模型在这里可以帮我们节省时间。

- 如果你是为一个客户设计这个程序，你一定想可以给他们展示一些可以看得到的东西，至少展示一点程序中令人兴奋的东西。

它们不需要是100%正确的，也不需要和最终的程序绝对一样（除非你想要它们那样），但是它们可以提供可以作为出发点的一个坚实的基础。

模型可以用许多工具创建，但是这里使用的是Digimmersion Flex 2.0 RIA stencil for Visio，它被开发用来填补我们自己程序开发中的巨大空白。它包括所有Flex 2.0对象的模板，还包括其它几种可以保证你的模型尽可能地接近最终Flex程序的工具。

下面的模型就是用这种模板创建的，随之列出的是Model Locator的状态变量，这些变量决定着View的状态。

**LoginView** - 一个接受用户验证信息的对话框或者表格

当...

ModelLocator.LoginState = “Not Authenticated”  
的时候，这个View将会可见

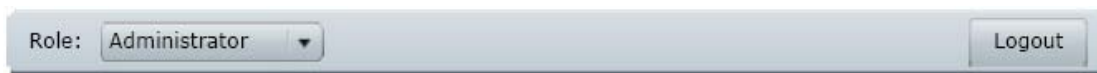
A screenshot of a login dialog box. It has a light blue border and a white background. Inside, there are two text input fields. The first field is labeled "Username:" and the second field is labeled "Password:". Below the password field, there is a button labeled "Login".

**ControlView** - 一个允许用户在Manager角色和Admin角色之间来回切换(取决于特权)的菜单并提供退出或关闭的功能。

```
ModelLocator.LoginState = "Authenticated"
```

另外,我们将跟踪Model Locator中的用户角色并使用它来改变View的外观或者状态。View中的大部分都不会改变,只会改变View中某些小细节。

```
ModelLocator.RoleState = "Administrator"
```



```
ModelLocator.RoleState = "Manager"
```



\* Administrator将会在他们的Role下拉列表框中同时看到“Administrator”和“Manager”选项。Manager将只能看到“Manager”选项,除非它们拥有Administrator的特权。

正如所看到的那样,这只是一个View, ControlView,但是其中的下拉列表框会随着ModelLocator.RoleState更改。如果你发现一个View需要在状态之间做巨大改变,那么你就需要考虑将你的View分解成两个完全不同的View。在这里,状态之间的改变非常小(没有添加/移除控件或者布局),所以它还是一个View。

**QAacctView** - 一个可以列出所有问题帐号的DataGrid。无论什么角色它都会显示相同的列，但是Administrator与Manager过滤DataGrid的方式会有所不同。

```
ModelLocator.LoginState = "Authenticated"  
ModelLocator.RoleState = "Administrator"  
ModelLocator.FilterState = "Filtered"
```

Accounts						
User ID ▼	Name	Job Title	Company	Department	Phone	Last Login ▲
JDoe	Doe, Jane	Secretary	Big Co.	Marketing	555-1234	2/7/2005

☒ Display only unassigned accounts Questionable Accounts: 1

```
ModelLocator.LoginState = "Authenticated"  
ModelLocator.RoleState = "Administrator"  
ModelLocator.FilterState = "Unfiltered"
```

Accounts						
User ID ▼	Name	Job Title	Company	Department	Phone	Last Login ▲
JDoe	Doe, Jane	Secretary	Big Co.	Marketing	555-1234	2/7/2005
JSmith	Smith, John	Clerk	Big Co.	Shipping	555-4321	1/1/2004

☐ Display only unassigned accounts Questionable Accounts: 2

```
ModelLocator.LoginState = "Authenticated"
```

```
ModelLocator.RoleState = "Manager"
```

```
ModelLocator.FilterState = "Filtered"
```

Accounts

User ID ▼	Name	Job Title	Company	Department	Phone	Last Login
JSmith	Smith, John	Clerk	Big Co.	Shipping	555-4321	1/1/2004

☒ Display accounts awaiting status Questionable Accounts: 1

```
ModelLocator.LoginState = "Authenticated"
```

```
ModelLocator.RoleState = "Manager"
```

```
ModelLocator.FilterState = "Unfiltered"
```

Accounts

User ID ▼	Name	Job Title	Company	Department	Phone	Last Login
JSmith	Smith, John	Clerk	Big Co.	Shipping	555-4321	1/1/2004
LFine	Fine, Larry	Clerk	Big Co.	Shipping	555-1122	2/2/2005
MHoward	Howard, Moe	Clerk	Big Co.	Shipping	555-3344	3/3/2005

☐ Display accounts awaiting status Questionable Accounts: 3

\* 你可以看到这个View在每种情况下都是一样的，但是由于状态的改变（RoleState, FilterState），View中显示的数据也会改变，过滤条件也随之改变。再说一次，View中一些细小的改变取决于状态变量的值。

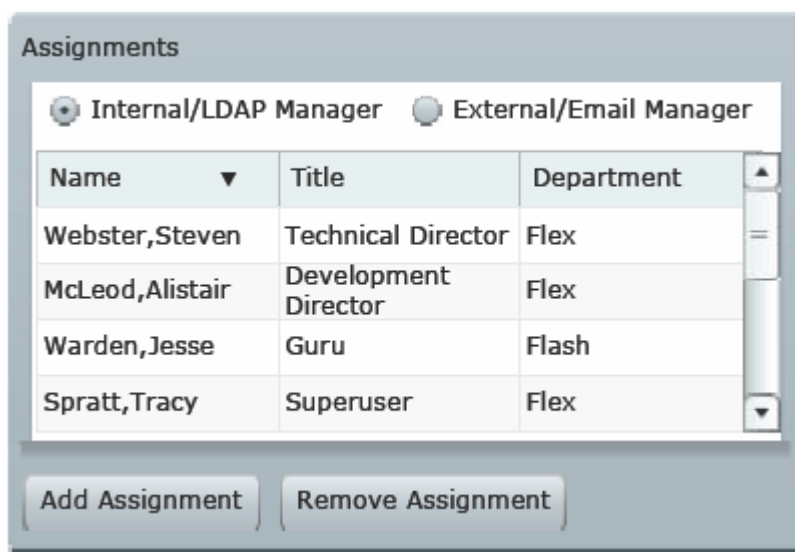
**AssignmentView** - 在这里为问题帐号指定一个管理员或状态。当用户是一个 Administrator 的时候，这个 View 会允许他们指派给某个 Manager 一个或多个帐号；当用户是一个 Manager 的时候，这个 View 会允许它们将某个状态指派给一个或多个帐号。

AssignmentView 有多种状态，这就导致它将显示有多种不同的布局。这个 View 可以被分解为多个单独的 View，例如通过角色来分解，但是在这里我们将会将其作为一个中等复杂的 View，它根据角色、经理的种类、离开状态等来发生改变。

```
ModelLocator.LoginState = "Authenticated"
```

```
ModelLocator.RoleState = "Administrator"
```

```
ModelLocator.ManagerState = "Internal"
```



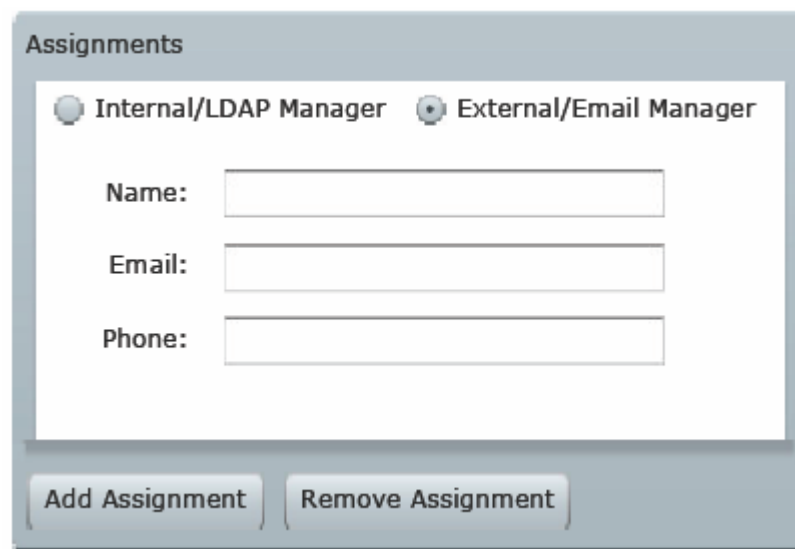
Name ▼	Title	Department
Webster, Steven	Technical Director	Flex
McLeod, Alistair	Development Director	Flex
Warden, Jesse	Guru	Flash
Spratt, Tracy	Superuser	Flex

☒ Internal/LDAP Manager ☐ External/Email Manager

```
ModelLocator.LoginState = "Authenticated"
```

```
ModelLocator.RoleState = "Administrator"
```

```
ModelLocator.ManagerState = "External"
```



☐ Internal/LDAP Manager ☒ External/Email Manager

Name:

Email:

Phone:



ModelLocator.LoginState = "Authenticated"

ModelLocator.RoleState = "Manager"

ModelLocator.EmployeeState = "Terminated"

The screenshot shows a window titled "Assignments". Inside, there are two radio buttons: "Terminated" (which is selected) and "Active". Below these buttons are two more buttons: "Add Assignment" and "Remove Assignment".

ModelLocator.LoginState = "Authenticated"

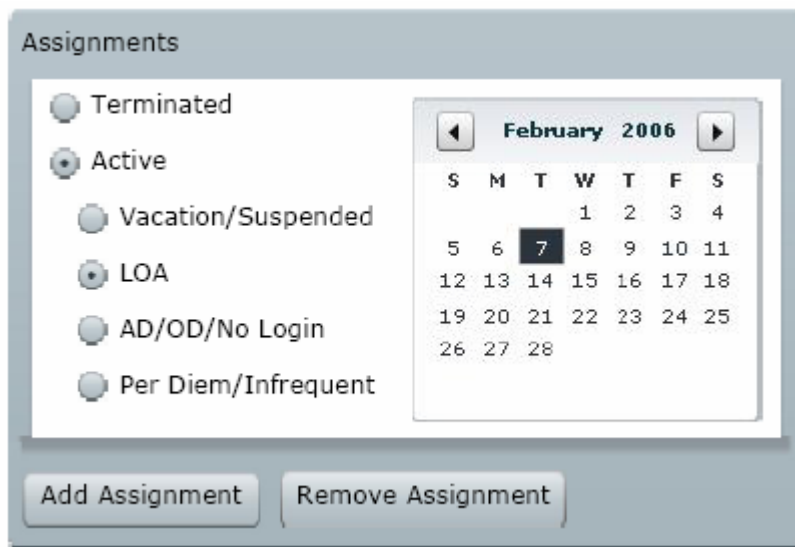
ModelLocator.RoleState = "Manager"

ModelLocator.EmployeeState = "Active"

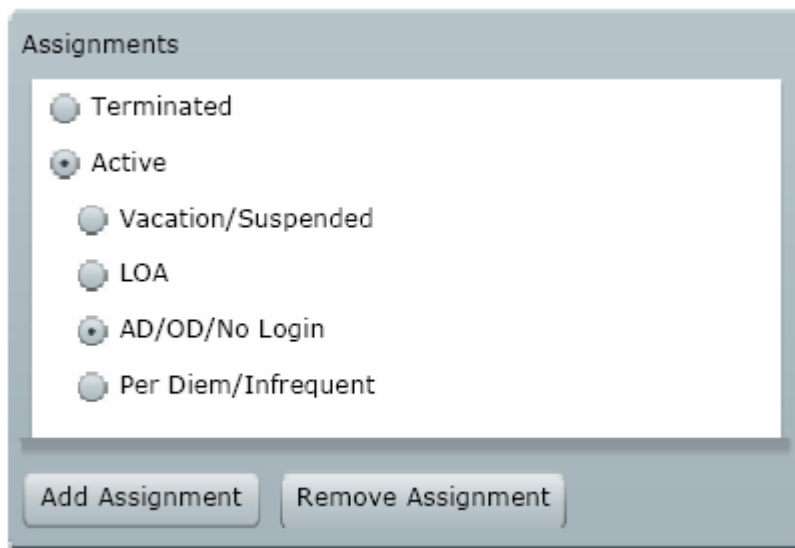
ModelLocator.ActiveState = "Vacation / Suspended"

The screenshot shows a window titled "Assignments". Inside, there are five radio buttons: "Terminated", "Active" (selected), "Vacation/Suspended", "LOA", and "AD/OD/No Login". Below these is a sixth radio button labeled "Per Diem/Infrequent". To the right of the radio buttons is a calendar for February 2006. The calendar shows the days of the week (S, M, T, W, T, F, S) and the dates. The date 7 is highlighted. Below the calendar are two buttons: "Add Assignment" and "Remove Assignment".

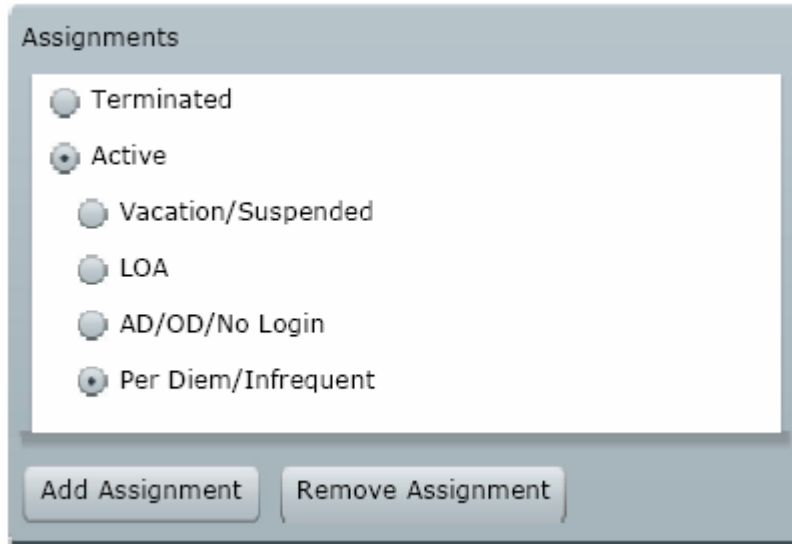
```
ModelLocator.LoginState = "Authenticated"
ModelLocator.RoleState = "Manager"
ModelLocator.EmployeeState = "Active"
ModelLocator.ActiveState = "LOA"
```



```
ModelLocator.LoginState = "Authenticated"
ModelLocator.RoleState = "Manager"
ModelLocator.EmployeeState = "Active"
ModelLocator.ActiveState = "AD / OD / No Login"
```

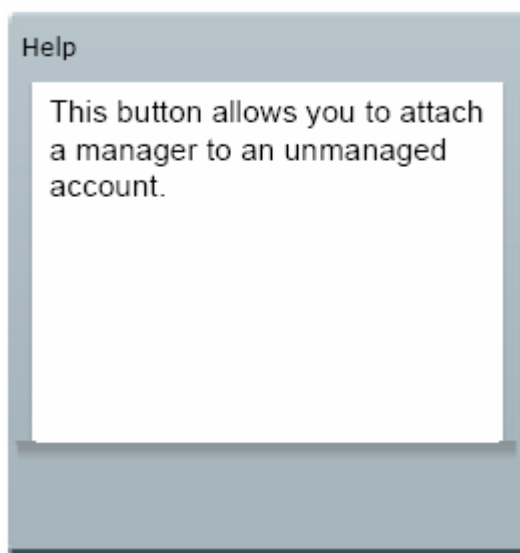


```
ModelLocator.LoginState = "Authenticated"  
ModelLocator.RoleState = "Manager"  
ModelLocator.EmployeeState = "Active"  
ModelLocator.ActiveState = "Per Diem / Infrequent"
```



**HelpView** - 一个显示帮助文档的单独的区域。因为这个程序可能被组织中的任何人使用, 使用者可能没有电脑经验, 所以当用户将鼠标放在不同界面上的时候, 一定要自动显示间接明了的帮助信息。

这个View不依赖于某个状态, 尽管当鼠标滑过不同的区域和组件上面的时候它会显示不同的文本信息。例如, 当鼠标滑过 AssignmentView 中 Add Assignment 按钮时, 可能会显示下面的信息。



\* 由于到现在为止我们只是要简单设计UI, 所以在这里并没有完善每个细节, 而只是专注于用户可以看到的布局以及使用的功能, 这可以让我们确定可能发生的用户动作, 这些动作以后可能要映射到命令。

## 完全组装好的程序

既然我们已经对每个独立的View及其相关的状态都建立了模型,那么针对对整个程序的每个状态建立完全的模型就是一件既多余又浪费时间的事情。但是,为了可以给读者(代码编写者,经理,客户等)一个关于整个程序的全景或者简单的印象,可以针对某个特定状态创建一个整个程序的模型,例如:

```
ModelLocator.LoginState = "Authenticated"  
ModelLocator.RoleState = "Manager"  
ModelLocator.FilterState = "Unfiltered"  
ModelLocator.EmployeeState = "Active"  
ModelLocator.ActiveState = "Per Diem / Infrequent"
```

Role: Manager Logout

Accounts

User ID	Name	Job Title	Company	Department	Phone	Last Login
JSmith	Smith, John	Clerk	Big Co.	Shipping	555-4321	1/1/2004
LFine	Fine, Larry	Clerk	Big Co.	Shipping	555-1122	2/2/2005
MHoward	Howard, Moe	Clerk	Big Co.	Shipping	555-3344	3/3/2005

☐ Display accounts awaiting status Questionable Accounts: 3

Assignments

- ☐ Terminated
- ☒ Active
- ☐ Vacation/Suspended
- ☐ LOA
- ☐ AD/OD/No Login
- ☐ Per Diem/Infrequent

Add Assignment Remove Assignment

Help

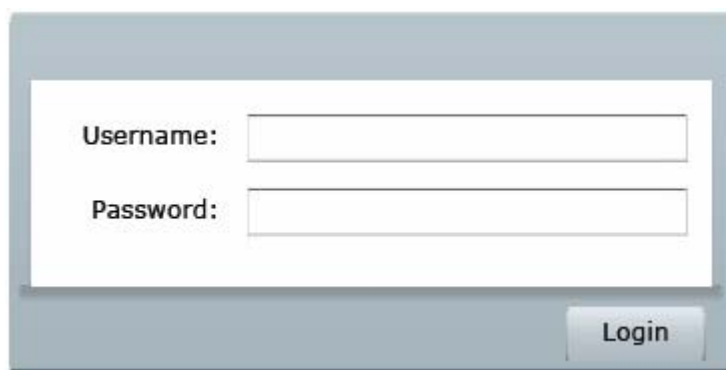
This button allows you to attach a manager to an unmanaged account.

\* 开发到这一步的时候,你可能将在客户面前演示这个模型来得到一些认同,然后再进行下一步的开发。在这个时候要更改模型也非常快速简单,而且不会影响到任何代码。

## Cairngorm: Events

现在我们已经创建好了View, 然后就可以确定用户在每个View的每个状态中可能的行为或动作。我们将列出这些动作并给它们命名, 使其在我们的程序中有意义。例如, 点击“Add Assignment”按钮将会映射为assignManagerEvent。这一步实际上非常简单, 你只需要检查View并且确定用户会有哪些交互, 如果有的话, 就执行一个函数。例如…

### LoginView



在这种情况下, 向文本框中输入不会触发任何Command和, 也不会引起任何处理过程。而另一方面, Login按钮则需要引发一些处理, 所以…

点击Login按钮=” ” 事件被映射…

LoginButton click event → LoginCommand

现在, 针对其它的View做同样的分析。注意这些事件可能是最后你写在代码中的事件, 也可能不是, 但是… 至少它是一个出发点并且可以在一定程度上使程序更加清晰。



## ControlView



RoleComboBox change event → ChangeRoleEvent

RoleComboBox rollover event → DisplayInfoEvent

LogoutButton click event → LogoutEvent

LogoutButton rollover event → DisplayInfoEvent

## QAcctsView



AcctDataGrid click event → SelectAcctEvent

AcctDataGrid rollover event → DisplayInfoEvent

FilterCheckbox change event → ToggleFilterEvent

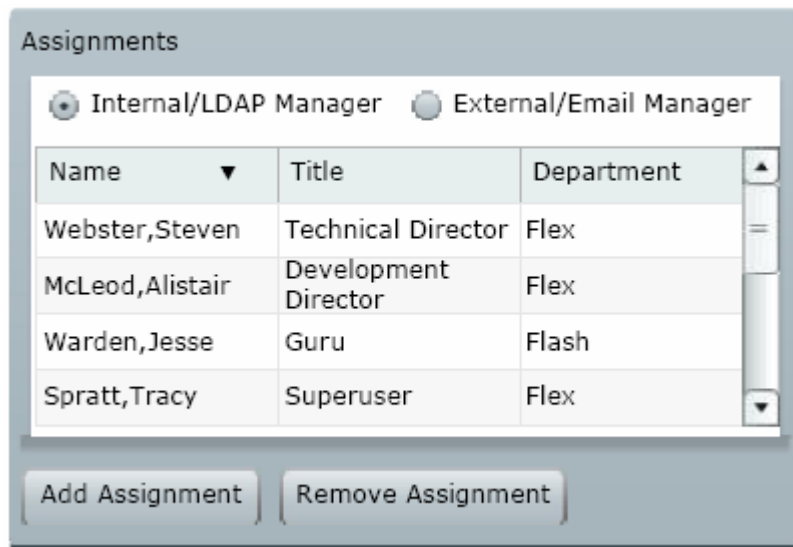
FilterCheckbox rollover event → DisplayInfoEvent

\* 没有必要列出这个View所有状态下的事件，因为改变状态（例如过滤，未过滤）并不会出现新的或者不同的按钮，复选框等，所以也不会出现新的用户动作。如果在不同的状态中会增加/移除组件或者出现不同的用户动作，那么你就需要列出每个状态下的事件了——就像下面的例子一样。

## AssignmentView

ModelLocator.RoleState = “Administrator”

ModelLocator.ManagerState = “Internal”



InternalRadioButton rollover event → DisplayInfoEvent

ExternalRadioButton select event → ToggleIntExtMgrEvent

ExternalRadioButton rollover event → DisplayInfoEvent

ManagerDataGrid select event → SelectManagerEvent

ManagerDataGrid rollover event → DisplayInfoEvent

AddAssignButton click event → AddAssignmentEvent

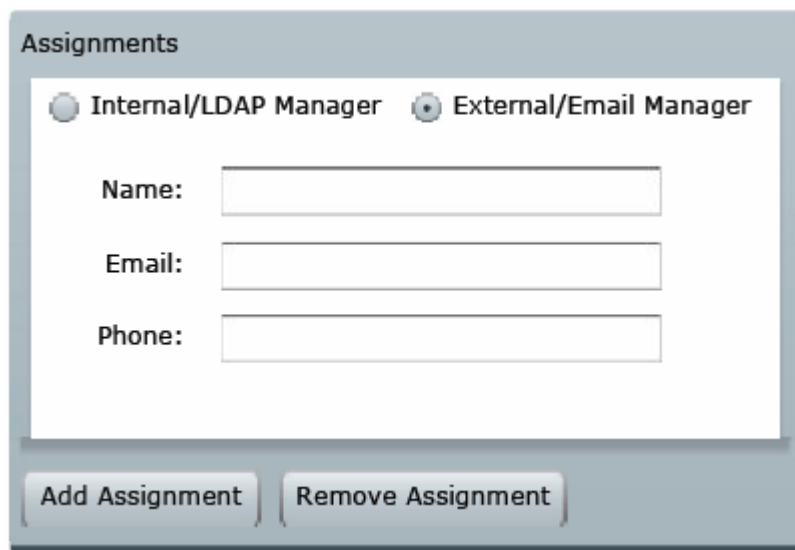
AddAssignButton rollover event → DisplayInfoEvent

RemoveAssignButton click event → RemoveAssignmentEvent

RemoveAssignButton rollover event → DisplayInfoEvent

```
ModelLocator.RoleState = "Administrator"
```

```
ModelLocator.ManagerState = "External"
```



```
ExternalRadioButton rollover event → DisplayInfoEvent
```

```
InternalRadioButton select event → ToggleIntExtMgrEvent
```

```
InternalRadioButton rollover event → DisplayInfoEvent
```

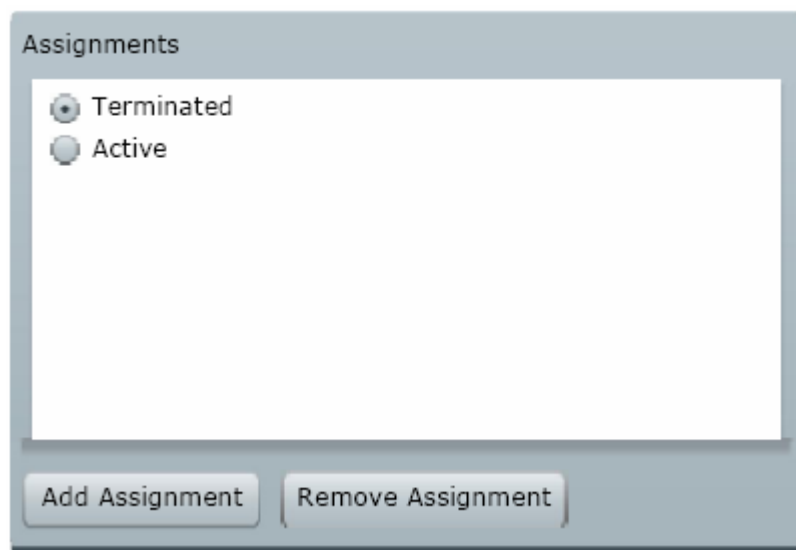
```
AddAssignButton click event → AddAssignmentEvent
```

```
AddAssignButton rollover event → DisplayInfoEvent
```

```
RemoveAssignButton click event → RemoveAssignmentEvent
```

```
RemoveAssignButton rollover event → DisplayInfoEvent
```

```
ModelLocator.RoleState = "Manager"  
ModelLocator.EmployeeState = "Terminated"
```



TerminatedRadioButton rollover event → DisplayInfoEvent

ActiveRadioButton select event → ToggleEmployeeStatusEvent

ActiveRadioButton rollover event → DisplayInfoEvent

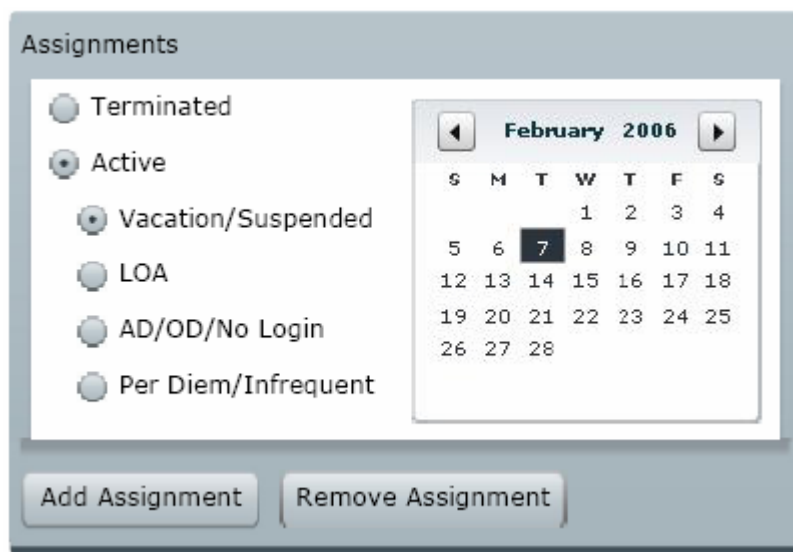
AddAssignButton click event → AddAssignmentEvent

AddAssignButton rollover event → DisplayInfoEvent

RemoveAssignButton click event → RemoveAssignmentEvent

RemoveAssignButton rollover event → DisplayInfoEvent

```
ModelLocator.RoleState = "Manager"
ModelLocator.EmployeeState = "Active"
```



ActiveRadioButton rollover event → DisplayInfoEvent

TerminatedRadioButton select event → ToggleEmployeeStatusEvent

TerminatedRadioButton rollover event → DisplayInfoEvent

VacaRadioButton rollover event → DisplayInfoEvent

LOARadioButton select event → ChangeActiveStatusEvent

LOARadioButton rollover event → DisplayInfoEvent

ADODRadioButton select event → ChangeActiveStatusEvent

ADODRadioButton rollover event → DisplayInfoEvent

PDRadioButton select event → ChangeActiveStatusEvent

PDRadioButton rollover event → DisplayInfoEvent

AddAssignButton click event → AddAssignmentEvent

AddAssignButton rollover event → DisplayInfoEvent

RemoveAssignButton click event → RemoveAssignmentEvent

RemoveAssignButton rollover event → DisplayInfoEvent

\* 如果这个View的其他Active状态中所有可能的用户动作都是一样的话,就不需要将它们列出;如果其他的Active状态中会出现新的组件或者不同的功能,你就要将它们列出来。在现在这种情况下,选择一个不同的Active状态只会增加/移除日期选择器,既然我们没有映射任何关于日期选择器的事件,也就没有新的事件,所以也就没有新的Command需要列出。



## Cairngorm: Front Controller

既然我们已经使用模型确定了那些需要使用业务逻辑的用户动作,那么现在我们就将所有这些事件放到一起并去掉相同的,这就得到了下面这些Event:

1. LoginEvent
2. ChangeRoleEvent
3. DisplayInfoEvent
4. SelectAcctEvent
5. ToggleFilterEvent
6. ToggleIntExtMgrEvent
7. SelectManagerEvent
8. ToggleEmployeeStatusEvent
9. ChangeActiveStatusEvent
10. AddAssignmentEvent
11. RemoveAssignmentEvent
12. LogoutEvent

本来看起来有很多的事件现在变成了少数几个被整个程序共享的事件。现在我们开始看到了提前使用模型进行设计的好处了吧!以前可能需要编写代码很多次然后才意识到要将它们放到一个共同使用的函数中,而现在在我们编写代码之前就把所有需要的函数列出来了。

在这个时候我们还可能决定是否需要其它一些触发后台处理的事件,例如一个 InitializeEvent (初始化事件), 或者查看哪里需要使用一个事件去触发另一个,例如, LoginEvent (登陆事件) 如果成功的话可能需要触发 InitializeEvent (初始化事件)。

在打完Event的草稿之后,下一步就是Front Controller,它是Cairngorm构架中很简单的一部分。Front Controller的工作就是监听我们的自定义Event并且触发相应的Command。在多数情况下这都是一对一的映射。

换句话说, actionscript代码是这样的:

当LogoutEvent发生 → 运行 LogoutCommand

使用上面列出的Event来起草你的Front Controller文件,接下来就是Command。

## Cairngorm: Commands

Cairngorm Commands是多数动作发生的地方。你将在这个模块中编写业务逻辑，检查Model Locator中的状态值，基于那些状态值编写分支代码，创建调用Service的Delegate，更新Model Locator等。我们不会讨论每个Command的具体代码，但是我们将为上面列出的每一个Event至少创建一个Command。在某些情况下你可能需要创建由其他Command或非用户事件触发的Command，例如InitializeCommand会在程序启动的时候自动运行。

下面是我们为需要编写的Command草拟的列表：

1. LoginCommand
2. ChangeRoleCommand
3. DisplayInfoCommand
4. SelectAcctCommand
5. ToggleFilterCommand
6. ToggleIntExtMgrCommand
7. SelectManagerCommand
8. ToggleEmployeeStatusCommand
9. ChangeActiveStatusCommand
10. AddAssignmentCommand
11. RemoveAssignmentCommand
12. LogoutCommand

就这么简单，即以前列出的Event的一对一的映射。在每个Command类中将添加一些业务逻辑来创建Value Object，将状态存储在Model Locator中，创建Delegate来呼叫Service进行RPC调用，检查调用结果，返回数据，在Model Locator对数据进行更改，设定新的状态信息等等。

## Cairngorm: Delegates

Delegate的最简单的形式就是一个中间人的角色。如果一个Command需要调用web service来获得一些数据，它将创建一个Delegate来完成这个调用。一个Command创建一个Delegate... Delegate调用一个指定的data Service... Service返回结果给Delegate... Delegate...返回结果给Command。

Delegate并不是100%必需的，但是当涉及到测试 & 程序环境的时候它们很有帮助。相对于在Command代码中使用查找 & 替换改变所有的引用来测试，将一个Delegate重映射到一个测试Service更为简单。

既然Delegate基本上只是映射到Service的简单类，那么每个Service只需要一个Delegate。首先我们需要确定我们的Service，然后再创建一个相应的Delegate来调用它。

例如，如果你有一个叫做GetAccountsService的Service，你将会创建一个相应的叫做GetAccountsDelegate的Delegate。因为ChangeRoleCommand将会获得用户帐号列表，所以ChangeRoleCommand将创建一个GetAccountsDelegate，然后这个GetAccountsDelegate将会调用GetAccountsService。

## Cairngorm: Services

Services有点像Model Locator，所有的远程过程调用（RPC）都保存在里面。在Services文件中你可以随心所欲地组织代码，例如…

```
Service1
    Operation1
    Operation2
或
Service1
    Operation
Service2
    Operation
```

这个Flex程序将会使用Web Services，不过Flex也可以处理其他类型的Service，例如Remote Objects 和 HTTP 调用，但是由于Cairngorm处理基本RPC Services的方式，使用什么类型的Service以及有多少Service并不重要。每个RPC Service都由一个Delegate呼叫来进行远程数据调用，然后将结果传给Delegate。

为了确定这个程序所需要的Services，要看一下我们的Commands（上文中已经列出）。每个Command都可能需要数据（参数）传入，而我們需要的就是可以处理那些的Services。

我们将查看每个Command并且检查其中的基本函数来确定可能会需要哪些Services。听起来有点复杂，但是当你读完下面的部分就会发现这实际上很简单。再一次，为了保持敏捷我们将使用直白的语言简单描述一下每个Command将要做些什么，而这些应该会提供足够的信息让我们开始编码。

**LoginCommand** - 需要传递uid和密码给Service，然后Service将会根据你选择的目录来验证信息并返回一个成功或失败响应。另外，由于定义了管理员，所以我们需要一个Service来确定用户是否是管理员。

- AuthenticateUserService (uid, pwd)
- IsAdminService (uid)

**InitializeCommand** - 我们需要使用我们的角色来生成帐号DataGrid。如果用户拥有Administrator权限我们还要生成Manager的DataGrid。获得LDAP帐号只是一个简单的调用，但是帐号的显示则依赖于角色，所以我们需要调用一个Service并将角色传送给它使其知道要返回什么数据。

- GetAcctsService (role)
- GetLDAPMgrsService()

**ChangeRoleCommand** - 这个Command会根据选中的角色来刷新显示帐号的DataGrid。上面已经有这个Service了，所以现在我们只是适用它。

- GetAcctsService (role)

**DisplayInfoCommand** - 如果帮助信息是存储在远程的话, 这个Command只需要调用一个Service。为了简单我们选择将信息硬编码到程序中, 所以这个Command不需要获取任何远程数据。

- N/A

**SelectAcctCommand** - 当一个帐号被选中的时候程序中只有一些状态信息会改变, 不会读取或写入远程数据, 所以这个Command不需要获取任何数据。

- N/A

**ToggleFilterCommand** - 这个Command将会通过改变帐号VO里的属性来改变DataGrid中显示的一些记录, 它不需要获取任何数据。

- N/A

**ToggleIntExtMgrCommand** - 这个Command将更改AssignmentView, 使其由显示LDAP帐号DataGrid变成显示几个文本输入框, 不需要任何远程数据。

- N/A

**SelectManagerCommand** - 当在Manager的DataGrid中选中一个Manager的时候我们只需要更改一些程序的状态和选中对象, 不需要任何Services。

- N/A

**ToggleEmployeeStatusCommand** - 同样地, 当在 活动/终止 之间来回切换时只有程序状态发生改变, 不需要任何Services。

- N/A

**ChangeActiveStatusCommand** - 只有程序状态发生改变, 不需要任何远程数据。

- N/A

**AddAssignmentCommand** - 这个Command需要将问题帐号DataGrid中的一个或多个选中的帐号分派给Manager DataGrid中一个选中的Manager。

- AddAssignmentService (MAcctVO, QAcctVO[])

**RemoveAssignmentCommand** - 和上面的一样，如果选中的帐号有重复，那么将为每个帐号移除前面指派的Manager。

- RemoveAssignmentService (QAcctVO[])

**LogoutCommand** - 只会改变状态并且清空Model Locator，不需要远程数据。

- N/A

在检查了所有的Command并确定了所需的Services之后，看起来我们只需要下面这几个：

- AuthenticateUserService (uid, pwd)
- IsAdminService (uid)
- GetAcctsService (role)
- GetLDAPMgrsService()
- AddAssignmentService (MAcctVO, QAcctVO[])
- RemoveAssignmentService (QAcctVO[])

既然已经确定了所需要的Services和传入的参数，那么我们就可以编写 Cairngorm Services文件并创建服务器端代码（Web Services）了

\* 注意：为了保持简单本程序并不安全，它没有加密任何web service信息来验证和鉴别客户端与服务器端的通信。要添加这个功能你需要在每个Service中传输额外的参数，例如经过加密的密码，session等，它们在web service返回数据之前被接收并验证。



## 编写程序

既然我们已经草拟了Events、 Commands 以及 Services, 那么现在就可以开始编码了。这个时候你可以发现编码阶段的工作有多么容易, Events、 Front Controller 和 Delegates 主要就是复制粘贴之后再做一些小的更改; Services 也很简单, 只需要一个由6个Operations (methods)组成的Web Service。创建好View并将业务逻辑写入到Commands之后主要工作就完成了, 而且定义那三个Value Objects也不会有很多问题。

所有的设计和模型的使用看起来很笨重, 而且公认如果设计过度肯定会导致问题并影响开发速度, 这些我都了解, 但是实际上对于Cairngorm而言, 使用模型并预可视化是可以为以后的工作节省时间的。

## 细微的调整

想要向一个已经完成的Cairngorm程序中添加新功能? 不会有大问题。你只要简单地定义并创建一个显示新数据的View, 使用这个View定义新的Event (用户动作), 将这些Event映射为Commands, 确定这些Commands是否需要新的远程Services还是可以使用已有的Services, 如果需要新的就创建它们, 然后为每个Service创建一个Delegate。

举个例子, 假如你想要添加一个可以删除帐号的按钮, 你也许会将它添加到帐号View中。

1. 这个按钮有一个点击事件, 将它映射为一个DeleteCompletedAccountEvent
2. 然后在Front Controller中添加一行代码将此事件映射为一个Command。
3. 这个Command需要包含一些业务逻辑: 创建一个Delegate调用一个Service, 当调用Service成功就移除该帐号并更新Model Locator。
4. 创建一个新的Service (或method)。

## 总结

希望这个教程会对你有用, 也希望它不只展示了提前设计Cairngorm Flex程序的好处, 还表明了使用模型设计工具和Cairngorm微架构的好处。在开始的时候适当地计划程序并使用象Cairngorm这样的设计模式可以帮助你更快地编码、更快地竣工, 使测试和调试更加简单, 而且当客户需要你添加新功能的时候可以更容易地进行调整。

谢谢观看, 祝你好运!