MFC CObject 클래스

HCI Programming 2 (321190)
2008년 가을학기
9/30/2008
박경신

CObject 클래스 특징

- 실시간 클래스 정보 (Run-Time Class Information)
- 동적 객체 생성 (Dynamic Object Creation)
- 직렬화 (Serialization)
- 실시간 객체 상태 검사 (Run-Time Object Diagnostics)

CObject 클래스

- 생성
  - 디폴트 생성자, 복사 생성자, new 연산자, delete 연산자, operator= 할당 연산자
- 실시간 클래스 정보 관련 함수
  - GetRuntimeClass – 객체의 클래스와 일치하는 CRuntimeClass 구조체를 반환
  - IsKindOf – 주어진 클래스와 객체와의 관계를 검사
- 직렬화 관련 함수
  - IsSerializable – 객체가 직렬화될 수 있는 지를 알아보기 위해 검사
  - Serialize – 아카이브를 이용해 일반 파일에 객체를 저장하거나 또는 파일로부터 읽어들임
- 진단관련 함수
  - AssertValid – 객체의 멤버들에 대한 유효성 여부를 검사
  - Dump – 객체의 멤버들에 대한 진단 덤프를 생성

http://msdn2.microsoft.com/en-us/library/c1t5f48c(VS.80).aspx

CRuntimeClass 구조체

- Data Members
  - m_lpszClassName – 현재 클래스 이름
  - m_nObjectSize - 객체의 크기 (in bytes)
  - m_pBaseClass - 기반 클래스의 CRuntimeClass 구조체 포인터
  - m_pfnCreateObject – 동적으로 객체를 생성하는 함수 포인터
  - m_pfnGetBaseClass - CRuntimeClass 구조체 반환 (only available when dynamically linked).
  - m_wSchema – 로드된 클래스의 수
- Operations
  - CreateObject – 동적으로 객체 생성
  - FromName – 클래스 이름을 이용해서 동적으로 객체 생성
  - IsDerivedFrom – 이 클래스가 m_pBaseClass에서 파생된 클래스인지를 확인

http://msdn2.microsoft.com/en-us/library/fych0hw6(VS.80).aspx

실행 시간 클래스 정보

- CObject의 실행 시간 클래스 정보 (RTCI, Run-Time Class Information)는 개발자에게 객체의 정보를 실행시간 (run-time) 시에 결정할 수 있도록 해줌
- DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 매크로를 추가하면 IsKindOf를 호출할 수 있게 됨

NOTE: RUNTIME_CLASS 매크로는 해당 클래스에 대한 CRuntimeClass 형을 반환함

```
#define RUNTIME_CLASS(class_name)
            ((CRuntimeClass*) (&class_name::class##class_name))
```


실행 시간 클래스 정보

- DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 매크로

```
#define DECLARE_DYNAMIC(class_name) \
public: \
    static const AFX_DATA CRuntimeClass class##class_name; \
    virtual CRuntimeClass* GetRuntimeClass() const;

#define IMPLEMENT_DYNAMIC(class_name, base_class_name) \
    IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, 0xFFFF, NULL)
#define IMPLEMENT_RUNTINECLASS(class_name, base_class_name, wSchema, pfnNew) \
    AFX_COMDAT const AFX_DATADEF CRuntimeClass class_name::class##class_name = { \
    #class_name, sizeof(class class_name), wSchema, pfnNew, \
    RUNTIME_CLASS(base_class_name), NULL}; \
    CRuntimeClass* class_name::GetRuntimeClass() const \
        { return RUNTIME_CLASS(class_name); }
```


실행 시간 클래스 정보 사용 예

```
// MyClass.h
class CMyClass : public CObject {
   DECLARE_DYNAMIC(CMyClass)
   …
};
// MyClass.cpp
#include "MyClass.h"
IMPLEMENT_DYNAMIC(CMyClass, CObject)
BOOL IsMyClass(CObject *pObj) {
   // pObj가 가리키는 객체가 CMyClass 타입인지 확인하여 맞으면 할당
   if(pObj->IsKindOf(RUNTIME_CLASS(CMyClass)){
      CMyClass *pMyObject = (CMyClass *)pObj;
   }
   else{
      ...
   }
}
```


동적 객체 생성

- 사용자가 정의한 CObject에서 파생된 클래스에 동적 객체 생성 (Dynamic Object Creation)을 가능하게 함
- DECLARE_DYNCREATE/IMPLEMENT_DYNCREATE 매크로를 추가하면 동적 객체 생성 사용 가능 – 즉 CreateObject 를 호출할 수 있게 됨
- 이 매크로를 사용할 경우에는 DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 을 사용할 필요가 없음 - 이미 포함되어 있으므로. 두 매크로를 함께 사용할 경우에는 컴파일 에러 발생

동적 객체 생성

- DECLARE_DYNCREATE/IMPLEMENT_DYNCREATE 매크로

```
#define DECLARE_DYNCREATE(class_name) \
    DECLARE_DYNAMIC(class_name) \
    static CObject* PASCAL CreateObject();

#define IMPLEMENT_DYNCREATE(class_name, base_class_name) \
    CObject* PASCAL class_name::CreateObject() \
        { return new class_name; } \
    IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, 0xFFFF, \
        class_name::CreateObject)
```


동적 객체 생성 사용 예

```
// MyClass.h
class CMyClass : public CObject {
   DECLARE_DYNCREATE(CMyClass)
public:
   CMyClass();
   ...
};
// MyClass.cpp
#include "MyClass.h"
IMPLEMENT_DYNCREATE(CMyClass, CObject)
...
// 객체를 동적으로 생성한다.
CRuntimeClass* pRuntimeClass = RUNTIME_CLASS(CMyClass);
CObject* pObject = pRuntimeClass->CreateObject();
// 객체를 성공적으로 생성했는지 여부를 확인한다.
ASSERT(pObject->IsKindOf(RUNTIME_CLASS(CMyClass)));
```


직렬화

- 객체의 상태를 저장한 후 나중에 그 상태를 읽어 복원시키는 것을 MFC에서는 직렬화 (Serialization)이란 용어로 부름
- DECLARE_SERIAL/IMPLEMENT_SERIAL 매크로를 추가하고, 개발자가 정의한 멤버 변수를 read/write 하도록 CObject::Serialize()를 재정의 (overriding) 해야 함
- 만약 개발자가 직렬화를 사용하게 된다면 개발자는 file의 형식 (format)에 대한 아무런 걱정 없이 read/write 할 수 있음 – 단지 객체를 read/write하는 순서만 맞추어주면 됨
- 직렬화 매크로를 사용하면 DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 과 DECLARE_DYNCREATE/IMPLEMENT_DYNCREATE 을 사용할 필요 없음 – 즉 직렬화된 데이터를 읽어 들일 때 저장된 데이터에 맞는 클래스가 동적 생성되는 기능을 사용할 수 있는 것임

직렬화

- DECLARE_SERIAL/IMPLEMENT_SERIAL 매크로

```
#define DECLARE_SERIAL(class_name) \
    DECLARE_DYNCREATE(class_name) \
    AFX_API friend CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb);

#define IMPLEMENT_SERIAL(class_name, base_class_name, wSchema) \
    CObject* PASCAL class_name::CreateObject() \
        { return new class_name; } \
    IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, wSchema, \
        class_name::CreateObject) \
    AFX_CLASSINIT _init_##class_name(RUNTIME_CLASS(class_name)); \
    CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb) \
    { \
        pOb = (class_name*) ar.ReadObject(RUNTIME_CLASS(class_name)); \
        return ar; \
    } \
```

직렬화

```
CObject* CArchive::ReadObject(const CRuntimeClass* pClassRefRequested)
{
    UINT nSchema;
    DWORD obTag;
    CRuntimeClass* pClassRef
                = ReadClass(pClassRefrequested, &nSchema, &obTag);
    // check to see if tag to already loaded object
    CObject* pOb;
    pOb = pClassRef->CreateObject();
    // Serialize the object
    UINT nSchemaSave = m_nObjectSchema;
    m_nObjectSchema = nSchema;
    pOb->Serialize(*this);
    m_nObjectSchema = nSchemaSave;
    return pOb;
}
```


직렬화

- Serialization 매크로에 연산자 <<가 포함되어 있지 않음
- 연산자 <<는 WriteObject을 호출하고, WriteObject는 WriteClass를 호출하고, 그런 다음 WriteObject는 그 객체의 Serialize()를 호출한다
- WriteObject는 CRuntimeClass information을 필요로 하지 않기 때문에 CObject를 위한 연산자 << 하나면 충분하다
- 따라서 CObject 파생물을 위한 연산자 <<는 선언될 필요도 구현될 필요도 없다

```
CArchive& operator<<(CArchive& ar, const CObject* pOb)
{
    ar.WriteObject(pOb);
    return ar;
}
```


직렬화 사용 예

```
// MyClass.h
class CMyClass : public CObject
{
   DECLARE_SERIAL(CMyClass)
public:
   CMyClass();
   WORD m_MyVar1
   DWORD my_MyVar2;
   virtual void Serialize (CArchive& ar);   // 가상함수를 재정의
};
// MyClass.cpp
#include "MyClass.h"
IMPLEMENT_SERIAL(CMyClass, CObject, 1)
void CMyClass::Serialize (CArchive& ar)
{
   // CObject가 제공하는 가상 함수인 Serialize() 함수를 재정의한다.
}
...
```


Serialize 함수

- 애플리케이션이 파일을 읽거나 쓰려 할 때 도큐먼트 객체의 Serialize( ) 함수가 호출되는데 이때 파일을 읽거나 쓰는데 사용되는 CArchive 객체가 인자로 넘겨짐
- IsLoading() 이나 IsStoring() 함수 호출해서 읽기 또는 쓰기에 있는 지 결정

```
void CMyClass::Serialize(CArchive &ar) {
    if(ar.IsStoring( )) { // 아카이브가 현재 쓰기 상태에 있나?
        // 그렇다. 이제, 변수를 순서에 맞춰 아카이브에 저장한다.
        ar << m_MyVar1 << m_MyVar2;
    }
    else { // 아니다. 변수를 순서에 맞춰 읽은다.
        ar >> m_MyVar1 >> m_MyVar2;
    }
}
```

직렬화 가능한 클래스 만들기

- 각각의 레코드를 캡슐화하는 클래스를 만들고 이들 레코드를 배열에 담아 처리
- 클래스는 CObject 클래스에서 파생하고, 뷰 클래스에 추가한 컨트롤에 물려둔 변수에 해당하는 데이터 필드를 멤버 변수로 가짐
- 멤버 변수를 액세스 하기 위한 멤버 함수를 가짐
- 이 클래스의 직렬화를 위한 매크로와 Serialize() 함수를 가져야 함

직렬화 가능한 클래스 만들기 예제

- Point3D.h

```
class CPoint3D : public CObject
{
public:
   DECLARE_SERIAL(CPoint3D)
   int m_nX, m_nY, m_nZ;
   CPoint3D();
   CPoint3D(int x, int y, int z);
  ~ CPoint3D();
   void Serialize(CArchive& ar);
};
```


직렬화 가능한 클래스 만들기 예제

- Point.cpp

```
#include "Point3D.h"
IMPLEMENT_SERIAL(CPoint3D, CObject, 1)
CPoint3D::CPoint3D( ) {
   m_nX = m_nY = m_nZ = 0;
}
CPoint3D::CPoint3D(int x, int y, int z) {
   m_nX = x; m_nY = y; m_nZ = z;
}
void CPoint3D::Serialize(CArchive& ar) {
   CObject::Serialize(ar);
   if(ar.IsStoring())        // 쓰고 읽는 변수의 순서가 동일해야 함
      ar << m_nX << m_nY << m_nZ;
   else
      ar >> m_nX >> m_nY >> m_nZ;
}
```


직렬화 가능한 클래스 만들기 예제

- main.cpp

```
void main()
{
   TRY { // 쓰기
      CPoint3D point(10, 20, 30);
      CFile file;
      file.Open(_T("File.dat"), CFile::modeCreate | CFile::modeWrite);
      CArchive ar(&file, CArchive::store);
      ar << &point;
   }
   CATCH(CFileException, e)  {
      e->ReportError();
   }
   END_CATCH
```

직렬화 가능한 클래스 만들기 예제

- main.cpp

```
  TRY  { // 읽기
      CPoint3D point(10, 20, 30);
      CFile file;
      file.Open(_T("File.dat"), CFile::modeRead);
      CArchive ar(&file, CArchive::load);
      ar >> &point;
   }
   CATCH(CFileException, e)  {
      e->ReportError();
   }
   END_CATCH
}
```


타당성 점검

- CObject의 진단 (diagnostics)은 (1) 메모리 누수 검출 (memory leak detection)과 (2) 메모리 statisticcs과 같은 고급진단 두 부류로 나눌 수 있음
- TRACE 매크로는 printf()와 같은 출력 결과를 output 창에 보여줌

```
CPoint point(10,20);
TRACE("At this point, my CPoint is : %d %d\n", point.x, point.y);
```

- 모든 CObject의 파생 클래스들은 상태를 출력할 수 있는 Dump 함수를 가짐 – Debug 모드에서만 실행되도록 설정
- 모든 CObject의 파생 클래스들은 AssertValid 함수로 실시간 (Run-Time) 시에 멤버가 유효한지를 검사
- MFC는 ASSERT 매크로로 실시간 시에 에러 상태를 검사

타당성 점검 사용 예

```
// MyClass.h
class CMyClass : public CObject {
   // 멤버 변수
   int m_start;
   int m_end;
public:
   virtual void AssertValid( ) const; // AssertValid() 함수 재정의

};
// MyClass.cpp
#include "MyClass.h"
virtual void CMyClass::AssertValid( ) const {
   CObject::AssertValid();
   ASSERT(m_start > 0);          // m_start 값이 0보다 큰 값인지를 검증
   ASSERT(m_end < 100);
}
…
```


예제

```
CString aMsg;
// 동적 객체 생성
CObject *ptrObj = (CObject *) RUNTIME_CLASS(CAccount)->CreateObject();
// 타당성 점검
ASSERT(ptrObj->IsKindOf(RUNTIME_CLASS(CAccount)));
// 실시간 클래스 정보
if(ptrObj->IsKindOf(RUNTIME_CLASS(CAccount))) {
      CAccount *a = (CAccount *) ptrObj;
      a->Deposit(1000);
      a->Withdraw(500);
      aMsg.Format("이   름: %s, 계좌번호: %s, 잔   액: %d", a->GetUser(),
                        a-> GetNumber(), a->GetBalance());

      // 윈도우창에Account a값을출력
      dc.TextOut(100, 120, (LPCTSTR)aMsg, lstrlen(aMsg));
}
```

Practice

- SimpleCObject 프로그램을 이용하여 CPoint3D에 Serialize (직렬화)를 구현하라
  - TRY {
    CPoint3D d(100,200,300);
    CPoint3D e(400,500,600);
    CPoint3D f(700,800,900);
    // serialize를 이용하여 (“Point.dat”) 로 저장
    } CATCH(CFileException, e) {
    // error handling..
    } END_CATCH
  - TRY {
    // serialize를 이용하여 (“Point.dat”)를 읽어서 프린트
    } CATCH(CFileException, e) {
    // error handling..
    } END_CATCH