**[C#集合--Dictionary](http://www.cnblogs.com/yang_sy/p/3678905.html)**

字典(dictionary)是一个集合，其中每个元素都是一个键/值对。字典(Dictionaries)是常用于查找和排序的列表。

.NET Framework通过IDictionary接口和IDictionary<TKey,TValue>接口，以及一些常用的子典了定义了子典协议。每个类在以下方面各有不同：

* 元素是否已经排序
* 元素是否能通过索引或键来获取
* 字典类是generic的还是非generic的
* 当字段较大时，根据键值获取元素速度的快慢

下表总结了每个字典类，以及它们在上述这几个方面的差异。它们都是在一个1.5G的PC上执行5000次操作得到的一个平均值。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Type** | **内部结构** | **支持索引** | **内存占用** | **随机插入的速度(毫秒)** | **顺序插入的速度(毫秒)** | **根据键获取元素的速度(毫秒)** |
| 未排序字典 |  |  |  |  |  |  |
| Dictionary<T,V> | 哈希表 | 否 | 22 | 30 | 30 | 20 |
| Hashtable | 哈希表 | 否 | 38 | 50 | 50 | 30 |
| ListDictionary | 链表 | 否 | 36 | 50000 | 50000 | 50000 |
| OrderedDictionary | 哈希表 +数组 | 是 | 59 | 70 | 70 | 40 |
| 排序字典 |  |  |  |  |  |  |
| SortedDictionary<K,V> | 红黑树 | 否 | 20 | 130 | 100 | 120 |
| SortedList<K,V> | 2xArray | 是 | 20 | 3300 | 30 | 40 |
| SortList | 2xArray | 是 | 27 | 4500 | 100 | 180 |

从时间复杂度来讲，从字典中通过键获取值所耗费的时间分别如下：

* Hashtable, Dictionary和OrderedDictionary的时间复杂度为O(1)
* SortedDictionary和SortList的时间复杂度为O(logN)
* ListDictinary的时间复杂度为O(n)

n是集合元素的数量。

**IDictionary<TKey, TValue>**

IDictionary<TKey,Tvalue>指定了所有以key/value为基础集合的标准协议。由于它添加了方法和属性用以通过键读取元素，从而扩展了ICollection<T>接口：

[复制代码](javascript:void(0);)

public interface IDictionary <TKey, TValue> :

ICollection <KeyValuePair <TKey, TValue>>, IEnumerable

{

bool ContainsKey (TKey key);

bool TryGetValue (TKey key, out TValue value);

void Add (TKey key, TValue value);

bool Remove (TKey key);

TValue this [TKey key] { get; set; } // Main indexer - by key

ICollection <TKey> Keys { get; } // Returns just keys

ICollection <TValue> Values { get; } // Returns just values

}

[复制代码](javascript:void(0);)

向字典中添加一个元素，你可以调用add方法，或者通过索引器的set方法；对于后者，如果添加元素的键在字段中不存在，那么把该元素插入到字典中；否则更新字典中相同键对应的值。所有的字典实现类都不接受重复键，所以两次调用add方法时使用相同键则会抛出异常。

从字段中获取一个元素，可以使用索引器的get方法或者调用TryGetValue方法。如果键不存在，使用索引器方法会抛出异常，而TryGetValue返回false。你可通过ContainsKey方法来确认某一个键是否在字典中存在；但是这样会导致额外的查询开销。

可以通过KeyValuePari结构来遍历IDictionary<TKey,TValue>。

[复制代码](javascript:void(0);)

[Serializable]

public struct KeyValuePair<TKey, TValue> {

private TKey key;

private TValue value;

public KeyValuePair(TKey key, TValue value) {

this.key = key;

this.value = value;

}

public TKey Key {

get { return key; }

}

public TValue Value {

get { return value; }

}

public override string ToString() {

StringBuilder s = StringBuilderCache.Acquire();

s.Append('[');

if( Key != null) {

s.Append(Key.ToString());

}

s.Append(", ");

if( Value != null) {

s.Append(Value.ToString());

}

s.Append(']');

return StringBuilderCache.GetStringAndRelease(s);

}

}

[复制代码](javascript:void(0);)

当然，你也可以通过字典的Keys或Values属性遍历字典的所有键或值。在Dictionary类中，将演示该接口是如何使用的。

**IDictionary**

IDictionary是非generic的字典接口；与IDictionary<TKey, TValue>比较有两处不同：

1. 如果获取的对象不存在，返回null，不会抛出异常
2. 使用Contains方法以测试一个成员是否在字典中存在，而不是ContainsKey方法

[复制代码](javascript:void(0);)

public interface IDictionary : ICollection

{

// Interfaces are not serializable

// The Item property provides methods to read and edit entries

// in the Dictionary.

Object this[Object key] {

get;

set;

}

// Returns a collections of the keys in this dictionary.

ICollection Keys {

get;

}

// Returns a collections of the values in this dictionary.

ICollection Values {

get;

}

// Returns whether this dictionary contains a particular key.

//

bool Contains(Object key);

// Adds a key-value pair to the dictionary.

//

void Add(Object key, Object value);

// Removes all pairs from the dictionary.

void Clear();

bool IsReadOnly

{ get; }

bool IsFixedSize

{ get; }

// Returns an IDictionaryEnumerator for this dictionary.

new IDictionaryEnumerator GetEnumerator();

// Removes a particular key from the dictionary.

//

void Remove(Object key);

}

[复制代码](javascript:void(0);)

通过DictionaryEntry接口来遍历非generic的字典

[复制代码](javascript:void(0);)

[Serializable]

public struct DictionaryEntry

{

private Object \_key;

private Object \_value;

// Constructs a new DictionaryEnumerator by setting the Key

// and Value fields appropriately.

public DictionaryEntry(Object key, Object value) {

\_key = key;

\_value = value;

}

public Object Key {

get {

return \_key;

}

set {

\_key = value;

}

}

public Object Value {

get {

return \_value;

}

set {

\_value = value;

}

}

}

[复制代码](javascript:void(0);)

**Dictionary<TKey, TValue>和Hashtable**

geneirc的Dictionary类是使用最多的集合类（此外，就是List<T>集合类）。Dictionary<TKey, TValue>使用哈希数据结构来存储键和值，因此它既快速又高效。

非generic的Dictionary<TKey, TValue>就是Hashtable；因此不存在非generic的类Dictionary。当我们提及Dictionary时，我们一般是指Dictionary<TKey, TValue>。

Dictionary实现了generic和非generic的IDictionary接口，generic的IDictonary都暴露为public。实际上，Dictionary如果教科书一般地实现了generic的IDictionary接口。

下面的代码演示了如何使用Ditionary<TKey, TValue>类：

[复制代码](javascript:void(0);)

var d = new Dictionary<string, int>();

d.Add("One", 1);

d["Two"] = 2; // adds to dictionary because "two" is not already present

d["Two"] = 22; // updates dictionary because "two" is now present

d["Three"] = 3;

Console.WriteLine (d["Two"]); // Prints "22"

Console.WriteLine (d.ContainsKey ("One")); // true (fast operation)

Console.WriteLine (d.ContainsValue (3)); // true (slow operation)

int val = 0;

if (!d.TryGetValue ("onE", out val))

Console.WriteLine ("No val"); // "No val" (case sensitive)

// Three different ways to enumerate the dictionary:

foreach (KeyValuePair<string, int> kv in d) // One ; 1

Console.WriteLine (kv.Key + "; " + kv.Value); // Two ; 22

// Three ; 3

foreach (string s in d.Keys) Console.Write (s); // OneTwoThree

Console.WriteLine();

foreach (int i in d.Values) Console.Write (i); // 1223

[复制代码](javascript:void(0);)

该类背后的哈希表，把每个键都转换成一个整数型的哈希码，然后通过算法将其转换成一个哈希键。在内部通过哈希键确定一个成员属于哪一个“桶”；如果一个“桶”包含多个值，那么对该“桶”执行线型搜索。一个好的哈希算法，不仅努力实现返回一个严格的哈希码，而且还努力实现所返回的哈希码在32位的整数中均匀地分布。

字典可以包含任何类型的键，只要这些键支持是否相等接口并能获取哈希码。在默认情况下，键的相等性取决于对象的Equals方法，而计算哈希键的算法也基于对象的GetHashCode方法。这些行为不是一成不变的，如果重载了Equals方法或GetHashCode方法，或在创建字典实例时提供了IEqualityComparer实例对象。一个常见的应用就是在使用字符串字段时，提供了区分大小写的相等性比较器实例。

var d = new Dictionary<string, int> (StringComparer.OrdinalIgnoreCase);

与其它集合类型一样，如果在构造字典实例时，指定字段的大小，那么可以在一定程度上改善性能。指定字典的大小，可以避免或减少内部调正大小的操作。

Dictioanry和Hashtable的缺点是items并没有排序。甚至，添加到字典中的成员也不会保留原有的顺序。此外，字典还有一个缺点就是不接收重复的键。

**OrderedDictionary**

OrderedDictionary是非generic的字典类，它保存了成员原有的顺序。使用OrderedDictioanry时，你可以通过索引或键获取字段元素。

OrderedDictionary结合了Hashtable和ArrayList。这就意味着，它不仅有Hashtable的所有功能，还有RemoveAt，整数索引器方法。它还根据元素的原始顺序对外暴露Keys和Values属性。

该类在.NET 2.0中引入，而且没有对应的非generic版本。

**ListDictionary和HybirdDictionary**

ListDictionary使用单链表存储数据。它不提供排序，尽管它保留了元素的原始顺序。当集合很大时，其性能相当低。它值得注意的地方仅仅在于当元素数量很小时有效率（元素少于10个）。

HybirdDictionary是一个ListDictionary，它会当元素数量达到一定数量后自动转换成Hashtable，以解决ListDictionary的性能问题。这种想法可以使得字典元素很少时，占用较低的内存；而字典数量较大时拥有较好的性能。然而，在到了一定的数目后需要从一个数据类型转换成另一个数据类型--而Dictionary在这两种情况下都不会太慢或性能低--因此，你为何不在一开始就使用Dicontary类。

此外，这两个类都是非generic的类。

**可排序的Dictionary**

Framework提供了两个字典类，它们通过排序的键来构建。这两个类就是SortedDictoanry<TKey, TValue>和 SortedList<Tkey,TValue>。

SortedDictoanry<TKey, TValue>，使用红黑树：一种数据结构，该数据结构保证了任何插入和获取元素行为都是一致地。

SortedList<Tkey,TValue>，内部由一个排序后的数组对实现，可以实现快速读取，但是插入性能较差。

SortedDictoanry<TKey, TValue>比SortedList快，按照随机顺序插入元素。 SortedList，有一个额外的功能，可以通过索引或键获取元素。 使用排序后的列表，你可以直接找到第几个元素。 而如果想在SortedDictionary中实现同样的目的，那么你需要手动的遍历n个元素。

下面的例子演示了使用反射加载所有System.Object类的方法到一个排序后的列表，然后遍历该列表的键和值

[复制代码](javascript:void(0);)

var sorted = new SortedList <string, MethodInfo>();

foreach (MethodInfo m in typeof (object).GetMethods())

sorted [m.Name] = m;

foreach (string name in sorted.Keys)

Console.WriteLine (name);

foreach (MethodInfo m in sorted.Values)

Console.WriteLine (m.Name + " returns a " + m.ReturnType);

[复制代码](javascript:void(0);)

第一个列表的结果如下：

|  |
| --- |
| Equals GetHashCode GetType ReferenceEquals ToString |

第二个的列表的结果如下：

|  |
| --- |
| Equals returns a System.Boolean GetHashCode returns a System.Int32 GetType returns a System.Type ReferenceEquals returns a System.Boolean ToString returns a System.String |

请注意，我们通过索引器填充字段类。如果我们使用add方法，那么会抛出异常，这是因为我们所依赖的对象类重载了Equals方法，而你不能添加重复的键到一个字典中。而使用索引器，这会避免该问题。

如果扩展我们的示例，下面的代码则会返回GetHashCode方法，其使用方法和一个普通的字典的使用方式一样

Console.WriteLine (sorted ["GetHashCode"]);

到现在，我们的代码既适用于SortedDictionary也适用于SortedList。然而，下面两行代码仅仅适用于SortedList

Console.WriteLine (sorted.Keys [sorted.Count - 1]); // ToString

Console.WriteLine (sorted.Values[sorted.Count - 1].IsVirtual); // True

分类: [E-Book](http://www.cnblogs.com/yang_sy/category/23471.html),[C#](http://www.cnblogs.com/yang_sy/category/23468.html)

标签: [CSharp\_ALL\_Dictionary](http://www.cnblogs.com/yang_sy/tag/CSharp_ALL_Dictionary/)