

Bài 10: DICTIONARY TRONG C#

Xem bài học trên website để ủng hộ Kteam: [Dictionary trong C#](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về [LIST TRONG C#](#). Hôm nay chúng ta sẽ cùng tìm hiểu về **Dictionary trong C#**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [BIẾN](#) , [KIỂU DỮ LIỆU](#), [TOÁN TỬ](#) trong C#
- [CÂU ĐIỀU KIỆN](#) trong C#
- Cấu trúc cơ bản của [VÒNG LẶP](#), [HÀM](#) trong C#
- [MẢNG](#) trong C#
- [LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG C#](#)

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Dictionary là gì?
- Một số thuộc tính và phương thức hỗ trợ sẵn trong Dictionary.

Dictionary là gì?

Tương tự như List, Dictionary chính là sự thay thế cho **Collections Hashtable** đã được học. Cho nên về khái niệm hay sử dụng thì Dictionary đều sẽ giống [Hashtable](#).

Dictionary trong C# là một Collections lưu trữ dữ liệu dưới dạng cặp **Key - Value**. Key đại diện cho 1 khoá giống như chỉ số phần tử của mảng và Value chính là giá trị tương ứng của khoá đó. Ta sẽ sử dụng Key để truy cập đến Value tương ứng.

Do Dictionary là 1 **Generic Collections** nên để sử dụng ta cần thêm thư viện **System.Collections.Generic** bằng câu lệnh:

```
using System.Collections.Generic;
```

Vì Dictionary là một lớp nên trước khi sử dụng ta cần khởi tạo vùng nhớ bằng toán tử **new**:

```
// khởi tạo 1 Dictionary rỗng với Key và Value đều có kiểu dữ liệu là chuỗi.  
Dictionary<string, string> MyHash = new Dictionary<string, string>();
```

Bạn cũng có chỉ định sức chứa (Capacity) ngay lúc khởi tạo bằng cách thông qua **constructor** được hỗ trợ sẵn:

```
/*  
 * khởi tạo 1 Dictionary với Key và Value có kiểu chuỗi  
 * đồng thời chỉ định Capacity ban đầu là 5  
 */  
Dictionary<string, string> MyDic2 = new Dictionary<string, string>(5);
```

Ngoài ra bạn cũng có thể khởi tạo 1 **Dictionary** chứa các phần tử được sao chép từ một Dictionary khác:

```
/*  
 * Khởi tạo 1 Dictionary có kích thước bằng với MyDic2.  
 * Sao chép toàn bộ phần tử trong MyDic2 vào MyDic3.  
 */  
Dictionary<string, string> MyDic3 = new Dictionary<string, string>(MyDic2);
```

Một số thuộc tính và phương thức hỗ trợ sẵn trong Dictionary

Một số thuộc tính thông dụng trong Dictionary:

TÊN THUỘC TÍNH	Ý NGHĨA
Count	Trả về 1 số nguyên là số phần tử hiện có trong Dictionary.
Keys	Trả về 1 danh sách chứa các Key trong Dictionary.
Values	Trả về 1 danh sách chứa các Value trong Dictionary.

Một số phương thức thông dụng trong Dictionary:

TÊN PHƯƠNG THỨC	Ý NGHĨA
Add(TKey Key, TValue Value)	Thêm 1 cặp Key - Value vào Dictionary.
Clear()	Xoá tất cả các phần tử trong Dictionary.
ContainsKey(TKey Key)	Kiểm tra đối tượng Key có tồn tại trong Dictionary hay không.
ContainsValue(TValue Value)	Kiểm tra đối tượng Value có tồn tại trong Dictionary hay không.
Remove(TKey Key)	Xoá đối tượng có Key xuất hiện đầu tiên trong Dictionary.
TryGetValue(TKey Key, TValue Value)	Kiểm tra Key có tồn tại hay không. Nếu có sẽ trả về true đồng thời trả về giá trị Value tương ứng qua biến Value. Ngược lại trả về false .

Một số lưu ý về Dictionary:

Mỗi một phần tử trong Dictionary (bao gồm 1 cặp **Key** - **Value**) được C# định nghĩa là 1 đối tượng có kiểu

KeyValuePair<TKey, TValue>

Trong đó, có 2 thuộc tính chính:

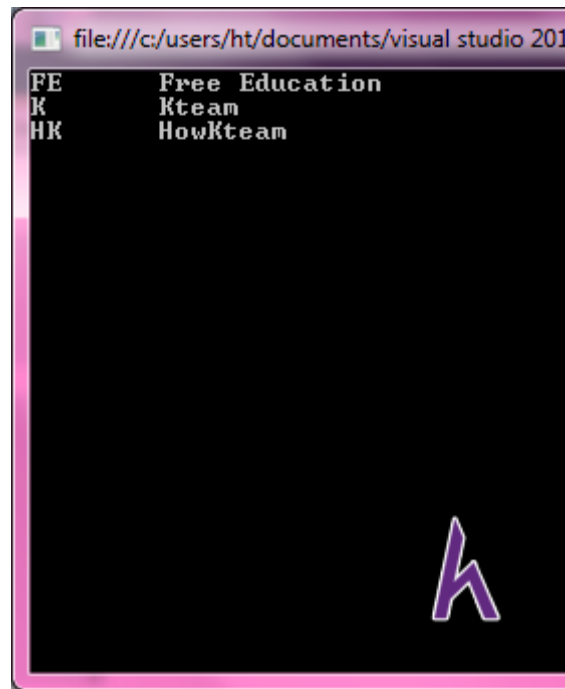
- **Key**: trả về giá trị Key của phần tử hiện tại.
- **Value**: trả về giá trị Value của phần tử hiện tại.

Điều này tương tự như **DictionaryEntry** trong **Hashtable**. Vì thế cách sử dụng cũng tương tự. Ví dụ mình thử dùng foreach duyệt 1 Dictionary và in ra giá trị **Key – Value** của mỗi phần tử:

```
// Tạo 1 Dictionary đơn giản và thêm vào 3 phần tử.
Dictionary<string, string> MyDic4 = new Dictionary<string, string>();
MyDic4.Add("FE", "Free Education");
MyDic4.Add("K", "Kteam");
MyDic4.Add("HK", "HowKteam");

/*
 * Duyệt qua các phần tử trong Dictionary.
 * Vì mỗi phần tử là 1 KeyValuePair nên ta chỉ định kiểu dữ liệu cho item là
 * KeyValuePair luôn.
 * Thử in ra màn hình cặp Key - Value của mỗi phần tử được duyệt.
 */
foreach (KeyValuePair<string, string> item in MyDic4)
{
    Console.WriteLine(item.Key + "\t" + item.Value);
}
```

Kết quả:



Việc truy xuất các phần tử trong Dictionary giống như truy xuất các phần tử mảng nhưng thông qua Key.

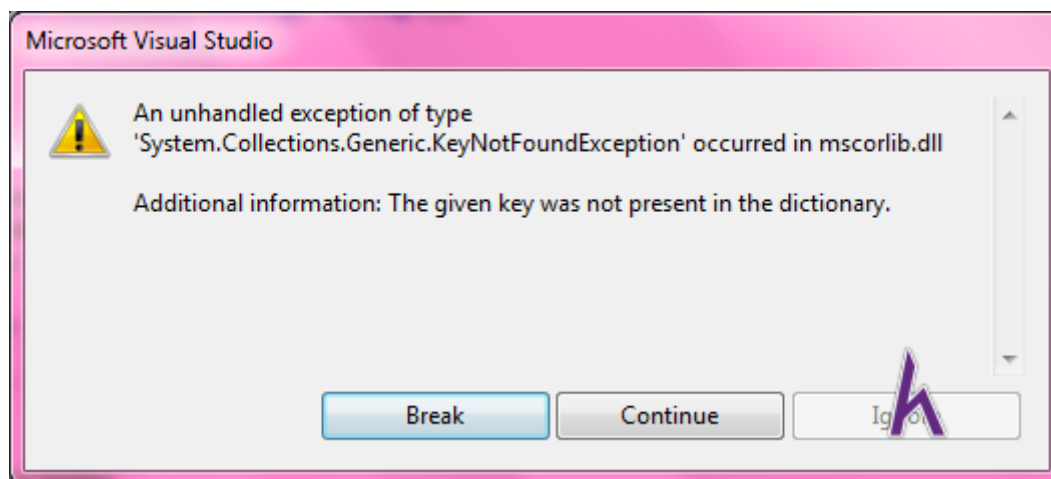
Ví dụ:

```
Console.WriteLine(MyDic4["FE"]);
```



Trong đó:

- **MyDic4** là tên của Dictionary.
- **"FE"** là tên Key cần truy xuất. Lưu ý là phải cùng kiểu dữ liệu TKey đã được chỉ định lúc khởi tạo Dictionary.
- **MyHash["FE"]** sẽ lấy ra giá trị Value tương ứng với Key trên.

Khi thao tác với **Hashtable** nếu như truy xuất đến phần tử có Key không tồn tại sẽ không báo lỗi (chi tiết mình đã trình bày trong bài [HASHTABLE TRONG C#](#)) nhưng với Dictionary thì không phải vậy. Nếu ta truy xuất đến Key không tồn tại sẽ nhận được lỗi sau:



Khác biệt giữa Dictionary và HashTable trong C#

HASHTABLE	DICTIONARY 
Threadsafe – Hỗ trợ multi threading không đụng độ tài nguyên	Không hỗ trợ
Cặp key – value lưu kiểu object	Phải xác định cụ thể kiểu dữ liệu của cặp key – value
Truy xuất phần tử không tồn tại trong HashTable sẽ không báo lỗi => return null	Truy xuất phần tử không tồn tại trong Dictionary sẽ báo lỗi
Hiệu quả cho dữ liệu lớn	Không hiệu quả cho dữ liệu lớn
Các phần tử được sắp xếp lại mỗi khi thêm hoặc xóa phần tử trong HashTable	Các phần tử nằm theo thứ tự được thêm vào
 Tìm kiếm nhanh hơn	Tìm kiếm chậm hơn

Kết luận

Nội dung bài này giúp các bạn nắm được:

- Dictionary là gì?
- Sự khác nhau giữa Dictionary và HashTable.

Một số thuộc tính và phương thức hỗ trợ sẵn trong Dictionary.

Bài học sau chúng ta sẽ cùng tìm hiểu về [TUPLE TRONG C#](#).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.

