第 12 章　反应器

当电子图版中一个事件发生时，某些对象（我们称之为通知者，notifier）就自动地将该事件传递到其它对象。例如,当用户修改或删除一个对象时候，每个事件的相应的通告就会自动触发。

所谓反应器，就是接受事件的电子图版对象。反应器在它能够从通告者那里接收事件之前，必须明确地添加到该通告者的反应器列表中。

121 反应器概述

12.1.1 说明

ObjectCRX 为开发者提供了反应器机制，它类似于 MFC 的消息处理，利用它可以响应输入事件、实体添加或编辑或删除等事件。

给定的通告者可以在它的反应器列表中包含多个反应器。反应器的类的定义包含各种通告函数。当一个事件发生时，通告者就自动激活其反应器列表中每个反应器的相应的通告函数。

12.1.2 思路

1. 反应器概述

反应器类是从 CRxRxClass 类中派生而来，而不是从 CRxDbObject 类派生而来，反应器的类型不同，接收的事件也不相同。

编辑反应器用 CRxEditorReactor 类表示，用于监视电子图板特定的事件，例如可以截获各类用户特定操作命令。

数据库反应器用 CRxDbDatabaseReactor 类表示，用于监视数据库对象的创建、修改和删除事件，该反应器的通知者对象是数据库，它被添加到 CRxDbDatabase 类的反应器列表中。

对象反应器 CRxDbObjectReactor 类用于监视对象的创建、修改和删除事件。

12.1.3 步骤

在应用程序中使用反应器，一般有下面几个步骤：

1)　继承一个新的反应器类，并且为给反应器将要响应的每个事件实现响应函数的通

告函数。

2) 将该反应器实例化。

3) 将该反应器添加到通告者的反应器列表中。

当我们完成反应器操作后，应该遵循下面的步骤：

1) 将所有的通告者反应器列表中先前添加的反应器移出。

2) 删除反应器（除非它是一个驻留数据库的对象）。

12.1.4 效果

本节不需要实例。

12.1.5 小结

学习本节的内容之后，读者需要掌握下面的知识点：

- 理解通告者概念。
- 理解反应器概念。
- 理解反应器概述和分类。
- 掌握使用反应器的步骤。

12.2 编辑反应器

12.2.1 说明

编辑反应器 CRxEditorReactor 主要监控监视电子图板特定事件，例如电子图板的命令执行。

12.2.2 思路

CRxEditorReactor 类

此类为用于接受 CRxEditor 事件通知的自定义类的基类。从此类派生的类重载表示用户希望被通知的事件的方法。当这些派生的类之一的一个对象增加至电子图版编辑器的反应器列表中，而且与一个重载的方法关联的一个事件发生时，则该方法在此对象中调用。

此类中所有方法的默认执行都有一个立即的返回。

12.2.3 步骤

(1) 在 visual studio2010 中使用 ObjectCRX 向导新建一个 CustomReactor 项目，然后 Solution Explorer 窗口中选择上述新建项目右键选择【Add/class/c++ class】新建 CCmdMonitorReactor 类如下图 12.1 所示，新建类基类是 CRxEditorReactor

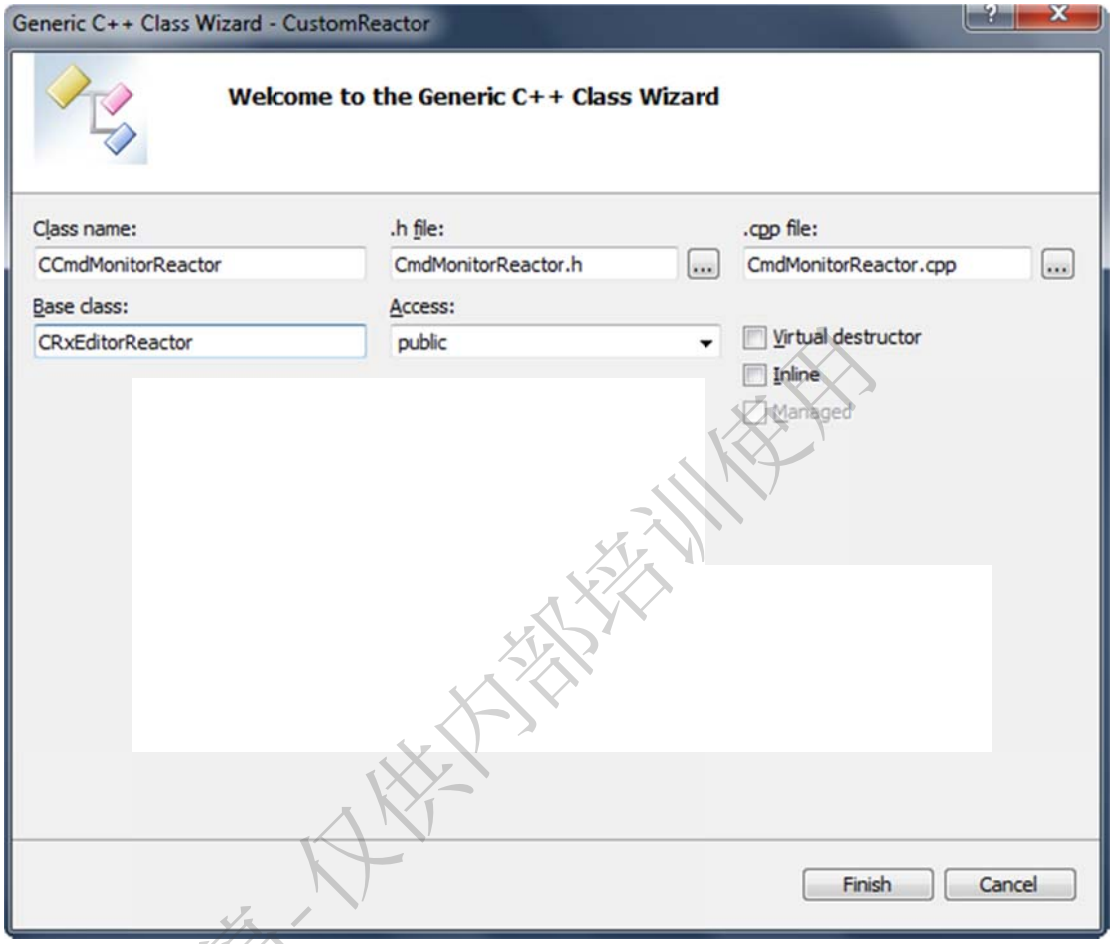

图 12.1 新建 CCmdMonitorReactor 向导

(2) 在新建类头文件中添加宏定义和成员属性如下：

```
public:
CRX_DECLARE_DYNAMIC(CCmdMonitorReactor) ;

protected:   bool mbAutoInitAndRelease ;
```

> 注意：自定义类时，要使用 CRX_DEFINE_NULL_CLSID 宏定义，例如：CRX_DEFINE_NULL_CLSID(CCmdMonitorReactor)class CCmdMonitorReactor :public CRxEditorReactor。这样可以保证类声明时唯一性。如果不包含上述宏定义，在编译时会出现链接错误。

然后添加新建类的构造函数和析构函数 声明如下：

```
CCmdMonitorReactor (const bool autoInitAndRelease =true) ;
 virtual ~CCmdMonitorReactor () ;
```

函数的实现如下：

```
CCmdMonitorReactor::CCmdMonitorReactor (const bool autoInitAndRelease) : CRxEditorReactor(),
mbAutoInitAndRelease(autoInitAndRelease)
{
    if ( autoInitAndRelease ) {
        if ( crxedEditor )
            crxedEditor->addReactor (this) ;
        else
            mbAutoInitAndRelease =false ;
    }
}

//-----------------------------------------------------------------------
CCmdMonitorReactor::~CCmdMonitorReactor () {
    Detach () ;
}
```

(3) 然后添加相关必须类

声明如下：

```
virtual void Attach () ;
virtual void Detach () ;
virtual CRxEditor *Subject () const ;
virtual bool IsAttached () const ;
```

实现函数如下：

```
//-----------------------------------------------------------------------
void CCmdMonitorReactor::Attach () {
    Detach () ;
    if ( !mbAutoInitAndRelease ) {
        if ( crxedEditor ) {
            crxedEditor->addReactor (this) ;
            mbAutoInitAndRelease =true ;
        }
    }
}

void CCmdMonitorReactor::Detach () {
    if ( mbAutoInitAndRelease ) {
        if ( crxedEditor ) {
            crxedEditor->removeReactor (this) ;
            mbAutoInitAndRelease =false ;
        }
```

```
    }
}

CRxEditor *CCmdMonitorReactor::Subject () const {
    return (crxedEditor) ;
}

bool CCmdMonitorReactor::IsAttached () const {
    return (mbAutoInitAndRelease) ;
}
```

(4)　添加commandWillStart和commandEnded两个函数的声明如下：

```
    public:
    virtual void commandWillStart(const CxCHAR * cmdStr);
    virtual void commandEnded(const CxCHAR* cmdStr);
```

实现代码如下：

```
void CCmdMonitorReactor::commandWillStart(const ACHAR * cmdStr)
{
    if(wcscmp(cmdStr, _T("MOVE"))==0 )
    {
        // 如果是MOVE，提示用户
        crxutPrintf(_T("\n开始执行MOVE命令:\n"));
    }

}

// ----------------------------------------------------------------------
void CCmdMonitorReactor::commandEnded(const ACHAR * cmdStr)
{
    if(wcscmp(cmdStr, _T("MOVE"))==0 )
    {
        // 如果是MOVE，提示用户
        crxutPrintf(_T("\n执行MOVE命令完毕.。\n"));
    }
}
```

(5)　自定义反应器完成后，需要将创建反应器对象添加到反应器列表中，可以通过注册命令的方式，也是程序初始化事件的处理函数中添加。需要首先声明一个反应器的实例对象：

```
CCmdMonitorReactor      *gEditorReactor = NULL;
```

在程序初始化事件处理函数中创建反应器对象实例：

```
virtual AcRx::AppRetCode On_kInitAppMsg(void *pkt)
```

```
{
    // TODO: Load dependencies here
    // You *must* call On_kInitAppMsg here
    AcRx::AppRetCode retCode =AcRxArxApp::On_kInitAppMsg(pkt);
    // TODO: Add your initialization code here
    //  创建编辑反应器对象实例
    if (gEditorReactor == NULL)
        gEditorReactor = new CCmdMonitorReactor ();
    return (retCode) ;
}
```

当程序加载时反应器即被创建，在应用程序卸载的时候将反应器对象从反应器列表中移除并删除反应器对象，代码如下：

```
virtual AcRx::AppRetCode On_kUnloadAppMsg(void *pkt)
{
    // TODO: Add your code here
    // You *must* call On_kUnloadAppMsg here
    AcRx::AppRetCode retCode =AcRxArxApp::On_kUnloadAppMsg(pkt);
    // TODO: Unload dependencies here
    //  将反应器对象移除
    if (gEditorReactor!= NULL)
    {
        delete gEditorReactor;
        gEditorReactor = NULL;
    }
    return (retCode);
}
```


12.2.4 效果

（1）编译链接程序，启动电子图板 2011，加载生成的 CRX 文件。

12.2.5 小结

学习本节的内容之后，读者需要掌握下面的知识点：

- 理解编辑反应器 CRxEditorReactor。
- 掌握编辑反应器 CRxEditorReactor 派生类方法。
- 掌握编辑反应器 CRxEditorReactor 派生类的方法的重载。
- 掌握编辑反应器 CRxEditorReactor 派生对象的注册和卸载。

12.3 数据库反应器

12.3.1 说明

CRxDbDatabaseReactor 类用于监视数据库对象的创建、修改和删除事件。

12.3.2 思路

CRxDbDatabaseReactor 类常用的事件的事件如下表：

事　　件	说　　明
objectAppended	当对象添加到数据库中时触发
objectModified	当对象被修改时触发
objectErased	当对象被删除时触发
objectOpenedForModify	当对象用写的方式打开时触发
objectReAppended	当对象重新到数据库中时触发
objectUnAppended	当对象取消添加到数据库中时触发

12.3.3 步骤

(1) 在建立的CustomReactor项目中，Solution Explorer窗口中选择上述新建项目右键选择【Add/class/c++ class】新建CSDatabaseReactor类如下图12.2所示，新建类基类是CRxDbDatabaseReactor

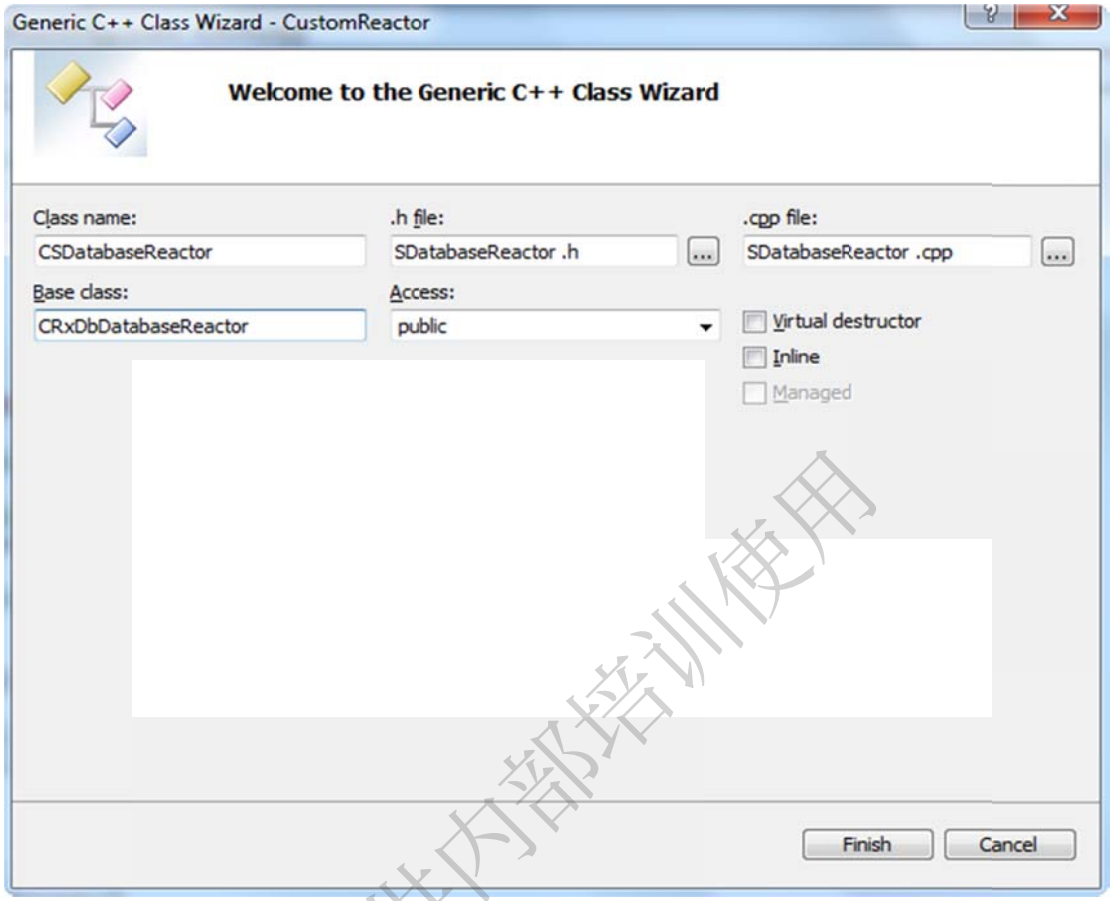

图 12.1 新建 CSDatabaseReactor 向导

(2) 在新建类头文件中添加宏定义和成员属性如下：

```
public:        CRX_DECLARE_DYNAMIC(CSDatabaseReactor) ;
protected:     CRxDbDatabase *mpDatabase ;
```

注意添加 CRX_DEFINE_NULL_CLSID 宏定义。

添加构造函数和析构函数，声明如下：

```
public:
    CSDatabaseReactor (CRxDbDatabase *pDb =NULL) ;
    virtual ~CSDatabaseReactor () ;
```

实现代码如下：

```
CSDatabaseReactor::CSDatabaseReactor (CRxDbDatabase *pDb) : CRxDbDatabaseReactor(),
mpDatabase(pDb) {
        if ( pDb )
            pDb->addReactor (this) ;
}
CSDatabaseReactor::~CSDatabaseReactor () {
    Detach () ;
}
```

(3)　添加相关函数，声明如下：

```
virtual void Attach (CRxDbDatabase *pDb) ;
virtual void Detach () ;
virtual CRxDbDatabase *Subject () const ;
virtual bool IsAttached () const ;
```

实现代码如下：

```
 void CSDatabaseReactor::Attach (CRxDbDatabase *pDb) {
Detach () ;
if ( mpDatabase == NULL ) {
      if ( (mpDatabase =pDb) != NULL )
            pDb->addReactor (this) ;
}
}

void CSDatabaseReactor::Detach () {
      if ( mpDatabase ) {
            mpDatabase->removeReactor (this) ;
            mpDatabase =NULL ;
      }
}

CRxDbDatabase *CSDatabaseReactor::Subject () const {
      return (mpDatabase) ;
}

bool CSDatabaseReactor::IsAttached () const {
      return (mpDatabase != NULL) ;
}
```

(4)　要重写的函数声明如下：

```
public:
      virtual void objectAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj);
      virtual void objectModified(const CRxDbDatabase * dwg, const CRxDbObject * dbObj);
      virtual void objectErased(const CRxDbDatabase * dwg, const CRxDbObject * dbObj,
      CAXA::Boolean pErased);
      virtual void objectOpenedForModify(const CRxDbDatabase * dwg, const CRxDbObject * dbObj);
      virtual void headerSysVarWillChange(const CRxDbDatabase * dwg, const CxCHAR * name);
      virtual void headerSysVarChanged(const CRxDbDatabase * dwg, const CxCHAR * name,
      CAXA::Boolean bSuccess);
      virtual void objectReAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj);
      virtual void objectUnAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj);
```

实现的代码如下：

```
void CSDatabaseReactor::objectAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj)
{

    //如果是圆，改变圆的颜色为红色
    if(dbObj->isKindOf (CRxDbCircle::desc ()) )
    {
        CRxDbCircle* pCir = CRxDbCircle::cast(dbObj);
        //升级打开模式,修改对象

        pCir->upgradeOpen ();
        pCir->setColorIndex (1);
        crxutPrintf(_T("\n创建了圆\n"));

    }
    else if(dbObj->isKindOf (CRxDbLine::desc ()))
    {
        //如果是直线，改变颜色为黄色
        CRxDbLine * pLine= CRxDbLine::cast(dbObj);
        //升级打开模式,修改对象
        pLine->upgradeOpen ();
        pLine->setColorIndex (2);
        crxutPrintf(_T("\n创建了直线\n"));

    }

}
// -----------------------------------------------------------------------
void CSDatabaseReactor::objectModified(const CRxDbDatabase * dwg, const CRxDbObject * dbObj)
{
    CRxDbDatabaseReactor::objectModified (dwg, dbObj) ;
}

// -----------------------------------------------------------------------
void CSDatabaseReactor::objectErased(const CRxDbDatabase * dwg, const CRxDbObject * dbObj,
CAXA::Boolean pErased)
{
    CRxDbDatabaseReactor::objectErased (dwg, dbObj, pErased) ;
}
```

```
// ----------------------------------------------------------------------------
void CSDatabaseReactor::objectOpenedForModify(const CRxDbDatabase * dwg, const CRxDbObject *
dbObj)
{
    CRxDbDatabaseReactor::objectOpenedForModify (dwg, dbObj) ;
}

// ----------------------------------------------------------------------------
void CSDatabaseReactor::headerSysVarWillChange(const CRxDbDatabase * dwg, const CxCHAR * name)
{
    CRxDbDatabaseReactor::headerSysVarWillChange (dwg, name) ;
}

// ----------------------------------------------------------------------------
void CSDatabaseReactor::headerSysVarChanged(const CRxDbDatabase * dwg, const CxCHAR * name,
CAXA::Boolean bSuccess)
{
    CRxDbDatabaseReactor::headerSysVarChanged (dwg, name, bSuccess) ;
}

// ----------------------------------------------------------------------------
void CSDatabaseReactor::objectReAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj)
{
    CRxDbDatabaseReactor::objectReAppended (dwg, dbObj) ;



}

// ----------------------------------------------------------------------------
void CSDatabaseReactor::objectUnAppended(const CRxDbDatabase * dwg, const CRxDbObject * dbObj)
{
    CRxDbDatabaseReactor::objectUnAppended (dwg, dbObj) ;
}
```

(5)　和编辑反应器一样，要在程序中加载完成的反应器的注册，则首先声明一个反应器的实例对象。代码如下：

```
CSDatabaseReactor *gDbReactor = NULL;
```

同样在程序的初始化事件处理函数中创建反应器对象实例，如下：

```
// 创建数据库反应器对象实例
```

```
        if (gDbReactor == NULL)
            gDbReactor = new CSDatabaseReactor
(crxdbHostApplicationServices()->workingDatabase());
```

在应用程序卸载的时候将反应器对象从反应器列表中移除并删除反应器对象，代码如下：

```
// 将反应器对象移除
        if (gDbReactor!= NULL)
        {
            delete gDbReactor;
            gDbReactor = NULL;
        }
```

12.3.4 效果

（1）编译链接程序，启动电子图板 2011，加载生成的 CRX 文件。

12.3.5 小结

学习本节的内容之后，读者需要掌握下面的知识点：

- 理解数据库编辑反应器 CDbDatabaseReactor。
- 掌握数据库编辑反应器 CDbDatabaseReactor 派生类方法。
- 掌握数据库编辑反应器 CDbDatabaseReactor 派生类的方法的重载。
- 掌握数据库编辑反应器 CDbDatabaseReactor 派生对象的注册和卸载。