# GENERIC, COLLECTION, LINQ

# GENERIC

## Định nghĩa

Generic cho phép viết một lớp hoặc phương thức mà có thể làm việc với bất kỳ kiểu dữ liệu nào, generic cũng là một kiểu dữ liệu trong C#, nó cũng như là int, float, string, bool,... nhưng điểm khác biệt ở đây, generic nó là một kiểu dữ liệu "tự do", nghĩa là nó có thể là kiểu dữ liệu nào cũng được, tùy vào mục đích sử dụng, nó như là 1 kiểu đại diện cho tất cả các kiểu dữ liệu còn lại.

## Khái báo generic

Khai báo phương thức có tên MyFunction, sau tên này có ký hiệu <X, Y> có nghĩa phương thức này có sử dụng hai kiểu là kiểu X và kiểu Y (tất nhiên nó chưa cụ thể là kiểu gì, nó chỉ cụ thể khi phương thức được gọi). Khi đã có kiểu Generic rồi thì dùng kiểu này cho các thành phần của phương thức - như kiểu trả về là X, tham số là kiểu X và Y, trong thân phương thức tương tự có thể khai báo sử dụng kiểu X và kiểu Y.

int a = 1;

string b = 2;

int rs = MyFunction<int, string>(a, b); // Phương thức chạy với vai trò X là kiểu int, Y là kiểu string.

double c = 2;

MyFunction<double, int>(c, a); // Phương thức chạy với X là kiểu double, Y kiểu int

## Khi nào nên sử dụng generic

Nếu có sự trùng lặp code về mặt logic và cách xử lý dữ liệu, chỉ khác biệt về kiểu dữ liệu.

Nếu lúc xây dựng class chưa xác định được kiểu dữ liệu của các biến thành viên, thuộc tính hoặc biến cục bộ (của phương thức) thì cần sử dụng lập trình generic

## Generic method

|  |  |
| --- | --- |
|  |  |
| Before: a = 1, b = 2  After : a = 2, b = 1 | type = System.Int32  Before: a = 1, b = 2  After : a = 2, b = 1  type = System.Boolean  Before: a = True, b = False  After : a = False, b = True |

Giả sử bạn cần hoán đổi giá trị của hai biến kiểu bool, sẽ phải viết thêm một phương thức có code tương tự, chỉ thay duy nhất int bằng bool. Nếu muốn hoán đổi hai biến kiểu char (ký tự), lại phải viết thêm một phương thức nữa tương tự. Rõ ràng ở đây các phương thức có cùng một logic, chỉ khác biệt duy nhất kiểu dữ liệu mà nó xử lý. Trong tình huống này, generics là một lựa chọn hợp lý giúp chống lặp code.

## Generic classes

Generic classes đóng gói các xử lý mà không chỉ định rõ kiểu dữ liệu. Hầu hết các trường hợp phổ biến sử dụng generic classes là với collections giống như: danh sách liên kết (Linked List), hash tables, queues, trees,..

|  |
| --- |
| class Demo<T>  {  T value;  public Demo(T t)  {  value = t;  }  public void Write()  {  Console.WriteLine(value);  }  }  class Program  {  static void Main()  {  Demo<int> test1 = new Demo<int>(10);  test1.Write();  Demo<string> test2 = new Demo<string>("Cat");  test2.Write();  Console.ReadLine();  }  } |
| 10  Cat |

## Generic Interface

Nó thường hữa ích để định nghĩa cho collection classes, hoặc cho generic classes.

Giả sử ta xây dựng một chức năng getAll và save data của đối tượng Book và Author xuống DB. Chúng ta sẽ sử dụng một base interface và triển khai

|  |
| --- |
| class Book  {  public string Name { get; set; }  public int Page { get; set; }  }  class Author  {  public string Name { get; set; }  public int Age { get; set; }  }  interface IBaseRepository<T>  {  List<T> getAll();  T Save(T item);  }  class BaseRepository<T> : IBaseRepository<T>  {  public List<T> getAll()  {  return new List<T>();  }  public T Save(T item)  {  return item;  }  }  class Program  {  static void Main()  {  Console.OutputEncoding = Encoding.UTF8;  BaseRepository<Book> bookRepository = new BaseRepository<Book>();  BaseRepository<Author> authorRepository = new BaseRepository<Author>();  Book book = bookRepository.Save(new Book { Name = "Truyện Kiều", Page = 1000 });  Author author = authorRepository.Save(new Author { Name = "Nguyễn Du", Age = 150 });  Console.WriteLine("Book: {0} - page: {1}", book.Name, book.Page);  Console.WriteLine("Name: {0} - age: {1}", author.Name, author.Age);  Console.ReadLine();  }  } |
| Book: Truyện Kiều - page: 1000  Name: Nguyễn Du - age: 150 |

# COLLECTION

## ArrayList

Là một lớp thuộc collection dùng để lưu trữ một mảng, truy xuất phần tử qua index giống như mảng,tuy nhiên tiện lợi hơn mảng do có hỗ trợ các phương thức thêm, xóa, sửa phần tử... và thay đổi kích thước mảng.

1 số thuộc tính và phương thức của ArrayList:

* + Count: Trả về số phần tử hiện có của ArrayList
  + Add(Object): Thêm một phần tử vào cuối ArrayList
  + AddRange(ICollection): Thêm danh sách phần tử vào cuối ArrayList
  + Clone(): tạo bản sao cho ArrayList
  + Insert(int Index, object Value): Thêm một phần tử vào vị trí index
  + Remove(object Value): Xóa phần tử đầu tiên xuất hiện trong ArrayList
  + Clear(): Xóa tất cả các phần tử trong ArrayList.

|  |  |
| --- | --- |
| ArrayList arrayList1 = new ArrayList();  ArrayList arrayList2 = new ArrayList();  arrayList1.Add("Item 1");  arrayList1.Add("Item 2");  arrayList1.Add("Item 3");  arrayList1.Add("Item 4");  arrayList2.Add("Item ArrayList 2");  arrayList1.AddRange(arrayList2);  arrayList1.Insert(2, "New Item");  for (var i = 0; i < arrayList1.Count; i++)  {  Console.WriteLine(arrayList1[i]);  }  Console.ReadLine(); | Item 1  Item 2  New Item  Item 3  Item 4  Item ArrayList 2 |

## HashTable

Là một lớp thuộc collection trong c#, lưu trữ tập dữ liệu với mỗi phần tử là một cặp key-value. Có thể truy xuất các phần tử qua key. HashTable hỗ trợ tìm kiếm phần tử qua key rất nhanh, có thể ứng dụng để lưu trữ dữ liệu từ điển...

Một số thuộc tính và phương thức phổ biến trong HashTable:

* Count: Số lượng phần tử hiện có trong Hashtable.
* Keys: Danh sách các key của phần tử trong Hashtable.
* Values: Danh sách các value của phần tử trong Hashtable.
* Add(Object, Object): Thêm một phần tử với cặp key-value vào Hashtable.
* Remove(Object): Xóa một phần tử có key xuất hiện đầu tiên.
* Clone(): Tạo một bản sao từ Hashtable
* ContainsKey(Object): Kiểm tra key có tồn tại trong danh sách key của Hashtable hay không.
* ContainsValue(Object): Kiểm tra value có tồn tại trong danh sách value của Hashtable hay không.

|  |  |
| --- | --- |
| Hashtable hashtable = new Hashtable();  hashtable.Add("Key1", "Hello");  hashtable.Add("Key2", "Everyone");  Console.WriteLine(hashtable["Key1"]);  foreach (DictionaryEntry item in hashtable)  {  Console.WriteLine("Key: {0} - Value:  {1}", item.Key, item.Value);  }  foreach (var key in hashtable.Keys)  {  Console.WriteLine("Key: {0} ", key);  }  Console.ReadLine(); | Hello  Key: Key2 - Value: Everyone  Key: Key1 - Value: Hello  Key: Key2  Key: Key1 |

## SortedList

Là một collection lưu trữ dữ liệu dưới dạng key-value. SortedList là sự kết hợp giữa ArrayList và Hashtable. Các phần tử được sắp xếp theo key. Có thể truy xuất phần tử qua key giống Hashtable và qua index giống ArrayList.

Một số thuộc tính và phương thức:

* Count: Trả về số phần tử hiện có trong SortedList.
* Keys: Trả về danh sách key trong SortedList.
* Values: Trả về danh sách value trong SortedList.
* Add(Object, Object): Thêm một phần tử với key-value vào SortedList.
* GetByIndex(Int32): Trả về giá trị phần tử tại vị trí index.
* SetByIndex(Int32, Object): Thay thế giá trị cho phần tử tại vị trí index.
* ContainsKey(Object): Kiểm tra key có tồn tại trong SortedList hay không.
* ContainsValue(Object): Kiểm tra value có tồn tại trong SortedList hay không.
* Clone(): Tạo ra một bản sao của SortedList.

|  |  |
| --- | --- |
| SortedList sortlist = new SortedList();  sortlist.Add("3", "!");  sortlist.Add("2", "World");  sortlist.Add("1", "Hello");  Console.WriteLine("Demo Program");  Console.WriteLine("Count: {0}", sortlist.Count);  Console.WriteLine("Capacity: {0}", sortlist.Capacity);  Console.WriteLine("Keys and Values:");  Console.WriteLine("\t-KEY-\t-VALUE-");  for (int i = 0; i < sortlist.Count; i++)  {  Console.WriteLine("\t{0}:\t{1}",sortlist.GetKey(i),  sortlist.GetByIndex(i));  }  Console.ReadLine(); | Demo Program  Count: 3  Capacity: 16  Keys and Values:  -KEY- -VALUE-  1: Hello  2: World  3: ! |

Khi insert 3 phần tử với Key lần lược là 3, 2 và 1 thì SortedList sẽ tự động sắp xếp lại theo key và kết quả sau sắp xếp với thứ tự là: 1, 2, 3.

## Stack

Stack hay còn gọi là ngăn xếp là một cấu trúc dữ liệu hoạt động theo nguyên lý LIFO (Last in first out). Nghĩa là phần tử vào sau cũng sẽ được sắp lên trên, khi lấy ra sẽ lấy phần tử trên cùng ra.

Một số thuộc tính và phương thức:

* Count: Số phần tử hiện có trong stack.
* Push(Object): Thêm một phần tử vào trí trên cùng của stack.
* Pop(): trả về phần tử được thêm vào cuối cùng và xóa phân tử đó khỏi stack.
* Peek(): trả về phần tử được thêm vào cuối cùng nhưng không xóa phần tử đó khỏi stack.

|  |  |
| --- | --- |
| Console.OutputEncoding = Encoding.UTF8;  Stack stack = new Stack();  stack.Push(" Xin chào");  stack.Push(" mọi người");  stack.Push("ahihihaa");  Console.WriteLine("Đây là chương trình demo");  Console.WriteLine("Count: {0}", stack.Count);  Console.Write("Values: ");  foreach (Object obj in stack)  Console.Write("{0}", obj);  Console.ReadLine(); | Đây là chương trình demo  Count: 3  Values: ahihihaa mọi người Xin chào |

## Queue

Là cấu trúc dữ liệu hoạt động theo nguyên lý FIFO (First In First Out). Nghĩa là phần tử được thêm vào đầu tiên sẽ được lấy ra đầu tiên.

Một số thuộc tính và phương thức:

* Count: Số phần tử hiện có trong Queue.
* Enqueue(Object): Thêm một phần tử vào cuối Queue.
* Dequeue(): trả về phần tử đầu tiên và xóa phân tử đó khỏi Queue.
* Peek(): trả về phần tử đầu tiên nhưng không xóa phần tử đó khỏi Queue.
* Contains(Object): Kiểm tra đối tượng có tồn tại trong Queue không.

|  |  |
| --- | --- |
| Console.OutputEncoding = Encoding.UTF8;  Queue queue = new Queue();  queue.Enqueue("Xin Chào");  queue.Enqueue(" mọi người");  queue.Enqueue(" ahihi");  Console.WriteLine("Đây là chương trình demo collection Queue (FIFO)");  Console.WriteLine("Count: {0}", queue.Count);  Console.Write("Values: ");  foreach (Object obj in queue)  Console.Write("{0}", obj);  Console.ReadLine(); | Đây là chương trình demo collection Queue (FIFO)  Count: 3  Values: Xin Chào mọi người ahihi |

## List

List là một Generic collection giúp lưu trữ và quản lý mảng. Có thể truy xuất các phần tử của mảng qua vị trí index. List tương tự như ArrayList nhưng cải tiến hơn.

Sử dụng List an toàn hơn ArrayList bởi vì List lưu trữ một danh sách các phần tử với kiểu cố định, không như ArrayList lưu trữ danh sách object đôi khi biên dịch không xảy ra lỗi nhưng lúc thực thi sẽ xảy ra lỗi nếu các phần tử có nhiều kiểu dữ liệu khác nhau.

|  |  |
| --- | --- |
| Console.OutputEncoding = Encoding.UTF8;  List<string> list1 = new List<string>();  list1.Add("Xin Chào!");  list1.Add(" Bạn thứ 1");  list1.Add(" Bạn thứ 2");  string result = null;  foreach (string key in list1)  {  result += key;  }  Console.WriteLine(result);  Console.ReadLine(); | Xin Chào! Bạn thứ 1 Bạn thứ 2 |

# LINQ

## Định nghĩa

(Truy vấn tích hợp ngôn ngữ) là cú pháp truy vấn thống nhất trong C# và VB.NET để truy xuất dữ liệu từ các nguồn và định dạng khác nhau. Nó khả năng truy cập các nguồn dữ liệu khác nhau (SQL Db, XML, List ...) với cùng cú pháp. LINQ hoạt động trên những kiểu tập hợp có khả năng duyệt qua nó (xem thêm Collection, List trong C#). Để sử dụng LINQ thì nạp hai thư viện Generic và Linq.

Để sử dụng LINQ cần có ba thành phần: nguồn dữ liệu (data source), truy vấn (query), lời gọi thực hiện truy vấn (query execution).

## Nguồn dữ liệu

LINQ cung cấp cho chúng ta những providers khác nhau để có thể truy vấn đến những nguồn dữ liệu khác nhau

|  |  |
| --- | --- |
| **Tên provider** | **Mô tả** |
| LINQ to Objects | Sử dụng LINQ đối với các đối tượng collection mà implement từ IEnumerable hoặc IEnumerable<T> (dữ liệu được lưu trong bộ nhớ). Được sử dụng rộng rãi đặc biệt đối với những bài toán cần hiệu năng cao. |
| LINQ to SQL | Thực hiện map các tables, views, store procedures thành các đối tượng. LINQ sẽ thực hiện truy vấn trên các đối tượng đó bằng cách chuyển đổi qua lại giữa đối tượng và câu lệnh sql.  Ngoài truy vấn ta cũng có thể thực hiện thêm/sửa/xóa dữ liệu dựa trên các đối tượng trên.  Hỗ trợ transaction.  Ưu điểm: được sử dụng khá nhiều trong thực tế dưới cái tên Entity Framework hoặc Entity Framework Core. Nhược điểm: chỉ làm việc với cơ sở dữ liệu là SQL Server. |
| LINQ to Entities | Tương tự như LINQ to SQL nhưng hỗ trợ nhiều loại cơ sở dữ liệu.  Nhược điểm: sử dụng phức tạp. Nhiều cơ sở dữ liệu không thích hợp sử dụng chung với .Net. |
| LINQ to DataSets | Sự kết hợp giữa LINQ và ADO.NET. |
| LINQ to XML | Truy vấn thông tin trong file XML. |

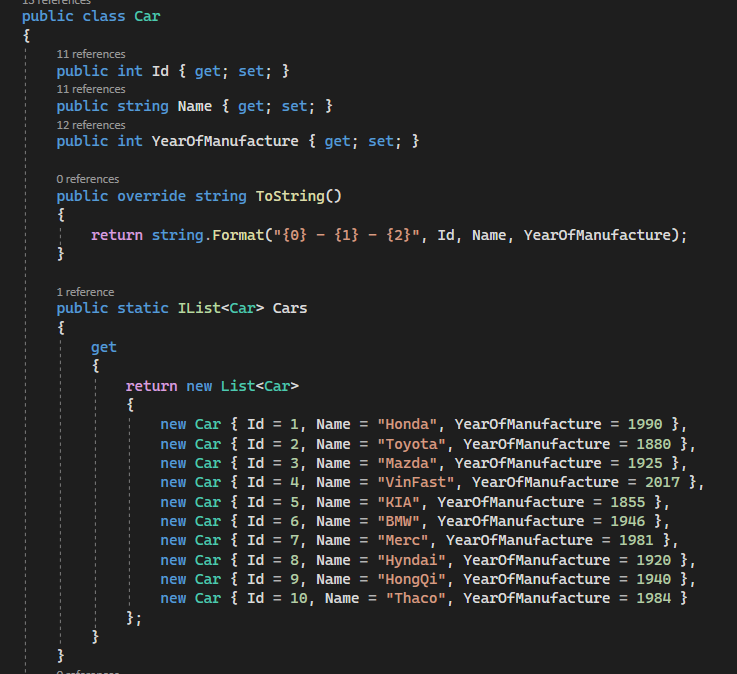
## Truy vấn LINQ

Có hai cách viết cho LINQ là cú pháp truy vấn (query syntax) và cú pháp phương thức (method syntax).

Cú pháp truy vấn (Query Syntax): Cú pháp này nhìn giống như một truy vấn select SQL đảo ngược với từ khóa đầu tiên là from, kết thúc là select. Cú pháp này có hình thức khác biệt với code C# thông thường với một số từ khóa mới.

Cú pháp phương thức (Method Syntax, còn gọi là Fluent): giống như cách gọi một phương thức mở rộng bình thường trên object của class. Đây lối viết cơ bản của LINQ.

Cú pháp pha trộn (mixed syntax): thực tế là một số phương thức LINQ không được hỗ trợ ở dạng query syntax, khi đó người ta có thể sử dụng lối viết pha trộn của cả hai loại cú pháp.



### Query Syntax

|  |
| --- |
| var lists = from <Biến lưu thông tin từng phần tử> in <Nguồn dữ liệu>  [<Phép toán truy vấn: where, join ... in, order by...> Biều thức lambda]  select <Biến lưu thông tin từng phần tử> |

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  var cars = Car.Cars;  var list = from item in cars where item.YearOfManufacture >= 1990 select item;  foreach (var item in list)  {  Console.WriteLine(item);  }  Console.ReadKey();  }  } |
| 1 - Honda - 1990  4 - VinFast - 2017 |

### Method Syntax

Là những phương thức mở rộng của IEnumerable hoặc IEnumerable<T>.

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  var cars = Car.Cars;  var list = cars.Where(f => f.YearOfManufacture >= 1990);  foreach (var item in list)  {  Console.WriteLine(item);  }  Console.ReadKey();  }  } |
| 1 - Honda - 1990  4 - VinFast - 2017 |

public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);

### Phân loại toán tử truy vấn chuẩn trong Linq

|  |  |
| --- | --- |
| Nhóm | Phương thức |
| Lọc dữ liệu | Where, OfType |
| Sắp xếp | OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse |
| Nhóm | GroupBy, ToLookup |
| Join | GroupJoin, Join |
| Projection | Select, SelectMany |
| Tổng hợp | Aggregate, Average, Count, LongCount, Max, Min, Sum |
| Định lượng | All, Any, Contains |
| Phần tử | ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault |
| Tập hợp | Distinct, Except, Intersect, Union |
| Phân vùng | Skip, SkipWhile, Take, TakeWhile |
| Liên kết | Concat |
| So sánh bằng | SequenceEqual |
| Tạo danh sách | DefaultEmpty, Empty, Range, Repeat |
| Chuyển đổi | AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList |