# What is the best way to iterate over a dictionary?

Ask Question

2215

330

I've seen a few different ways to iterate over a dictionary in C#. Is there a standard way?

c# dictionary loops

edited May 10 '18 at 15:09

Uwe Keim
**27.7k** 32 135 216

asked Sep 26 '08 at 18:20

Jake Stewart
**11.2k** 3 16 13

80 I am surprised there are many answers to this question, along with one upvoted 923 times.. (uses foreach ofcourse).. I'd argue, or at the least add that if you have to iterate over a dictionary, chances are you are using it incorrectly / inappropriately.. I had to make this comment, because I have seen dictionaries being abused in ways which IMHO were not appropriate... Yes there can be rare circumstances when you iterate the dictionary, rather than lookup, which is what it is designed for.. Please bear that in mind, before wondering how to iterate over a dictionary. – Vikas Gupta Sep 18 '14 at 6:52

59 @VikasGupta What would you suggest for doing something with a collection of key-value pairs when you don't know what the keys are going to be? – nasch Feb 26 '15 at 22:27

19 @displayName If you want to do something with each key-value pair but don't have a reference to the keys to use to look up values, you

would iterate over the dictionary, right? I was just pointing out that there could be times you would want to do that, despite Vikas' claim that this is usually incorrect usage. – nasch Oct 28 '15 at 18:01

25 To say that it's incorrect usage implies that there's a better alternative. What's that alternative? – Kyle Delaney Jun 20 '17 at 21:13

17 VikasGupta is wrong, I can affirm that after many years of high-performance C# and C++ programming in non-theoretical scenarios. There are indeed frequent cases where one would create a dictionary, store unique key-value pairs, and then iterate over these values, which are proven to have unique keys within the collection. Creating any further collections is a really inefficient and costly way of avoiding dictionary iteration. Please provide a good alternative as answer to the question clarifying your point of view, otherwise your comment is pretty nonsensical. – ɹɐqɐlɔıunɐs dɐqɐ Aug 1 '18 at 20:53

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs



## 26 Answers

3216

```
foreach(KeyValuePair<string, string> entry in myDictionary)
{
    // do something with entry.Value or entry.Key
}
```

edited Dec 20 '17 at 23:14

jasonmerino
**2,477** 1 13 30

answered Sep 26 '08 at 18:22

Pablo Fernandez
**70.2k** 47 167 218

16 What if I don't exactly know the type of key/value in Dictionary. Using `var entry` is better in that case, and thus i did vote this answer on a second look instead of the above one. – Ozair Kafray Jan 5 '16 at 7:32

70 @OzairKafray using `var` when you don't know the type is generally bad practice. – Nate Jan 25 '16 at 23:34

41 This answer is superior because Pablo didn't default to the lazy coder "var" usage that obfuscates the return type. – MonkeyWrench May 17 '16 at 17:19

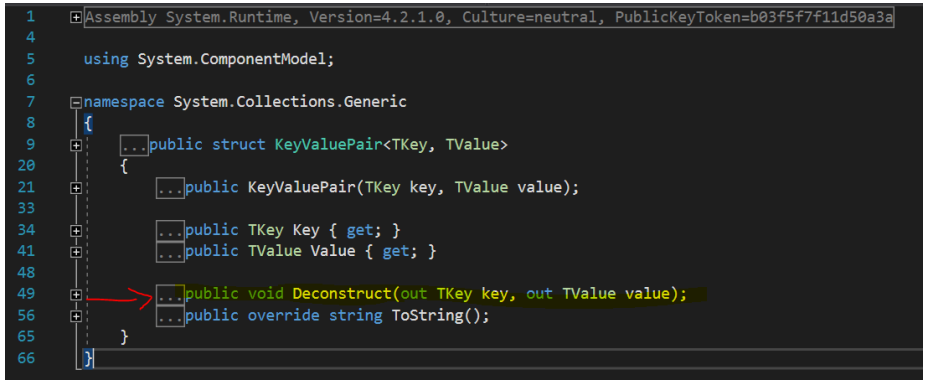
82 @MonkeyWrench: Meh. Visual Studio knows what the type is; all you have to do is hover over the variable to find out. – Robert Harvey ♦ Dec 9 '16 at 18:30

19 As I understand it, `var` only works if the type is known at compile time. If Visual Studio knows the type then it's available for you to find out as well. – Kyle Delaney Sep 21 '17 at 13:06

8

*C# 7.0* introduced **Deconstructors** and if you are using *.NET Core 2.0+* Application, the struct `KeyValuePair<>` already include a `Deconstruct()` for you. So you can do:

```
var dic = new Dictionary<int, string>() { { 1, "One" }, { 2, "Two" }
foreach (var (key, value) in dic) {
    Console.WriteLine($"Item [{key}] = {value}");
}
//Or
foreach (var (_, value) in dic) {
    Console.WriteLine($"Item [NO_ID] = {value}");
}
//Or
foreach ((int key, string value) in dic) {
    Console.WriteLine($"Item [{key}] = {value}");
}
```




answered Jan 7 at 20:30

Jaider
**9,259** 4 47 70

-1

in addition to the highest ranking posts where there is a discussion between using

```
foreach(KeyValuePair<string, string> entry in myDictionary)
{
    // do something with entry.Value or entry.Key
}
```

or

```
foreach(var entry in myDictionary)
{
    // do something with entry.Value or entry.Key
}
```

most complete is the following because you can see the dictionary type from the initialization, kvp is KeyValuePair

```
var myDictionary = new Dictionary<string, string>(x);//fill dictionar

foreach(var kvp in myDictionary)//iterate over dictionary
{
    // do something with kvp.Value or kvp.Key
}
```

answered Aug 19 '18 at 20:50

BigChief

**864** 3 15 31

1 Creating and copying a second dictionary is not a valid solution to code readability. In fact, I would argue it would make the code harder to understand because now you have to ask yourself: "Why did the last guy create a second dictionary?" If you want to be more verbose, just use option one. – Matthew Goulart Sep 6 '18 at 19:23

I just meant to show you that when decl dict before for each, usage in the foreach is clear from the declaration – BigChief Sep 8 '18 at 9:03

8

Using **C# 7**, add this **extension method** to any project of your solution:

```
public static class IDictionaryExtensions
{
    public static IEnumerable<(TKey, TValue)> Tuples<TKey, TValue>(
        this IDictionary<TKey, TValue> dict)
    {
        foreach (KeyValuePair<TKey, TValue> kvp in dict)
            yield return (kvp.Key, kvp.Value);
    }
}
```

And use this simple syntax

```
foreach (var(id, value) in dict.Tuples())
{
    // your code using 'id' and 'value'
}
```

Or this one, if you prefer

```
foreach ((string id, object value) in dict.Tuples())
{
    // your code using 'id' and 'value'
}
```

In place of the traditional

```
foreach (KeyValuePair<string, object> kvp in dict)
{
    string id = kvp.Key;
    object value = kvp.Value;

    // your code using 'id' and 'value'
}
```

The extension method transforms the `KeyValuePair` of your `IDictionary<TKey, TValue>` into a strongly typed `tuple`, allowing you to use this new comfortable syntax.

It converts -just- the required dictionary entries to `tuples`, so it does NOT converts the whole dictionary to `tuples`, so there are no performance concerns related to that.

There is a only minor cost calling the extension method for creating a `tuple` in comparison with using the `KeyValuePair` directly, which should NOT be an issue if you are assigning the `KeyValuePair`'s properties `Key` and `Value` to new loop variables anyway.

In practice, this new syntax suits very well for most cases, except for low-level ultra-high performance scenarios, where you still have the option to simply not use it on that specific spot.

Check this out: MSDN Blog - New features in C# 7

edited Aug 1 '18 at 20:38

answered May 27 '18 at 12:25

sɐuɹɔlɐqɐp
**1,697** 6 20 25

1 What would be the reason to prefer 'comfortable' tuples to key-value-pair's? I don't see a gain here. Your tuple contains a key and a value, so does the key-value-pair. – Maarten Sep 10 '18 at 12:57

2 Hi Maarten, thanks for your question. The main benefit is code readability without additional programming effort. With KeyValuePair one must always use the form `kvp.Key` and `kvp.Value` for using respectively key and value. With tuples you get the flexibility to name the key and the value as you wish, without using further variable declarations inside the foreach block. E.g. you may name your key as `factoryName`, and the value as `models`, which is especially useful when you get nested loops (dictionaries of dictionaries): code maintenance gets much easier. Just give it a try! ;-) – sɐuɹɔlɐqɐp Sep 10 '18 at 13:49

2

As of C# 7, you can deconstruct objects into variables. I believe this to be the best way to iterate over a dictionary.

**Example:**

Create an extension method on `KeyValuePair<TKey, TVal>` that deconstructs it:

```
public static void Deconstruct<TKey, TVal>(this KeyValuePair<TKey, T
out TVal val)
{
   key = pair.Key;
   val = pair.Value;
}
```

Iterate over any `Dictionary<TKey, TVal>` in the following manner

```
// Dictionary can be of any types, just using 'int' and 'string' as
Dictionary<int, string> dict = new Dictionary<int, string>();

// Deconstructor gets called here.
foreach (var (key, value) in dict)
{
   Console.WriteLine($"{key} : {value}");
}
```

answered Jul 11 '18 at 18:05

Domn Werner
**169** 4 13

---

118

In some cases you may need a counter that may be provided by for-loop implementation. For that, LINQ provides `ElementAt` which enables the following:

```
for (int index = 0; index < dictionary.Count; index++) {
  var item = dictionary.ElementAt(index);
  var itemKey = item.Key;
  var itemValue = item.Value;
}
```

edited Jul 4 '18 at 12:25

Kolappan Nathan
**606** 1 15 20

answered Mar 10 '11 at 20:44

Maurício Fedatto
**1,725** 1 10 2

17 To use the '.ElementAt' method, remember: using System.Linq; This is not incluted in fx. auto generated test classes. – Tinia Nov 24 '11 at 14:47

12 This is the way to go if you are modifying the values associated with the keys. Otherwise an exception is thrown when modifying and using foreach(). – Mike de Klerk Apr 3 '13 at 7:18

8 Be careful when using this. See here: stackoverflow.com/a/2254480/253938 – RenniePet Jan 14 '14 at 18:54

22 Isn't `ElementAt` a O(n) operation? – Arturo Torres Sánchez Feb 7 '15 at 1:46

138 This answer is completely undeserving of so many upvotes. A dictionary has no implicit order, so using `.ElementAt` in this context might lead to subtle bugs. Far more serious is Arturo's point above. You'll be iterating the dictionary `dictionary.Count + 1` times leading to O(n^2) complexity for an operation that should only be O(n). If you really need an index (if you do, you're probably using the wrong collection type in the first place), you should iterate `dictionary.Select( (kvp, idx) => new {Index = idx, kvp.Key, kvp.Value})` instead and not use `.ElementAt` inside the loop. – spender Mar 2 '15 at 2:17

1

I wrote an extension to loop over a dictionary.

```
public static class DictionaryExtension
{
    public static void ForEach<T1, T2>(this Dictionary<T1, T2> dicti
T2> action) {
        foreach(KeyValuePair<T1, T2> keyValue in dictionary) {
            action(keyValue.Key, keyValue.Value);
        }
    }
}
```

Then you can call

```
myDictionary.ForEach((x,y) => Console.WriteLine(x + " - " + y));
```

answered Jun 8 '18 at 7:07

Steven Delrue
**139** 2 4

You defined a `ForEach` method in which you have `foreach (...) { }` ... Seems like unnecessary. – Maarten Mar 26 at 13:51

44

I would say foreach is the standard way, though it obviously depends on what you're looking for

```
foreach(var kvp in my_dictionary) {
  ...
}
```

Is that what you're looking for?

edited May 17 '18 at 15:59

Community ♦
**1** 1

answered Sep 26 '08 at 18:22

George Mauer
**50.2k** 104 310 533

38 Um, isn't naming the item "value" rather confusing? You would typically be using syntax like "value.Key" and "value.Value", which isn't very intuitive for anyone else who will be reading the code, especially if they aren't

familiar with how the .Net Dictionary is implemented. – RenniePet Jan 11 '14 at 12:09

1 @RenniePet `kvp` is commonly used to name KeyValuePair instances when iterating over dictionaries and related data structures: `foreach(var kvp in myDictionary){...` . – mbx Mar 9 '16 at 16:10

23

I appreciate this question has already had a lot of responses but I wanted to throw in a little research.

Iterating over a dictionary can be rather slow when compared with iterating over something like an array. In my tests an iteration over an array took 0.015003 seconds whereas an iteration over a dictionary (with the same number of elements) took 0.0365073 seconds that's 2.4 times as long! Although I have seen much bigger differences. For comparison a List was somewhere in between at 0.00215043 seconds.

However, that is like comparing apples and oranges. My point is that iterating over dictionaries is slow.

Dictionaries are optimised for lookups, so with that in mind I've created two methods. One simply does a foreach, the other iterates the keys then looks up.

```
public static string Normal(Dictionary<string, string> dictionary)
{
    string value;
    int count = 0;
    foreach (var kvp in dictionary)
    {
        value = kvp.Value;
        count++;
    }

    return "Normal";
}
```

This one loads the keys and iterates over them instead (I did also try pulling the keys into a string[] but the difference was negligible.

```
public static string Keys(Dictionary<string, string> dictionary)
{
    string value;
    int count = 0;
    foreach (var key in dictionary.Keys)
    {
        value = dictionary[key];
        count++;
    }

    return "Keys";
}
```

With this example the normal foreach test took 0.0310062 and the keys version took 0.2205441. Loading all the keys and iterating over all the lookups is clearly a LOT slower!

For a final test I've performed my iteration ten times to see if there are any benefits to using the keys here (by this point I was just curious):

Here's the RunTest method if that helps you visualise what's going on.

```
private static string RunTest<T>(T dictionary, Func<T, string> funct
{
    DateTime start = DateTime.Now;
    string name = null;
    for (int i = 0; i < 10; i++)
    {
        name = function(dictionary);
    }
    DateTime end = DateTime.Now;
    var duration = end.Subtract(start);
    return string.Format("{0} took {1} seconds", name, duration.Tota
}
```

Here the normal foreach run took 0.2820564 seconds (around ten times longer than a single iteration took - as you'd expect). The iteration over the keys took 2.2249449 seconds.

**Edited To Add:** Reading some of the other answers made me question what would happen if I used Dictionary instead of Dictionary. In this example the array took 0.0120024 seconds, the list 0.0185037 seconds and the dictionary 0.0465093 seconds. It's reasonable to expect that the data type makes a difference on how much slower the dictionary is.

**What are my Conclusions**?

- Avoid iterating over a dictionary if you can, they are substantially slower than iterating over an array with the same data in it.
- If you do choose to iterate over a dictionary don't try to be too clever, although slower you could do a lot worse than using the standard foreach method.

edited Apr 28 '18 at 12:22

Mohammadreza Farahani

**2,502** 4 16 26

answered Jul 30 '14 at 10:54

Liath

**7,050** 9 38 74

9 You should measure with something like StopWatch instead of DateTime: hanselman.com/blog/… – Even Mien Feb 23 '15 at 19:26

2 could you please describe your test scenario, how many items where in your dictionary, how often did you run your scenario to calculate the average time, ... – WiiMaxx Jul 29 '15 at 8:57

3 Interestingly you will get different results depending upon what data you have in the dictionary. While itterating over the Dictionary the Enumerator function has to skip a lot of empty slots in the dictionary which is what causes it to be slower than iterating over an array. If the Dictionary is full

up there will be less empty slots to skip than if it is half empty. –
Martin Brown Jun 30 '16 at 8:36

0

**Dictionary< TKey, TValue >** It is a generic collection class in c# and it stores the data in the key value format.Key must be unique and it can not be null whereas value can be duplicate and null.As each item in the dictionary is treated as KeyValuePair< TKey, TValue > structure representing a key and its value. and hence we should take the element type KeyValuePair< TKey, TValue> during the iteration of element.**Below is the example.**

```
Dictionary<int, string> dict = new Dictionary<int, string>();
dict.Add(1,"One");
dict.Add(2,"Two");
dict.Add(3,"Three");

foreach (KeyValuePair<int, string> item in dict)
{
    Console.WriteLine("Key: {0}, Value: {1}", item.Key, item.Value);
}
```

answered Apr 16 '18 at 11:25

Sheo Dayal Singh
**673** 6 7

22

There are plenty of options. My personal favorite is by KeyValuePair

```
Dictionary<string, object> myDictionary = new Dictionary<string, obj
// Populate your dictionary here

foreach (KeyValuePair<string,object> kvp in myDictionary)
{
```

```
    // Do some interesting things
}
```

You can also use the Keys and Values Collections

edited Apr 12 '18 at 10:17

particulator
**58** 1 7

answered Sep 26 '08 at 18:22

theo
**3,415** 3 17 22

---

80

Depends on whether you're after the keys or the values...

From the MSDN `Dictionary(TKey, TValue)` Class description:

```
// When you use foreach to enumerate dictionary elements,
// the elements are retrieved as KeyValuePair objects.
Console.WriteLine();
foreach( KeyValuePair<string, string> kvp in openWith )
{
    Console.WriteLine("Key = {0}, Value = {1}",
        kvp.Key, kvp.Value);
}

// To get the values alone, use the Values property.
Dictionary<string, string>.ValueCollection valueColl =
    openWith.Values;

// The elements of the ValueCollection are strongly typed
// with the type that was specified for dictionary values.
Console.WriteLine();
foreach( string s in valueColl )
{
    Console.WriteLine("Value = {0}", s);
}

// To get the keys alone, use the Keys property.
```

```
Dictionary<string, string>.KeyCollection keyColl =
    openWith.Keys;

// The elements of the KeyCollection are strongly typed
// with the type that was specified for dictionary keys.
Console.WriteLine();
foreach( string s in keyColl )
{
    Console.WriteLine("Key = {0}", s);
}
```

edited Apr 9 '18 at 10:16

NearHuscarl
65 1 2 8

answered Sep 26 '08 at 18:27

J Healy
1,834 16 13

---

-3

Dictionaries are special lists, whereas every value in the list has a key which is also a variable. A good example of a dictionary is a phone book.

```
Dictionary<string, long> phonebook = new Dictionary<string, long>
 phonebook.Add("Alex", 4154346543);
 phonebook["Jessica"] = 4159484588;
```

Notice that when defining a dictionary, we need to provide a generic definition with two types - the type of the key and the type of the value. In this case, the key is a string whereas the value is an integer.

There are also two ways of adding a single value to the dictionary, either using the brackets operator or using the Add method.

To check whether a dictionary has a certain key in it, we can use the ContainsKey method:

```
Dictionary<string, long> phonebook = new Dictionary<string, long>();
phonebook.Add("Alex", 415434543);
phonebook["Jessica"] = 415984588;

if (phonebook.ContainsKey("Alex"))
{
    Console.WriteLine("Alex's number is " + phonebook["Alex"]);
}
```

To remove an item from a dictionary, we can use the Remove method. Removing an item from a dictionary by its key is fast and very efficient. When removing an item from a List using its value, the process is slow and inefficient, unlike the dictionary Remove function.

```
Dictionary<string, long> phonebook = new Dictionary<string, long>();
phonebook.Add("Alex", 415434543);
phonebook["Jessica"] = 415984588;

phonebook.Remove("Jessica");
Console.WriteLine(phonebook.Count);
```

edited Mar 15 '18 at 14:19

Sebastian Hofmann

**1,335** 5 9 19

answered Mar 9 '18 at 6:02

Amit Kumar Verma

**318** 2 13

With `.NET Framework 4.7` one can use *decomposition*

10

```
var fruits = new Dictionary<string, int>();
...
foreach (var (fruit, number) in fruits)
{
    Console.WriteLine(fruit + ": " + number);
}
```

To make this code work on lower C# versions, add `System.ValueTuple NuGet package` and write somewhere

```
public static class MyExtensions
{
    public static void Deconstruct<T1, T2>(this KeyValuePair<T1, T2>
        out T1 key, out T2 value)
    {
        key = tuple.Key;
        value = tuple.Value;
    }
}
```

answered Oct 17 '17 at 15:18

Pavel
**969** 2 9 25

6 This is incorrect. .NET 4.7 simply has `ValueTuple` built in. It's available as a nuget package for earlier versions. More importantly, C# 7.0+ is needed for the `Deconstruct` method to work as a deconstructor for `var (fruit, number) in fruits` . – David Arno Oct 18 '17 at 11:58

Simplest form to iterate a dictionary:

8

```
foreach(var item in myDictionary)
{
    Console.WriteLine(item.Key);
    Console.WriteLine(item.Value);
}
```

edited Mar 13 '17 at 15:35

Brent Rittenhouse
**413** 4 14

answered Oct 2 '16 at 4:00

Ron
**919** 4 10 27

67

Generally, asking for "the best way" without a specific context is like asking what is the best color.

One the one hand, there are many colors and there's no best color. It depends on the need and often on taste, too.

On the other hand, there are many ways to iterate over a Dictionary in C# and there's no best way. It depends on the need and often on taste, too.

## Most straightforward way

```
foreach (var kvp in items)
{
    // key is kvp.Key
    doStuff(kvp.Value)
}
```

If you need only the value (allows to call it `item`, more readable than `kvp.Value`).

```
foreach (var item in items.Values)
{
    doStuff(item)
}
```

## If you need a specific sort order

Generally, beginners are surprised about order of enumeration of a Dictionary.

LINQ provides a concise syntax that allows to specify order (and many other things), e.g.:

```
foreach (var kvp in items.OrderBy(kvp => kvp.Key))
{
    // key is kvp.Key
    doStuff(kvp.Value)
}
```

Again you might only need the value. LINQ also provides a concise solution to:

- iterate directly on the value (allows to call it `item`, more readable than `kvp.Value`)
- but sorted by the keys

Here it is:

```
foreach (var item in items.OrderBy(kvp => kvp.Key).Select(kvp => kvp
{
    doStuff(item)
}
```

There are many more real-world use case you can do from these examples. If you don't need a specific order, just stick to the "most straightforward way" (see above)!

edited Sep 23 '16 at 15:38

answered Aug 10 '15 at 11:15



Stéphane Gourichon

**3,164** 20 41

The last one should be `.Values` and not a select clause. – Mafii Sep 23 '16 at 10:05

@Mafii Are you sure? Values returned by OrderBy are not of a KeyValuePair type, they have no `Value` field. Exact type I see here is `IOrderedEnumerable<KeyValuePair<TKey, TValue>>` . Perhaps you meant something else? Can you write a complete line showing what you mean (and test it)? – Stéphane Gourichon Sep 23 '16 at 14:30

I think this answer contains what I mean: stackoverflow.com/a/141105/5962841 but correct me if I confused something – Mafii Sep 23 '16 at 14:34

1 @Mafii Re-read my whole answer, explanations between code sections tell the context. The answer you mention is like second code section in my answer (no order required). There I just wrote `items.Value` like you suggested. In the case of the fourth section that you commented, the `Select()` is a way to cause `foreach` to enumerate directly on the values in the dictionary instead of key-value pairs. If somehow you don't like the `Select()` in this case, you might prefer the third code section. The point of the fourth section is to show that one can pre-process the collection with LINQ. – Stéphane Gourichon Sep 23 '16 at 15:14

1 If you do `.Keys.Orderby()` you'll iterate on a list of keys. If that's all you need, fine. If you need values, then in the loop you'd have to query the dictionary on each key to get the value. In many scenarios it won't make a practical difference. In high-performance scenario, it will. Like I wrote in the beginning of the answer: "there are many ways (...) and there's no best way. It depends on the need and often on taste, too." – Stéphane Gourichon Sep 23 '16 at 15:23

1

Just wanted to add my 2 cent, as the most answers relate to foreach-loop. Please, take a look at the following code:

```
Dictionary<String, Double> myProductPrices = new Dictionary<String, D

//Add some entries to the dictionary
```

```
myProductPrices.ToList().ForEach(kvP =>
{
    kvP.Value *= 1.15;
    Console.Writeline(String.Format("Product '{0}' has a new price:
kvP.Value));
});
```

Altought this adds a additional call of '.ToList()', there might be a slight performance-improvement (as pointed out here foreach vs someList.Foreach(){}), espacially when working with large Dictionaries and running in parallel is no option / won't have an effect at all.

Also, please note that you wont be able to assign values to the 'Value' property inside a foreach-loop. On the other hand, you will be able to manipulate the 'Key' as well, possibly getting you into trouble at runtime.

When you just want to "read" Keys and Values, you might also use IEnumerable.Select().

```
var newProductPrices = myProductPrices.Select(kvp => new { Name = kv
kvp.Value * 1.15 } );
```

edited May 23 '17 at 12:34

Community ♦
1 1

answered Sep 16 '16 at 16:03

Alex
99 6

5 Copying the entire collection for no reason at all will *not* improve performance. It'll dramatically slow down the code, as well as doubling the memory footprint of code that ought to consume virtually no additional memory. – Servy Sep 16 '16 at 16:08

I avoid the side-effect 'List.ForEach' method: `foreach` forces side-effect

visibility up, where it belongs. – user2864740 Mar 28 '17 at 22:30

3

The standard way to iterate over a Dictionary, according to official documentation on MSDN is:

```
foreach (DictionaryEntry entry in myDictionary)
{
     //Read entry.Key and entry.Value here
}
```

answered Jul 28 '16 at 10:58

Nick
**69** 2 10

This doesn't work for Dictionaries implementing generics... – ɹǝqɐlɔıunɐs Jul 23 '18 at 9:26

33

You can also try this on big dictionaries for multithreaded processing.

```
dictionary
.AsParallel()
.ForAll(pair =>
{
    // Process pair.Key and pair.Value here
});
```

edited Jun 11 '15 at 14:50

answered Jun 11 '15 at 13:32

Onur
**564** 6 15

6 @WiiMaxx and more important if these items do NOT depend on each other – Mafii Sep 23 '16 at 10:07

1

```
var dictionary = new Dictionary<string, int>
{
    { "Key", 12 }
};

var aggregateObjectCollection = dictionary.Select(
    entry => new AggregateObject(entry.Key, entry.Value));
```

answered May 28 '15 at 15:00

Pixar
**388** 1 4 16

2 There needs to be more justification/description in this answer. What does `AggregateObject` add to `KeyValuePair` ? Where is the "iteration," as requested in the question? – Marc L. Dec 29 '17 at 16:37

Select iterates over dictionary and allows us to work on each object. It is not as general as `foreach` , but I've used it a lot. Did my answer truly deserve downvote? – Pixar Jun 2 '18 at 18:37

No, `Select` *uses* iteration to effect the result, but is not an iterator itself. The types of things that iteration ( `foreach` ) is used for--especially operations with side-effects--are outside the scope of Linq, including `Select` . The lambda will not run until `aggregateObjectCollection` is actually enumerated. If this answer is taken as a "first path" (i.e., used before a straight `foreach` ) it encourages bad practices. Situationally, there may Linq operations that are helpful *before* iterating a dictionary, but that doesn't address the question as asked. – Marc L. Jun 4 '18 at 16:11

733

If you are trying to use a generic Dictionary in C# like you would use an associative array in another language:

```
foreach(var item in myDictionary)
{
  foo(item.Key);
  bar(item.Value);
}
```

Or, if you only need to iterate over the collection of keys, use

```
foreach(var item in myDictionary.Keys)
{
  foo(item);
}
```

And lastly, if you're only interested in the values:

```
foreach(var item in myDictionary.Values)
{
  foo(item);
}
```

(Take note that the `var` keyword is an optional C# 3.0 and above feature, you could also use the exact type of your keys/values here)

edited Feb 2 '15 at 19:19

answered Sep 26 '08 at 18:22

Jacob

**18k** 6 34 55

9 the var feature is most required for your first code block :) – nawfal Nov 1

'13 at 14:26

18 I appreciate that this answer points out you can iterate over the keys or the values explicitly. – Rotsiser Mho Jan 13 '15 at 6:17

9 I don't like the use of var here. Given that it is just syntactic sugar, why use it here? When someone is trying to read the code, they will have to jump around the code to determine the type of `myDictionary` (unless that is the actual name of course). I think using var is good when the type is obvious e.g. `var x = "some string"` but when it is not immediately obvious I think it's lazy coding that hurts the code reader/reviewer – James Wierzba Jan 25 '17 at 17:23

6 `var` should be used sparingly, in my opinion. Particularly here, it is not constructive: the type `KeyValuePair` is likely relevant to the question. – Sinjai Aug 21 '17 at 16:21

2 `var` has a unique purpose and i don't believe it is 'syntactic' sugar. Using it purposefully is an appropriate approach. – Joshua K Sep 7 '18 at 18:43

3

I will take the advantage of .NET 4.0+ and provide an updated answer to the originally accepted one:

```
foreach(var entry in MyDic)
{
    // do something with entry.Value or entry.Key
}
```

answered Oct 1 '14 at 23:17

yazanpro

**1,932** 3 30 55

Sometimes if you only needs the values to be enumerated, use the dictionary's value collection:

6

```
foreach(var value in dictionary.Values)
{
    // do something with entry.Value only
}
```

Reported by this post which states it is the fastest method: http://alexpinsker.blogspot.hk/2010/02/c-fastest-way-to-iterate-over.html

answered Jul 2 '14 at 1:55

ender
338 4 20

+1 for bringing performance in. Indeed, iterating over the dictionary itself does include some overhead if all you need are values. That is mostly due to copying values or references into KeyValuePair structs. – Jaanus Varus Aug 5 '15 at 10:25

1 FYI the last test in that link is test the wrong thing: it measures iterating over the keys, disregarding values, whereas the previous two tests do use the value. – Crescent Fresh Jul 13 '18 at 14:15

10

You suggested below to iterate

```
Dictionary<string,object> myDictionary = new Dictionary<string,objec
//Populate your dictionary here

foreach (KeyValuePair<string,object> kvp in myDictionary) {
    //Do some interesting things;
}
```

FYI, `foreach` doesn't work if the value are of type object.

edited May 12 '10 at 16:01



George Stocker ♦

**46.1k** 28 155 220

answered Oct 28 '09 at 20:49

Khushi

3 Please elaborate: `foreach` will not work if *which* value is of type `object` ? Otherwise this doesn't make much sense. – Marc L. Dec 29 '17 at 16:22

5

I found this method in the documentation for the DictionaryBase class on MSDN:

```
foreach (DictionaryEntry de in myDictionary)
{
    //Do some stuff with de.Value or de.Key
}
```

This was the only one I was able to get functioning correctly in a class that inherited from the DictionaryBase.

answered Feb 17 '09 at 23:51

Zannjaminderson

3 This looks like when using the non-generics version of Dictionary... i.e. prior to .NET framework 2.0. – joedotnot Apr 26 '14 at 13:21

@joed0tnot : it is the non-generics version used for `Hashtable` objects – ɐɹnuıɔlɐqɐp Aug 1 '18 at 19:42

2

If say, you want to iterate over the values collection by default, I believe you can implement IEnumerable<>, Where T is the type of the values object in the dictionary, and "this" is a Dictionary.

```
public new IEnumerator<T> GetEnumerator()
{
   return this.Values.GetEnumerator();
}
```

answered Dec 9 '08 at 4:16

mzirino

This doesn't explain much about how to iterate... – sɐunıɔlɐqɐp Jul 23 '18 at 9:27

**protected** by Moinuddin Quadri Jan 22 '17 at 14:32

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?