

Bài: Lớp String trong Lập trình C# căn bản

Xem bài học trên website để ủng hộ Kteam: [Lớp String trong Lập trình C# căn bản](#)

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 [CẤU TRÚC LẶP FOREACH TRONG C#](#). Hôm nay chúng ta sẽ cùng tìm hiểu về **String trong C#**.

Nội dung

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

- [BIẾN](#) và [KIỂU DỮ LIỆU](#) trong C#
- [TOÁN TỬ TRONG C#](#)
- [CÂU ĐIỀU KIỆN TRONG C#](#)
- [CẤU TRÚC CƠ BẢN CỦA VÒNG LẶP TRONG C#](#)
- [CẤU TRÚC CƠ BẢN CỦA HÀM TRONG C#](#)
- [MẢNG 1 CHIỀU TRONG C#](#)

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

- Lớp String trong C#
- Ứng dụng lớp String vào việc xử lý chuỗi.
- Lớp StringBuilder trong C#

Lớp String trong C#

String là một kiểu dữ liệu tham chiếu được dùng để lưu trữ chuỗi ký tự. Vì là một kiểu dữ liệu nên cách khai báo và sử dụng hoàn toàn tương tự các kiểu dữ liệu khác (đã trình bày trong bài [KIỂU DỮ LIỆU](#) và [BIẾN](#))


Hôm nay mình sẽ không bàn đến khai báo của nó nữa mà đi sâu vào các thuộc tính và phương thức mà lớp **String** hỗ trợ.

- **Thuộc tính** trong lớp **String**:

| Tên thuộc tính | Ý nghĩa |
|----------------|--|
| Length | Trả về một số nguyên kiểu int là độ dài của chuỗi. |

- Một số **phương thức** thường dùng trong lớp **String**:

| Tên phương thức | Ý nghĩa | Ghi chú |
|---|--|--|
| <code>String.Compare(string strA, string strB)</code> | So sánh 2 chuỗi <code>strA</code> và <code>strB</code> có bằng nhau hay không. Nếu bằng nhau thì trả về 0, nếu <code>strA > strB</code> thì trả về 1 trường hợp còn lại trả về -1 | Chúng ta có thể gọi phương thức so sánh này thông qua tên biến bằng cách: <code><tên biến>.CompareTo(string strB)</code> |
| <code>String.Concat(string strA, string strB)</code> | Nối 2 chuỗi <code>strA</code> và <code>strB</code> thành một chuỗi | Phương thức này tương tự như phép cộng chuỗi bằng toán tử cộng. |
| <code>IndexOf(char value)</code> | Trả về một số nguyên kiểu <code>int</code> là vị trí xuất hiện đầu tiên của ký tự <code>value</code> trong chuỗi. | Nếu như không tìm thấy thì phương thức sẽ trả về -1. |
| <code>Insert(int startIndex, string value)</code> | Trả về một chuỗi mới trong đó bao gồm chuỗi cũ đã chèn thêm chuỗi <code>value</code> tại vị trí <code>startIndex</code> . | |
| <code>String.IsNullOrEmpty(string value)</code> | Kiểm tra chuỗi <code>value</code> có phải là chuỗi <code>null</code> hoặc rỗng hay không. Nếu đúng thì trả về <code>true</code> ngược lại trả về <code>false</code> . |  |
| <code>LastIndexOf(char value)</code> | Trả về một số nguyên kiểu <code>int</code> là vị trí xuất hiện cuối cùng của ký tự <code>value</code> trong chuỗi. | |

| | | |
|--|---|--|
| <code>Remove(int startIndex, int count)</code> | Trả về một chuỗi mới đã gỡ bỏ count ký tự từ vị trí <code>startIndex</code> trong chuỗi ban đầu. | |
| <code>Replace(char oldValue, char newValue)</code> | Trả về một chuỗi mới đã thay thế các ký tự <code>oldValue</code> bằng ký tự <code>newValue</code> trong chuỗi ban đầu. |  |
| <code>Split(char value)</code> | Trả về một mảng các chuỗi được cắt ra từ chuỗi ban đầu tại những nơi có ký tự <code>value</code> | |
| <code>Substring(int startIndex, int length)</code> | Trả về một chuỗi mới được cắt từ vị trí <code>startIndex</code> với số ký tự cắt là <code>length</code> từ chuỗi ban đầu. | Nếu như bạn gọi <code>Substring</code> mà chỉ truyền vào giá trị <code>startIndex</code> thì mặc định phương thức sẽ cắt từ vị trí <code>startIndex</code> đến cuối chuỗi. |

Lưu ý:

- Các phương thức mà mình có ghi `String` phía trước là các phương thức gọi thông qua tên lớp. Các phương thức còn lại được gọi thông qua đối tượng.
- Các phương thức khi gọi sẽ tạo ra đối tượng mới rồi thao tác trên đối tượng đó chứ không thao tác trực tiếp với đối tượng đang xét. Vì thế nếu như các bạn gọi

C#:

```
string a = "HowKteam";

a.Substring(3, 1);
```

Thì biến `a` sau khi thực hiện lệnh `Substring` vẫn mang giá trị là `"HowKteam"`.

Nếu muốn biến `a` mang giá trị mới khi thực hiện `Substring` thì bạn phải gán ngược lại giá trị mới đó cho biến `a`:

C#:

```
a = a.Substring(3, 1);
```

- Bạn có thể xem 1 chuỗi là 1 mảng các ký tự. Như vậy ta hoàn toàn có thể truy xuất đến từng ký tự như truy xuất đến phần tử của mảng (Truy xuất đến một phần tử của mảng đã trình bày trong bài [MẢNG 1 CHIỀU TRONG C#](#))
- Ở trên chỉ là các phương thức hay dùng ngoài ra còn rất nhiều phương thức khác các bạn có thể tự khám phá thêm.

Ứng dụng lớp String vào việc xử lý chuỗi

Để hiểu rõ cách sử dụng các phương thức trên. Chúng ta cùng thực hiện chuẩn hoá một chuỗi họ tên của người dùng với các yêu cầu:

- Cắt bỏ hết các khoảng trắng dư ở đầu cuối chuỗi. Các từ cách nhau một khoảng trắng nếu phát hiện có nhiều hơn 1 khoảng trắng thì thực hiện cắt bỏ.
- Viết hoa chữ cái đầu tiên của mỗi từ, các chữ cái tiếp theo thì viết thường.

Ý tưởng:

- Cắt khoảng trắng dư ở đầu và cuối chuỗi thì ta có thể sử dụng phương thức **Trim**.
- Khoảng trắng ở giữa thì ta sẽ duyệt cả chuỗi nếu phát hiện có 2 khoảng trắng thì thay thế nó bằng 1 một khoảng trắng. Để làm điều này ta có thể dùng:
 - **IndexOf** để phát hiện khoảng trắng.
 - **Replace** để thay thế 2 khoảng trắng thành 1 khoảng trắng.
- Viết hoa chữ cái đầu và viết thường các chữ cái còn lại thì ta có thể cắt chuỗi họ tên ra thành các từ và ứng với mỗi từ ta thực hiện như yêu cầu đề bài. Để làm điều này ta có thể sử dụng:
 - **Split** để cắt ra các từ.
 - **Substring** để cắt ra các chữ cái mong muốn.
 - **ToUpper** để viết hoa và **ToLower** để viết thường.

Bây giờ các bạn hãy làm thử trước khi xem code tham khảo của mình nào!

Sau đây là code tham khảo để giải quyết các vấn đề trên:

C#:

```

/*
 * Khai báo 1 biến kiểu chuỗi tên là FullName
 * Khai báo 1 biến Result chứa kết quả chuẩn hoá chuỗi.
 * Giá trị biến FullName được nhập từ bàn phím.
 */
string FullName;
string Result = "";

Console.Write(" Mọi bạn nhập họ và tên: ");
FullName = Console.ReadLine();

/* Cắt các khoảng trắng dư ở đầu và cuối chuỗi */
FullName = FullName.Trim();

/*
 * Trong khi còn tìm thấy 2 khoảng trắng
 * thì thực hiện thay thế 2 khoảng trắng bằng 1 khoảng trắng
 */
while (FullName.IndexOf("  ") != -1)
{
    FullName = FullName.Replace("  ", " ");
}
/*
 * Cắt chuỗi họ tên ra thành mảng các từ.
 * Sau đó duyệt mảng để chuẩn hoá từng từ một.
 * Khi duyệt mỗi từ ta thực hiện cắt ra chữ cái đầu trên và lưu trong biến FirstChar
 * Cắt các chữ cái còn lại và lưu trong biến OtherChar.
 * Thực hiện viết hoa chữ cái đầu và viết thường các chữ cái còn lại.
 * Cuối cùng là lưu chữ vừa chuẩn hoá vào biến Result.
 */
string[] SubName = FullName.Split(' ');

for (int i = 0; i < SubName.Length; i++)
{
    string FirstChar = SubName[i].Substring(0, 1);
    string OtherChar = SubName[i].Substring(1);
    SubName[i] = FirstChar.ToUpper() + OtherChar.ToLower();
    Result += SubName[i] + " ";
}

Console.WriteLine(" Họ tên của bạn là: " + Result);

```

Trong code mình đã chú thích rất rõ ý nghĩa của từng đoạn lệnh, các bạn xem qua để hiểu hơn. Ở đoạn:

C#:

```

for (int i = 0; i < SubName.Length; i++)
{
    string FirstChar = SubName[i].Substring(0, 1);
    string OtherChar = SubName[i].Substring(1);
    SubName[i] = FirstChar.ToUpper() + OtherChar.ToLower();
    Result += SubName[i] + " ";
}

```

Chúng ta có thể viết gọn lại thành:

C#:

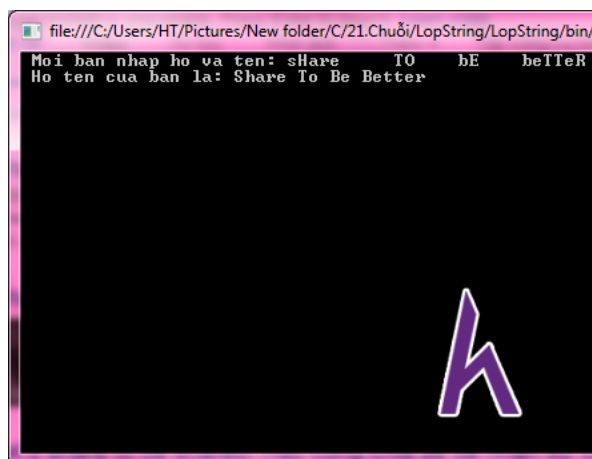
```

for (int i = 0; i < SubName.Length; i++)
{
    SubName[i] = SubName[i].Substring(0, 1).ToUpper() + SubName[i].Substring(1).ToLower();
    Result += SubName[i] + " ";
}

```

Nhưng viết như vậy đôi khi gây khó hiểu cho những bạn mới bắt đầu nên mình cố tình viết tường minh để các bạn dễ hiểu hơn.

Kết quả khi chạy chương trình trên:



Chương trình trên vẫn còn 1 lỗi nhỏ đó là nếu như bạn không nhấn phím **space** để tạo khoảng trắng mà nhấn phím **Tab** thì chương trình sẽ không ra kết quả như ý. Các bạn hãy thử vận dụng kiến thức đã học để giải quyết xem sao nhé!

Lớp StringBuilder trong C#

Đặc điểm

Lớp `StringBuilder` được .NET xây dựng sẵn giúp chúng ta thao tác trực tiếp với chuỗi gốc và giúp tiết kiệm bộ nhớ hơn so với lớp `String`.

Đặc điểm của `StringBuilder` là:

- Cho phép thao tác trực tiếp trên chuỗi ban đầu.
- Có khả năng tự mở rộng vùng nhớ khi cần thiết.
- Không cho phép lớp khác kế thừa (khái niệm về kế thừa sẽ được trình bày trong bài [KẾ THỪA TRONG C#](#)).

Từ 2 đặc điểm này đã làm nổi bật lên ưu điểm của `StringBuilder` so với `String` đó là ít tốn bộ nhớ. Cụ thể qua ví dụ sau:

C#:

```
string Value = "How";  
Value = Value + "Kteam";
```

Ở 2 dòng lệnh ta có thể thấy bộ nhớ sẽ được lưu trữ như sau:

- Đầu tiên tạo 1 vùng nhớ đối tượng kiểu `string` tên là `Value`.
- Tạo 1 vùng nhớ chứa giá trị `"Kteam"`.
- Khi thực hiện toán tử cộng trên 2 chuỗi sẽ tạo ra 1 vùng nhớ nữa để chứa giá trị chuỗi mới sau khi cộng.
- Cuối cùng là phép gán sẽ thực hiện trỏ đối tượng `Value` sang vùng nhớ chứa chuỗi kết quả của phép cộng.

Như vậy ta thấy sẽ có 1 vùng nhớ không sử dụng nhưng vẫn còn nằm trong bộ nhớ, đó là vùng nhớ chứa giá trị `"How"` – giá trị ban đầu của biến `Value`.

Đối với `StringBuilder` thì khác:

C#:

```
StringBuilder MutableValue = new StringBuilder("How");
MutableValue.Append("Kteam");
```

Ở 2 câu lệnh trên bộ nhớ sẽ lưu trữ như sau:

- Tạo một vùng nhớ cho đối tượng MutableValue chứa giá trị "How".
- Tạo một vùng nhớ chứa giá trị "Kteam".
- Mở rộng vùng nhớ của MutableValue để nối chuỗi "Kteam" vào sau chuỗi "How".

Rõ ràng là ta không tạo ra quá nhiều vùng nhớ và cũng không lãng phí bất cứ vùng nhớ nào.

Sử dụng

Cách khởi tạo 1 đối tượng `StringBuilder` có đôi chút khác so với `String`.

Cú pháp:

- Khởi tạo một đối tượng rỗng:

```
StringBuilder <tên biến> = new StringBuilder();
```

- Khởi tạo một đối tượng chứa 1 chuỗi cho trước:

```
StringBuilder <tên biến> = new StringBuilder(<chuỗi giá trị>);
```

Trong lớp `StringBuilder` có các phương thức như: `Remove`, `Insert`, `Replace` được sử dụng hoàn toàn giống như lớp `String`.

Chỉ có vài phương thức mới các bạn cần chú ý:

| Tên phương thức | Ý nghĩa |
|-----------------------------------|---|
| <code>Append(string Value)</code> | Nối chuỗi Value vào sau chuỗi ban đầu. |
| <code>Clear()</code> | Xoá bỏ toàn bộ nội dung trong đối tượng (lưu ý không phải xoá vùng nhớ của đối tượng) |
| <code>ToString()</code> | Chuyển đối tượng kiểu <code>StringBuilder</code> sang kiểu <code>String</code> . |

Lưu ý:

- Các bạn nhớ đây là đối tượng kiểu `StringBuilder` nên thao tác với chuỗi như gán, nối chuỗi, ... phải thông qua các phương thức chứ không thể thực hiện trực tiếp được.
- Giữa `String` và `StringBuilder` đều có cái hay riêng của nó. Tùy vào từng yêu cầu của bài toán mà nên sử dụng cho hợp lý, tránh lạm dụng quá nhiều 1 kiểu:
 - Thông thường đối với các bài toán đòi hỏi thao tác nhiều với chuỗi gốc như cộng chuỗi, chèn chuỗi, xoá bỏ một số ký tự, ... thì nên sử dụng `StringBuilder` để tối ưu bộ nhớ.
 - Còn lại thì nên sử dụng `String` để việc thao tác thuận tiện hơn.

Kết luận

Nội dung bài này giúp các bạn nắm được:

- Lớp `String` trong C#.
- Ứng dụng lớp `String` vào xử lý chuỗi.
- Lớp `StringBuilder` trong C#.

Bài sau chúng ta sẽ tìm hiểu về [STRUCT 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"**.

