# 这里通过一个泛型类来实现一个链表,这个泛型类里面有一个嵌套类

泛型类和泛型方法同时具备可重用性、类型安全和效率，这是非泛型类和非泛型方法无法具备的。泛型通常用与集合以及作用于集合的方法一起使用。.NET Framework 2.0 版类库提供一个新的命名空间 [System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx)，其中包含几个新的基于泛型的集合类。建议面向 .NET Framework 2.0 及更高版本的所有应用程序都使用新的泛型集合类，而不要使用旧的非泛型集合类如 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx)。有关更多信息，请参见 [.NET Framework 类库中的泛型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ac30fak4.aspx)。

当然，也可以创建自定义泛型类型和方法，以提供自己的通用解决方案，设计类型安全的高效模式。下面的代码示例演示一个用于演示用途的简单泛型链接列表类。（大多数情况下，应使用 .NET Framework 类库提供的 [List<T>](https://msdn.microsoft.com/zh-cn/library/6sh2ey19.aspx) 类，而不是自行创建类。）在通常使用具体类型来指示列表中存储的项的类型的场合，可使用类型参数 T。其使用方法如下：

* 在 **AddHead** 方法中作为方法参数的类型。
* 在 **Node** 嵌套类中作为公共方法 **GetNext** 和 **Data** 属性的返回类型。
* 在嵌套类中作为私有成员数据的类型。

注意，T 可用于 **Node** 嵌套类。如果使用具体类型实例化 GenericList<T>（例如，作为 GenericList<int>），则所有的 T 都将被替换为 int。

// type parameter T in angle brackets

public class GenericList<T>

{

// The nested class is also generic on T.

private class Node

{

// T used in non-generic constructor.

public Node(T t)

{

next = null;

data = t;

}

private Node next;

public Node Next

{

get { return next; }

set { next = value; }

}

// T as private member data type.

private T data;

// T as return type of property.

public T Data

{

get { return data; }

set { data = value; }

}

}

private Node head;

// constructor

public GenericList()

{

head = null;

}

// T as method parameter type:

public void AddHead(T t)

{

Node n = new Node(t);

n.Next = head;

head = n;

}

public IEnumerator<T> GetEnumerator()

{

Node current = head;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

}

下面的代码示例演示客户端代码如何使用泛型 GenericList<T> 类来创建整数列表。只需更改类型参数，即可方便地修改下面的代码示例，创建字符串或任何其他自定义类型的列表：

class TestGenericList

{

static void Main()

{

// int is the type argument

GenericList<int> list = new GenericList<int>();

for (int x = 0; x < 10; x++)

{

list.AddHead(x);

}

foreach (int i in list)

{

System.Console.Write(i + " ");

}

System.Console.WriteLine("\nDone");

}

}

## 完整代码如下

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Data.SqlTypes;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace csharp\_basic\_generic  {  public class GenericList<T>//泛型类  {  private class Node //泛型类的嵌套类也是泛型的  {  private Node next;  public Node Next  {  get { return next; }  set { next = value; }  }  private T data;  public T Data  {  get{ return data; }  set{ data = value; }  }  public Node(T t)  {  next = null;  data = t;  }  }  private Node head;  public GenericList()  {  head = null;  }  public void AddHead(T t) //头插法,就是新节点会插入到链表的头部  {  Node n = new Node(t);  n.Next = head;  head = n;  }  public IEnumerator<T> GetEnumerator()  {  Node current = head;  while(current!=null)  {  yield return current.Data;  current = current.Next;  }  }  }  internal class Program  {  static void TestGenericList()  {  GenericList<int> list = new GenericList<int>();  for (int i = 0; i < 10; i++)  {  list.AddHead(i);  }  foreach (var j in list)  {  Console.Write(j+" ");  }  Console.WriteLine("\nDone");  }  static void Main(string[] args)  {  TestGenericList();  }  }  } |

### 运行效果

|  |
| --- |
|  |

### 可以看到,这个链表使用的是头插法,新节点会插入到链表的头部

## 扩展:如何实现尾插法?,

## 更新后的完整代码如下

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Data.SqlTypes;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace csharp\_basic\_generic  {  public class GenericList<T>//泛型类  {  private class Node //泛型类的嵌套类也是泛型的  {  private Node next;  public Node Next  {  get { return next; }  set { next = value; }  }  private T data;  public T Data  {  get{ return data; }  set{ data = value; }  }  public Node(T t)  {  next = null;  data = t;  }  }  private Node head,body;  public GenericList()  {  head = null;  }  public void AddHead(T t) //头插法,就是新节点会插入到链表的头部  {  Node n = new Node(t);  n.Next = head;  head = n;  }  public void AddTail(T t) //尾插法,需要一个辅助变量body,头是不能移动的,需要移动body  {  Node n = new Node(t);  if (head == null) head = body = n;  body.Next = n;  body = n;  }  public IEnumerator<T> GetEnumerator()  {  Node current = head;  while(current!=null)  {  yield return current.Data;  current = current.Next;  }  }  }  internal class Program  {  static void TestGenericList()  {  GenericList<int> list = new GenericList<int>();  for (int i = 0; i < 10; i++)  {  list.AddHead(i);  }  foreach (var j in list)  {  Console.Write(j+" ");  }  Console.WriteLine("\nDone");  }  static void TestGenericListTail()  {  GenericList<int> list = new GenericList<int>();  for (int i = 0; i < 10; i++)  {  list.AddTail(i);  }  foreach (var j in list)  {  Console.Write(j + " ");  }  Console.WriteLine("\nDone");  }  static void Main(string[] args)  {  //TestGenericList();  TestGenericListTail();  }  }  } |

## 请参阅

[System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx)

[C# 编程指南](https://msdn.microsoft.com/zh-cn/library/67ef8sbd.aspx)

[泛型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/512aeb7t.aspx)