能力，但对于大多数情况，VB 6.0 不失为一个快速实现 COM 组件的开发环境。

COM+推出之后，它的开发模式也将有一些转变，尤其对于 Visual C++程序员，在编译时刻程序员可以在代码中使用一些说明性的语句来设置 COM+组件的属性，比如 CLSID、ProgID、线程模型以及双接口等。如果不指定这些属性，编译器将使用缺省值。以前为了使 COM 组件支持某些非缺省的特性，必须通过编写代码来实现这些特性，所以程序员一定要对各种特性了解得非常清楚才能够编写出正确的代码来，这也是实现 COM 组件的一个难点。COM+一方面与操作系统紧密结合，另一方面从开发的角度来讲，COM+将进一步与编译器结合，它将扩展 C++的一些语法，使得可以在代码中描述 COM+特性，然后由编译器直接提供这些特性的支持，从而减少程序员的工作量，提高 COM+组件的生产效率。

在代码中利用说明性的语句指示编译器产生与 COM+组件有关的元数据(Metadata)，COM+运行系统将利用这些元数据管理 COM+组件。从某种意义上讲，可以认为元数据是一些类型库信息，所以，实际上支持 COM+组件的 C++开发系统将把 IDL/ODL 的语法与 C++语法结合起来。后面讲到基于属性的编程模型时将会看到这种情况。

全面支持 COM+组件的开发工具要等到 Windows 2000 发布之后，在 VisualStudio 的下一个版本中才可能实现。作为一种兼容的方案，在现在的 VisualC++版本中，编译器仍然只支持原先的 C++语法，当它在预处理过程中，遇到说明性的描述信息时，它把这些属性信息交给属性分析器去处理。属性分析器是一个编译扩展模块，它把属性信息转换成 C++代码，然后送回编译器，编译器再把这些源代码编译到目标代码中。属性分析器产生的其他一些信息，比如类型信息，也被编入最终代码。

2. 基于属性的 C++编程语言

基于属性的编程模型将直接把 COM+组件的属性信息写到 C++源代码中，指导编译器产生 COM+组件，这样可以使程序员不必编写底层的处理代码，因为这些代码对于几乎所有的组件都差不多，因此让开发工具直接产生这些代码可避免重复劳动。这种方式比 MFC 的宏以及 ATL 的模板类更为直接。

属性可以用在任何说明性的语句前面，比如 C++类的声明、变量的声明都可以在其前面用方括号指定其属性。需要注意的是，通常不在类型或者实例定义语句中指定属性信息。下面的代码（程序 9-2-1）说明了属性的用法：

程序 9-2-1：

```
//SecurityMgr.idl:IDLsourceforSecurityMgr.dll
```

```
//ThisfilewillbeprocessedbytheMIDLtoolto
//producethetypelibrary(SecurityMgr.tlb)andmarshallingcode.

import"oaidl.idl";
import"ocidl.idl";
    [
        object,
        uuid(9189A64D-B246-11D4-A01A-0C50BAC5C6F4),
        dual,
        helpstring("IAccessControlInterface"),
        pointer_default(unique)
    ]
    interfaceIAccessControl:IDispatch
    {
        [propget, id(1), helpstring("propertyUserName")]
        HRESULTUserName([out, retval]BSTR*pVal);
        [propput, id(1), helpstring("propertyUserName")]
        HRESULTUserName([in]BSTRnewVal);
        [id(2), helpstring("methodIsUserAllowed")]
        HRESULTIsUserAllowed([in]BSTRUserName, [in]
        shortReqFunction, [out, retval]VARIANT_BOOL*pbIsAllowed);
        [id(3), helpstring("methodIsAllowed")]HRESULTIsAllowed
        ([in]shortReqFunction, [out, retval]VARIANT_BOOL*
        pbIsAllowed);
    };

[
    uuid(9189A641-B246-11D4-A01A-0050BAC5C6F4),
    version(1.0),
    helpstring("SecurityMgr1.0TypeLibrary")
]
librarySECURITYMGRLib
{
    importlib("stdole32.tlb");
```

```
    importlib("stdole2.tlb");

    [
        uuid(9189A64F-B246-11D4-A01A-0050BAC5C6F4),
        helpstring("AccessControlClass")
    ]
    coclassAccessControl
    {
        [default]interfaceIAccessControl;
    };
};
```

如果读者熟悉 IDL 或者 ODL 语法，那么对上面例子中的属性描述一定非常清楚。Visual C++的属性分析器分析属性关键字，并产生相应的 C++源代码(实际上是 ATL 代码)。

基于属性的编程模型为 VisualC++程序员开发 COM+组件提供了捷径，它避免了 MFC 繁杂的宏定义和 ATL 晦涩的模板类。属性编程模型还包括其他一些语义或语法，比如事件定义、对象构造等，将可以在新版本的 Visual C++或者 COM+ SDK 中看到这些变化。

第十章　考试系统的应用需求分析

在进行一定的理论探讨之后，将讨论一个考试系统的设计。这个考试系统原型特别简单，的想法是通过简单的系统来说明基于组件的应用系统是如何设计和实现的。在这里将描述基于组件的多层应用开发的某些问题，包括面向对象的设计、数据存取设计、设计 Visual C++的组件以及分布组件，使网络上的客户能够共享这些服务。

10.1　问题描述

下面所举的例子是一个简单考试系统。学生通过 Web 浏览器(或 Win32 客户端)注册自己的信息到数据库中，然后登录进入到考试系统。应用程序通过数据库中考试题目的资料，产生试题。学生通过 Web 浏览器(Win32 客户端)提交自己的答案。最后，考试结束后，应用程序计算考试分数显示给学生看。在这个系统的需求中有以下几点需要特别注意：

- 必须保证学生在考试过程中，如果中间出现了断电或其他故障，学生在另一台机子上仍能继续考试。不需要重新生成试题，也不需要从头开始做题。
- 试题的产生遵从以下规则。每一种题型都出若干道题。每一章节都出若干道题，难度也合理分配。在这些规则的基础上，随机出题。
- 为了遵从简单的原则，这个考试系统只实现判断题、单选题、多选题三种题型。关于难度的分配及章节的分配，不去实现，读者可以自己扩充。

10.2　面向对象的设计

面向对象分析是软件开发过程中的问题定义阶段。这一阶段最后得到的是对问题域的清晰、精确的定义。传统的系统分析产生一组面向过程的文档，定义目标系统的功能，面向对象的分析则产生一种描述系统功能和问题域基本特征的综合文档。它在更大的问题域内考虑问题，在分析过程中识别的概念是高层的抽象。这些抽象成为一个灵活的可扩充软件的基本构件块。面向对象的分析文档把问题当作一组相互作用的实体，并确定这些实体之间的关系。它是把系统看作一个能够以有控制的方式执行的模型。

任何良好的设计工作的第一步是决定（考虑）需要完成什么工作，谁是系统的使用者以及应用中所包含的业务规则。这个原型系统的场景非常简单，主要涉及4个概念：学生、教师、试题和考试过程。将对这四个概念进行建模和实现。学生和教师这两个概念具有许多共同的信息，对他们的操作也类似。因而可以将他们统一为一个概念——用户。这个原型系统的使用者是老师和学生，这意味着老师和学生是原型系统中的参与者（角色）。

10.2.1　标识对象（组件）

可以抽象出原型系统中包含的几个组件：

- 学生组件（Student Component）：学生组件代表了学生的信息，包括学生姓名、性别、学校、院系、班级等。它包括以下一些方法，检查学生身份的合法性，输入学生信息，修改和删除学生信息，根据学生 ID 获得班级、系、学校、学号和用户名信息。
- 老师组件（Teacher Component）：老师组件代表了老师的信息。它包括以下一些方法，检查老师的合法性，输入老师信息，修改和删除老师信息。
- 试题组件（Question Component）：试题组件代表了试题信息，包括问题、答案、题目类型、所属章节、难易程度等。它主要实现以下功能：试题的增删改，获取试题信息，获取试题类型等功能。
- 考试组件（Exam Component）：考试组件代表了考试信息，包括参与考试的学生信息，试题信息，学生答题信息，考试分数等。它主要实现了以下功能：产生试题，计算分数，保存答案等。

首先，考察一下老师组件和学生组件之间的相同点。寻找两类对象之间的相同点的行为称为“泛化”。老师和学生都有用户名，密码，姓名、性别等属性，都需要验证。这样发现存在一些公共的行为模式，可以将它们从这两类对象中抽取出来并放到了一个超类中。将创建一个“用户”类，它包含从上述两个类中抽取出来的公共方法和属性。在下面的讨论中，还将寻找两类帐户特性上的差异，相应地，这种行为称为“特化”。

10.2.2　用例

在面向对象的设计中用例是非常重要的。它们是用户与系统、系统内部各部分之间交互的场景。用例（Use Case）描述了系统是如何工作的，它们可以揭示需要建造什么样的系统，并指导制定完善的测试计划以验证系统是否满足了用户需求。

下面是原型系统中的几个用例：

• 登录操作：根据用户提供的信息（如用户名和密码等）查找 StudentAccount 表或 TeacherAccount 表，检验其是否合法。

• 产生试题：根据前面所提到的产生试题规则，随机产生试题。

• 保存答案：将学生提交的答案保存到数据库中。

• 计算分数：将学生提交的答案与标准答案比较，计算分数。

• 检查考试时间：检查考试剩余时间是否到 0。

10.2.3 对象设计

需要确定对象的数据成员及方法。公共的数据成员及方法代表了对象“是什么”，而具有私有或保护特性的方法及数据成员则负责描述一个商业对象是如何实现的。通过这种分离便可在改变对象的实现时不影响其公共界面。商业对象的接口仅在将其转移成分布对象时需要改变，因为需要遵循分布式对象标准并且不可能完全确定其他分布式对象或应用是如何与之交互的。

图 10-1 描述了原型系统所涉及的对象模型，这是经过初步抽象的结果，进一步的抽象将在后面给出。

图 10-1 原型系统对象图（一）

10.2.4 通过事件顺序图调整对象设计

对象中所定义的方法必须支持要解决问题。那么在对原型进行分析时，如

何确定对象中定义的方法是完整的，即定义的所有方法是否能够支持实现用户需求？可以通过事件顺序图进行检验。事件顺序图描述了在一个用例中对象是如何进行交互的，它指出了对象间的事件流，每个事件代表了一个对象应该提供的服务。

下面考察这样一个场景——考生登录参加考试。在后面，介绍系统的具体实现时主要围绕这个场景来介绍。图 10-2 显示了对象之间的交互方式。

通过考察事件顺序图，发现方法 SaveAnswer()及 CheckExist()没有出现在的初始对象模型中。但是这两个方法是实现这一场景所必需的。SaveAnswer()方法是实现将考生输入的答案保存到数据库中的方法。计算分数时，从数据库中取出考生的答案与试卷的正确答案比较，然后累计分数。CheckExist()方法帮助检查该考生是否已参加了考试，这主要是防止考生在答题过程出现故障被迫换一台机子使用或重新启动机子时，必须重新产生试题从头开始答题的情况发生。将产生的试题及考生做每道题的答案保存在数据库表中。如果考生中途停止考试了，他以前做题的记录已在数据库表中保存下来了。当他下次登录时，系统检查数据库表中是否已经有这个考生的纪录，他是否已完成考试，如果系统发现数据库表有他的记录并且他的考试尚未完成，则系统将它的考试记录显示出来，他就可以继续完成他的考试。这是实现考试过程处理所必需的，因此，需要更改对象模型。更改后的对象图如图 10-3 所示。

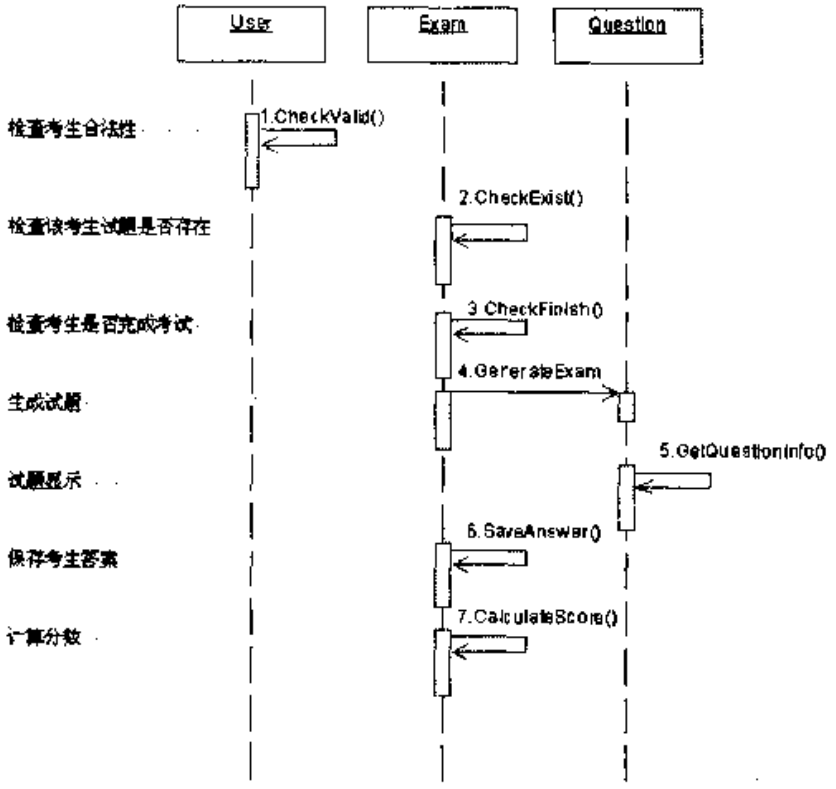

图 10-2　考生考试过程的场景图

图 10-3 原型系统的对象图（二）

10.2.5 数据存取设计

在设计中数据存取是需要仔细考虑的部分，必须确定对象模型映射到关系模型的方法。同时，对数据的访问必须遵循面向对象范型，在三层结构应用中，表示层是不直接与数据库服务器打交道的，它通过应用逻辑层中的对象（组件）访问数据库并取回结果，因此，在客户端应用不可能直接使用 SQL 语句或开发工具提供的数据库访问 API。例如，某个客户机应用想取出所有学生的列表，那么这个应用不能使用 SQL 语句直接访问数据库，而是调用某个对象的 GetStudentInfo 方法，这个方法通过 ADO 查询数据库，并将结果合并为一个字符串返回。客户机应用不需要知道如何取得结果集，这是对象的责任。

10.2.6 对象模型与关系模型的映射

大部分企业应用系统的数据存储使用关系型数据库，它们使用关系模型，关系模型是不支持继承的。那么如何用关系表来表示上述的学生、老师、试题考试对象结构的呢？这就需要解决对象模型到关系模型的映射问题，总的来说有三种策略：

（1）只把超类转换成表

超类把所有子类的属性和子类与其他实体的关系都收集起来，并加入一个辨别属性。这种方法可帮助查询所有子类的信息。例如，当验证登录合法性时，仅需要用简单的 SQL 语句表查询超类表便可以完成这一操作而无需考虑用户的类型（老师或学生）。但是这种方法将不可避免地造成存储空间的浪费；此外，并非表中的所有字段都是有意义的（例如，对于老师的记录，班级字段并无意义）。

（2）只把子类转换成表

只把子类转换成互不相关的表，而超类中的属性及其他实体的关系都由各个表收集。这种方法是最容易理解以及编码的。但超类更新困难，超类的改动将引起所有子类生成表的改动，同时查询所有用户信息的操作将变得很困难。

（3）一对一关联

将子类与超类都转移成表，并在它们之间建立一对一的关联。这种方法概念清晰，超类更新与另一种方法一样，同样可以查询超类及其所有子类的信息，但不得不进行连接操作，这使得查询速度较慢。同时当进行更新时需要涉及一系列的数据表。由于这种方法提供了清晰的概念，在原型系统中采用了这种方法。

下面是原型系统中所包含的数据表（程序 10-2-1）：

程序 10-2-1：

StudentAccount 表：

```
SID          numeric,     学生 ID
Username     varchar(15),
Password     varchar(15),
StudentNO    varchar(15),  学号
Name         varchar(16),
School       varchar(40),Username     varchar(15),
Department   varchar(40),Password     varchar(15),
Class        varchar(40)StudentNO     varchar(15),  学号
Name         varchar(16),
School       varchar(40),
Department   varchar(40),
Class        varchar(40)
```

TeacherAccount 表：

```
TID          numeric,
Username     varchar(15),
Password     varchar(15),
```

```
Name        varchar(16),
School      varchar(40),
Department  varchar(40),
Email       varchar(40),
```

Logic 表：

```
QID         numeric,
Chapter     int,
Segment     int,
Difficulty  int,
Content     varchar(255),
Answer      int
```

Single 表：

```
QID             numeric,
Chapter     int,
Segment     int,
Difficulty  int,
Content     varchar(255),
AnswerA     varchar(255),
AnswerB     varchar(255),
AnswerC     varchar(255),
AnswerD     varchar(255),
Answer      int
```

Multiply 表：

```
QID             numeric,
Chapter     int,
Segment     int,
Difficulty  int,
Content     varchar(255),
AnswerA     varchar(255),
AnswerB     varchar(255),
AnswerC     varchar(255),
AnswerD     varchar(255),
AnswerE     varchar(255),
Answer      int  1代表A,2代表B,4代表C,8代表D,16代表E
```