제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

닷넷 기반에서 Iterator Pattern을 이용한 데이터 접속 객체의 구현

남석모*, 김상훈**, 정정수***
*동명정보대학교 컴퓨터공학과, **동명정보대학교 정보기술원
***동명정보대학교 컴퓨터공학과 부교수
e-mail:omkus@msn.com

Implementation of Data Connection Object Using Iterator Pattern in .NET Base

Suk-Mo Nam*, Sang-Hoon Kim**, Jung-Sue Jung***
*Dept. of Computer Engineering, Tongmyong University
**Institute of Information Technology, Tongmyong University
***Dept. of Computer Engineering, Tongmyong University

요 약

최근 소프트웨어 개발에서의 핵심어인 객체지향(object oriented)은 1970년대 초부터 중요하게 인식되기 시작했다. 현재는 객체지향에 대한 많은 관련 연구를 통하여 그 활용범위가 증대되고 있다. 본 연구에서는 객체지향 연구의 한 부분인 디자인 패턴 중에서 많이 이용되는 Iterator(Cursor) Pattern을 활용하여 닷넷(.NET) 기반에서의 특정 비즈니스를 위한 데이터 접속 객체를 연구하였다.

1. 서론

최근의 소프트웨어 개발 환경은 객체지향을 많이 선호하고 있다. 이와 관련하여 객체지향 프로그래밍, 객체지향 모델링, 객체지향 방법론 등이 객체지향 연구와 밀접한 관계를 가지고 있다. 객체지향 패러다임은 수십 년에 걸쳐 학문적으로 연구되어 왔으며, 그 결과로 OOAD(Object Oriented Analysis & Design), 개발 방법론의 객체지향에 대한 적용, 디자인 패턴 등이 연구 발표되었다.

GoF라고 불리는 4명의 개발자가 정리 발표한 디자인 패턴은 객체지향에 관한 수많은 연구 중에서 활용 사례가 많은 한 분야이다. 4명의 개발자는 시스템을 설계하면서 많은 작업들이 비슷한 패턴으로 공통적인 구조를 가진다는 것을 발견했으며, 이러한 공통적인 패턴을 크게 3가지(생성, 구조, 행위)로 구분지어 디자인 패턴이라 명명했다.

과거의 디자인 패턴은 시스템 개발뿐만 아니라 특정 프로그램 언어의 API나 프레임워크(framework)의 제작에 사용되었으며, 현재에도 많은 시스템 개발에 적용되고 있다. 특히 e-Biz(Internet Business)에서 많이 사용되는 플랫폼인 자바에서 제공하는 클래스 라이브러리와 컴포넌트 모델인 EJB(Enterprise Java Been)는 디자인 패턴을 활용하고 있으며, 마이크로소프트 역시 클래스 라이브러리와 서비스에서 디자인 패턴을 활용하고 있다.

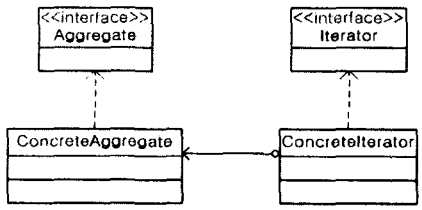
2000년에 등장한 마이크로소프트의 닷넷 프레임워크는 데이터베이스에 접속하기 위한 객체들의 집합인 ADO.NET을 제공한다. ADO.NET에는 데이터베이스에서 패치되는 결과 집합 데이터를 클라이언트의 캐시에 저장하거나 스트림으로 반환하는 여러 객체들이 포함되어 있고, 이러한 데이터 접속 객체들은 마이크로소프트의 웹 개발 도구인 ASP.NET이 제공하는 특정한 데이터 바인딩 컨트롤에 데이터 바인딩의 형태로 프리젠테이션 할 수 있다.

본 연구에서는 가장 많이 활용되는 디자인 패턴중의 하나인 Iterator Pattern을 이용하여 닷넷 기반에서의 특정 비즈니스 전용 데이터 접속 객체를 구현하는 방법을 논하고, 이에 따라 작성된 데이터 접속 객체는 닷넷에서 제공하는 데이터바인딩 컨트롤이나 닷넷 기반의 사용자 정의 컨트롤에 데이터바인딩의 형태로 프리젠테이션 될 수 있도록 구현하였다.

2. Iterator Pattern

GoF에 의한 디자인 패턴 분류에 따르면 Iterator Pattern은 어떤 처리의 책임을 어느 객체에 할당하는 것이 좋은지, 알고리즘은 어느 객체에 정의하는 것이 좋은지 등에 대한 설계 패턴들을 정의하는 행위 패턴(Behavioral Patterns)으로 분류된다. Iterator Pattern은 내부 표현 방법을 캡슐화(encapsulation)하고 복합 객체의 원소를 순차적으로 접근할 수 있는 객체에 대한 설계 방법이다[1],[2].

Iternator Pattern의 정의에 따르면 원소를 순차적으로 접근하기 위한 컴포넌트를 작성하기 위해서는 4개의 객체가 필요하다. 첫 번째, 반복적으로 표현되는 각각의 요소를 표현하는 오퍼레이션을 가진 인터페이스 Iterator. 두 번째, 집합으로 가지고 있는 각각의 요소들을 열거자형으로 반환하는 오퍼레이션을 가진 인터페이스 Aggregate. 세 번째, Iterator 인터페이스를 실체화한 Concrete Class ConcreteIterator. 네 번째, Aggregate 인터페이스를 실체화하며 집합에 포함될 각각의 객체를 포함하고 제거하는 행위를 포함하는 ConcreteAggregate 객체이다. 그림 1은 객체 구조를 UML(Unified Modeling Language) 다이어그램으로 나타낸 것이다.



(그림 1) 객체 구조의 UML 다이어그램

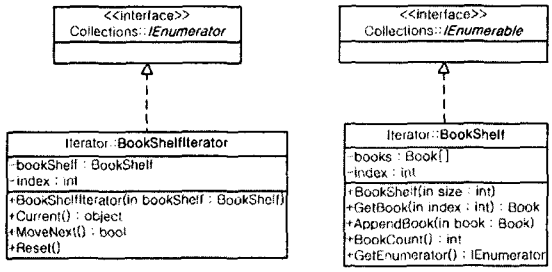
Iterator 인터페이스는 열거할 요소들을 반복하는 인터페이스를 결정하는 역할을 한다. Iterator 인터페이스는 다음 요소가 존재하는지의 여부를 알기 위한 오퍼레이션과 다음 요소를 반환하는 오퍼레이션을 가지고 있어야 한다. Aggregate 인터페이스는 저장하고 있는 요소들을 Iterator 형식으로 반환할 수 있는 오퍼레이션을 가지고 있어야 한다. 각 인터페이스를 실체화하는 객체들은 각 인터페이스가 가지고 있는 오퍼레이션을 요구에 맞게 구현하는 코드들을 가지고 있어야 한다.

Iterator 패턴을 활용하여 객체를 구성하면 내부 요소들의 표현 방식에 상관없이 같은 방식으로 집합 객체의 요소들을 순회할 수 있게 되며, 집합 객체를 순회하는 다양한 방법들을 제시할 수 있게 되고, 서로 다른 집합 객체 구조에 대해서도 동일한 방법으로 순회될 수 있는 등의 여러 장점이 있다.

대부분의 클래스 라이브러리에 정의된 Collection 클래스는 한 가지 이상의 방법으로 Iterator를 제공한다.

3. 닷넷 기반의 Iterator Pattern의 구현

닷넷 프레임워크는 API들의 집합을 클래스 라이브러리 형태로 제공하며, Collections Framework를 가지고 있다. 즉, Iterator 형식에 사용될 수 있는 인터페이스 IEnumerator와 IEnumerable을 제공한다. 인터페이스 IEnumerator는 Iterator Pattern의 역할에서 Iterator로 사용되며, 인터페이스 IEnumerable은 Aggregate 역할로 사용된다[3]. 그림 2는 닷넷 프레임워크에서 Collections Framework의 API를 재구성한 것을 보여준다.



(그림 2) Collections Framework의 API 재구성

그림 2의 Iterator 역할을 하는 BookShelfIterator 객체는 다음과 같다. 다음 요소의 존재를 확인하기 위한 MoveNext 메소드, 현재 요소를 반환하기 위한 Current 프로퍼티, 요소 인덱스의 초기화를 위한 Reset() 메소드를 구현할 것을 닷넷 프레임워크의 인터페이스 IEnumerator에서 강제한다.

BookShelfIterator 클래스의 MoveNext 메소드는 0으로 초기화된 클래스의 전역변수 index에 접근하여 읽을 데이터의 존재여부를 그림 3처럼 알아낸다.

```
public bool MoveNext() {
    if (this.index < this.bookShelf.BookCount)
        return true;
    else
        return false;
}
```

(그림 3) MoveNext 메소드

그림 4에서 Current 프로퍼티는 index 변수에 해당하는 요소를 현재 요소로 인식하여 반환하고 index 변수의 값을 1 증가시킨다.

```
public object Current {
    get { Book book =
              this.bookShelf.GetBook(this.index);
          this.index++; return book; }
}
```

(그림 4) Current 프로퍼티

그림 5에서 Reset 메소드는 index 변수의 값을 0으로 설정하여 현재 요소를 집합의 0번째 요소로 설정한다.

```
public void Reset() {
    this.index = 0;
}
```

(그림 5) Reset 메소드

ConcreateAggregate의 역할을 하는 BookShelf 클래스의 구현은 다음과 같으며, 현재 요소를 반환하는 메소드는 그림 6과 같이 구현된다.

```
public Book GetBook(int index) {
    return this.books[index];
}
```

(그림 6) 요소 반환 메소드

집합에 요소를 더하는 메소드는 그림 7과 같이 구현된다.

```
public void AppendBook(Book book) {
    this.books[this.index] = book; this.index++;
}
```

(그림 7) 요소를 더하는 메소드

집합의 요소 수를 반환하는 메소드는 그림 8과 같다.

```
public int BookCount {
    get { return this.index; }
}
```

(그림 8) 요소 수 반환 메소드

그림 9는 IEnumerator 인터페이스의 실체화 메소드인 GetEnumerator 메소드의 구현 결과이다.

```
public IEnumerator GetEnumerator() {
    return new BookShelfIterator(this);
}
```

(그림 9) GetEnumerator 메소드 구현

위와 같이 객체는 요소의 집합 형태나 구현에 상관없이 BookShelfIterator 객체의 MoveNext 메소드와 Current 프로퍼티를 사용하여 반환할 수 있다.

```
BookShelf bookShelf = new BookShelf(2);
bookShelf.AppendBook(new Book("디자인 패턴"));
bookShelf.AppendBook(new Book("객체지향"));
BookShelfIterator bookShelfIterator =
            new BookShelfIterator(bookShelf);
while(bookShelfIterator.MoveNext()) {
            Book book =
           (Book)bookShelfIterator.Current;
            Console.WriteLine(book.Name);
}
```

(그림 10) 응용 예제

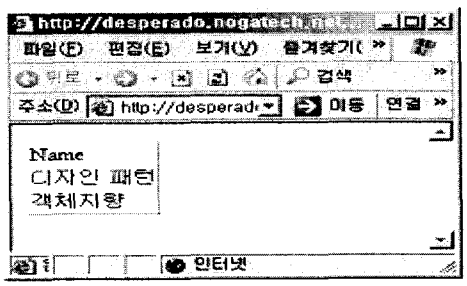
그림 10의 응용에서 알 수 있듯이 요소를 저장하는 집합이 ArrayList, 사용자 정의 Collection 또는 집합인지의 여부에 상관없이 Add메소드로 집합에 데이터를 더할 수 있고 또한 MoveNext 메소드와 Current 프로퍼티로 읽을 수 있다.

BookShelf 클래스의 집합에 포함되는 Book 클래스가 자신의 멤버들을 프로퍼티 형태로 포함하고 있다면, Book 객체는 ASP.NET의 데이터바인딩 컨트롤에 직접 바인딩 되어 데이터를 표현할 수 있다.

ASP.NET 페이지를 그림 11과 같이 작성하여 BookShelf 하면 DataGrid 컨트롤은 객체를 직접 바인딩 한다.

```
<%@ Import Namespace="Iterator" %>
<%@ Import Namespace="System.Collections" %>
<Script Runat="Server">
void Page_Load(Object Sender, EventArgs e) {
    BookShelf books = new BookShelf(2);
    books.AppendBook(new Book("디자인 패턴"));
    books.AppendBook(new Book("객체지향"));
    myGrid.DataSource = books;
    myGrid.DataBind();
}
</Script>
<asp:DataGrid id="myGrid" Runat="Server" />
```

(그림 11) 객체의 바인딩



(그림 12) 컨트롤에 바인딩된 Iterator 개체

4. 닷넷 기반의 Iterator Pattern을 활용한 데이터 접속 객체의 구현

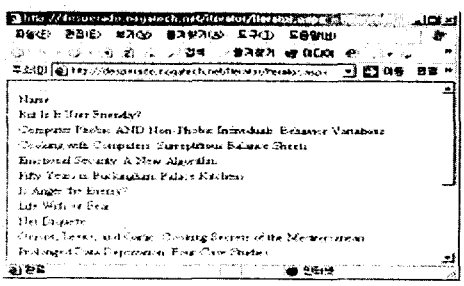
Iterator Pattern을 활용하여 데이터 접속 개체를 생성하는 방법은 BookShelf 클래스는 Book 개체를 생성하여 BookShelf 클래스에 포함된 요소 배열에 저장한다. 그러므로 BookShelf 클래스에서 Book개체를 생성할 때 데이터베이스에 접속하여 데이터베이스에서 반환받은 데이터들로 Book 개체를 생성하는 것이다.

BookShelf 클래스 생성자에서 데이터베이스에 액세스하여 요소 집합에 저장될 데이터의 갯수를 알아내어 배열을 초기화 한 다음, 초기화된 배열의 개수만큼 반환하여 해당 인덱스에 새 Book 개체를 할당하게 되며, 구현은 그림 13과 같다.

```
public BookShelf()
 SqlConnection mConn = new SqlConnection(...);
 string sql = "Select Count(*) from titles";
 SqlCommand myCom =
   new SqlCommand(sql, mConn);
 // 데이터의 개수를  index 전역 변수에 할당
 // ...
 this.books = new Book[this.index];
  // 실제 저장되는 데이터를 패치
  // 반복하여 배열에 저장
  sqlQuery = "Select title from titles";
  //...
  SqlDataReader myReader =
     myCommand.ExecuteReader (...)
  for (int i = 0; myReader.Read(); ++i)
    this.books[i] =
     new Book(myReader["title"].ToString());
```

(그림 13) 데이터 접속 객체 구현

그림 13과 같이 구현한 데이터 접속 객체는 ASP.NET의 데이터그리드 컨트롤에 직접 바인딩 된다. 그림 14는 생성한 데이터 접속 객체를 데이터그리드 컨트롤에 바인딩 결과이다.



(그림 14) 데이터 접속 객체의 바인딩 결과

5. 결론

쿼리를 이용하여 데이터를 클라이언트의 캐쉬에 유지하는 객체나 데이터의 스트림을 반환하는 객체는 모든 형태의 데이터 접속에 사용될 수 있도록 일반화되어 있다. 이것은 특정 비즈니스에 알맞게 동작하는 객체를 생성하거나 여러 요구에 맞는 다양한 비즈니스 객체를 생성하여야 할 때 Business-Logic 계층에서 캡슐화되어 동작하는 객체로 생성될 수 있다.[4]

데이터 접속 객체는 데이터베이스에 두 번 액세스하기 때문에 효율이 저하될 수 있다. 그것을 방지하기 위하여 ArrayList를 사용할 수 있지만 ArrayList 객체는 너무 많은 역할을 구현하고 있어 효율 저하의 원인이 될 수 있고[5] 또한 자신이 IEnumerable 객체의 서브타입이므로 적당하지 않다.

e-Biz 등으로 인한 특정 데이터 접속 컴포넌트의 필요성이 대두되고 있고, 웹상에서 보다 편리한 방법의 프로그래밍 작성방법이 필요한 지금 Iterator Pattern을 이용한 데이터 접속객체의 사용은 솔루션의 개발을 구조적으로 만들어주고 또한 사용과 유지보수가 편리한 환경을 구성해주는 방법이 될 수 있을 것이다.

데이터 접속 객체의 효율향상에 관련된 내용은 이 연구범위에서 벗어나므로 차후 연구해 보아야 할 과제로 남겨두기로 한다.

참고문헌

[1] Design Patterns - Elements of Reuseable Object-Oriented Software, Addison Wesley, Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995

[2] http://st-www.cs.uiuc.edu/users/patterns/DPBook/DPBook.html

[3] Introduction to Design Patterns in C#, IBM T J Watson Research Center, James W Cooper, 2002

[4] A Compositional Collections Component Framework, Aurora Borealis Software, Yahya H. Mirza, 2002

[5] Practical Java, Addison-Weasley, Peter Hagger, 1998