03

Consultando ViaCEP Com Caelum Stella CSharp

Transcrição

Aprenderemos como fazer consultas ao **ViaCEP** usando da biblioteca `Caelum Stella CSharp`. Primeiramente, vamos extrair o trecho de código abaixo para o método `GetEndereco()`.

```
    string url = "https://viacep.com.br/ws/" + cep + "/json/";

    string result = new HttpClient().GetStringAsync(url).Result;
```

Com essa extração, nosso código ficará mais limpo.

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);

    Debug.WriteLine(result);
}

private static string GetEndereco(string cep)
{
    string url = "https://viacep.com.br/ws/" + cep + "/json/";
}
```

Para começar a trabalhar com o **Stella CSharp**, temos que procurá-la no *Manage NuGet Package...*, e instalar o pacote, como já fizemos antes.

Agora podemos declarar o componente ViaCEP, e com isso, podemos utilizar o método que é mais adequado no momento, a consulta de endereços JSON.

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);

    Debug.WriteLine(result);

    string enderecoJson = new ViaCEP().GetEnderecoJson(cep);
    Debug.WriteLine(enderecoJson);
}

private static string GetEndereco(string cep)
{
    string url = "https://viacep.com.br/ws/" + cep + "/json/";
}
```

Vamos rodar a aplicação para ver o resultado. Como podemos ver... temos o endereço em formato **JSON**, a partir do componente ViaCEP do `Caelum Stella CSharp`.

O que aconteceria se quiséssemos esse mesmo resultado, porém, no formato **XML**? Vamos declarar o componente ViaCEP e chamar o método `GetEnderecoXml()`. Sabendo que temos que instanciar novamente o ViaCEP, podemos extraí-lo para evitar a repetição de código.

Selecionando o trecho `new ViaCEP()` e teclando "Ctrl + .", conseguimos extrair em uma **localmente**.

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);

    Debug.WriteLine(result);

    ViaCEP viaCEP = new ViaCEP();
    strring enderecoJson = viaCEP.GetEnderecoJson(cep);
    Debug.WriteLine(enderecoJson);
}
```

A partir dessa extração, conseguimos chamar o `viaCEP` para obter o endereço em formato **XML**.

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);

    Debug.WriteLine(result);

    ViaCEP viaCEP = new ViaCEP();
    strring enderecoJson = viaCEP.GetEnderecoJson(cep);
    Debug.WriteLine(enderecoJson);

    string enderecoXml = viaCEP.GetEnderecoXml(cep);
    Debug.WriteLine(enderecoXml);
}
```

Como já esperávamos, o endereço foi retornado no formato XML. Em uma outra situação, como podemos fazer para obtermos um resultado assíncrono? Imagine que estamos em uma aplicação, e fazemos a requisição para obter o endereço a partir do CEP, só que não queremos esperar esse resultado pois precisamos fazer outras coisas.

Dada a situação, como utilizaríamos o componente **ViaCEP** do `Caelum Stella CSharp`?

Novamente precisamos fazer uma requisição para a chamada assíncrona, obtendo o endereço JSON, só que de uma forma diferente:

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);
```

```
    Debug.WriteLine(result);

    // extração do endereço no formato JSON

    // extração do endereço no formato XML

    viaCEP.GetEnderecoJsonAsync(cep);
}
```

Repare que o retorno desse método é diferente, pois ele retorna um objeto `Task`. Como estamos retornando uma task, armazenaremos esse resultado em uma variável chamada de `task`.

```
    var task = viaCEP.GetEnderecoJsonAsync(cep);
    Debug.WriteLine(task.Result);
```

Quando rodamos a aplicação, vimos que foi obtido o resultado que veio do `GetEnderecoJson` assíncrono.

O que acontece se acessarmos parte desse endereço, pegar somente o logradouro e o bairro, por exemplo?

Vamos obter o endereço a partir do ViaCEP, porém, dessa vez não queremos no formato JSON e nem no formato XML... Simplesmente, queremos um **objeto** endereço!

Utilizaremos o método `GetEndereco`, que retorna um **endereço**, uma instância da classe `Endereço`.

```
static void Main(string[] args)
{
    string cep = "01001000";
    string result = GetEndereco(cep);

    Debug.WriteLine(result);

    // extração do endereço no formato JSON

    // extração do endereço no formato XML

    // obtendo um resultado assíncrono

    var endereco = viaCEP.GetEndereco(cep);
}
```

Para imprimir essa variável no Debug, de forma que seja apresentado somente o **logradouro** e o **bairro**, o `WriteLine()` receberá uma string formatada.

```
var endereco = viaCEP.GetEndereco(cep);
Debug.WriteLine(string.Format("Logradouro: , Bairro: "));
```

Depois do logradouro e do bairro, temos que passar a posição entre `{}`. Em seguida, colocaremos a lista de parâmetros dessa string formatada, como um outro argumento.

```
Debug.WriteLine(string.Format("Logradouro: {0}, Bairro: {1}", endereco.Logradouro, endereco.Bai
```

Conseguimos acessar parte do nosso endereço. Rodaremos a aplicação para ver o resultado.

```
Logradouro: Praça da Sé, Bairro: Sé
```

Esse método é bastante útil, e não tínhamos esse recurso em nossa chamada utilizando `HttpClient`. O que acontece se o nosso CEP for inválido?

Como já fizemos antes, colocaremos novamente um dígito a mais, a fim de ver o resultado.

```
string cep = "010010007";
```

Ao rodarmos a aplicação, aparecerá um erro referente ao `HttpClient`, porém, ele não está sendo utilizado agora. Porém, o erro que está sendo apresentado é o *Bad Request* - uma informação que não ajuda. Por causa disso, vamos comentar essas linhas do método antigo em nosso projeto.

```
string cep = "010010007";
//string result = GetEndereco(cep);

//Debug.WriteLine(result);
```

Rodaremos novamente, agora ignorando o erro do método antigo. Temos já uma exceção diferente da anterior, o `InvalidZipCodeFormat` (o valor não recai no intervalo esperado), ou seja, o `Caelum Stella CSharp` pegou o CEP e viu que não era válido.

Diante disso, temos um tratamento mais granular, mais específico para esse problema do CEP.

Vimos como utilizar o ViaCEP do `Caelum Stella CSharp` para fazer um acesso mais robusto, simples, adequado e controlado das requisições que são feitas para o *web service* do ViaCEP.

Com todo o conhecimento adquirido até agora, faremos um contrato de trabalho temporário, assim seremos capazes de utilizar diversas funções brasileiras que abordamos no curso de C# Brasil!