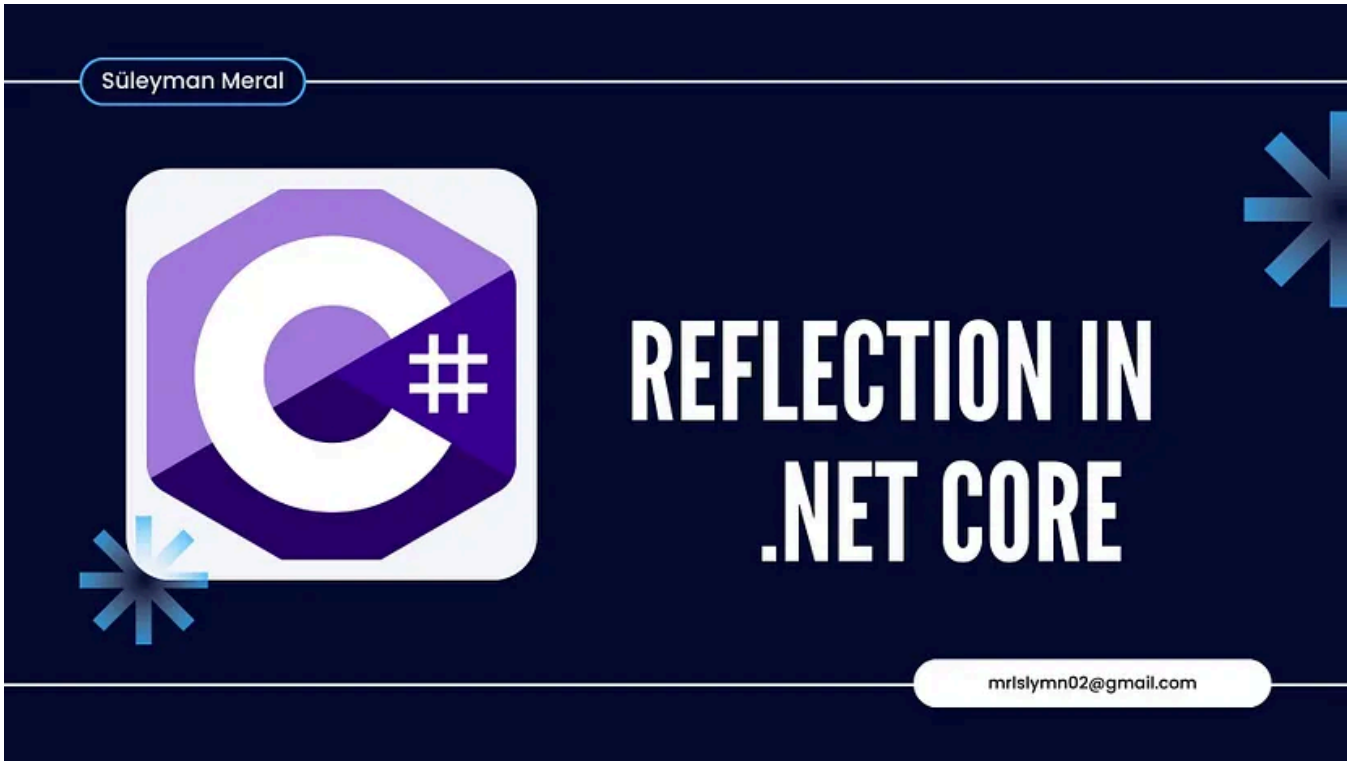
.NET CORE Reflection(Runtime'da Kodu Okumak ve Değiştirmek: Reflection'ın Gücü)

Süleyman Meral

4 min read · 16 hours ago

Share More



Herkese merhaba! Bu yazımda sizlere Reflection kavramından bahsedeceğim. Reflection Nedir? Niçin Kullanılır? Avantajları Ve Dezavantajları Nelerdir ? Bu sorular üzerinde duracağız ve bir örnek de yapacağız.

Reflection Nedir?

C# dilinde **Reflection**, bir programın çalışma zamanında (runtime) kendi yapısını (tiplerini, metodlarını, özelliklerini, vb.) incelemesine ve hatta bazı durumlarda değiştirmesine olanak sağlayan bir özelliktir.

Reflection, canlıda kodu analiz edip müdahale etmemizi sağlayan bir "ayna" gibidir.

Kodun kendisini **çalışma zamanında okuyabilen**, anlayan ve değiştiren sistemler yapmak istiyorsak, **reflection** vazgeçilmezdir.

Reflection Niçin Kullanılır?

- **Çalışma zamanında nesnelerin yapısını öğrenmek**
- Metot çağırmak (invoke) (Oyun Programlamada Çokça Kullanılır)
- Özelleştirilmiş davranışlar yazmak (örneğin plugin sistemi)
- Attribute'lara göre özel iş kuralları
- Test frameworkleri, ORM araçları, web frameworkleri

Avantajları Ve Dezavantajları Nelerdir?

Avantajları

- Kodun çalışma zamanında analiz edilmesini ve çalıştırılmasını sağlar. Bu sayede dinamiklik sağlar.
- Sınıfın, metodun, property'nin isimleri, türleri gibi bilgiler runtime'da erişilebilir.
- Özelleştirilmiş davranışlar attribute'larla işaretlenip kullanılabilir.

Dezavantajları

- Performansı düşüktür. Doğrudan kod çalıştırmaya göre çok yavaştır.
- Hatalar ancak uygulama çalıştırıldığında (runtime'da) ortaya çıkar.
- Kodun okunabilirliğini düşürebilir.
- Güvenlik riskleri doğurabilir

Şimdi ise bir reflection örneği yapalım.

Person adında bir class oluşturalım ve içine Name isimli bir property ve Hello isimli bir method ekleyelim.

```
public class Person
{
    public string? Name { get; set; }

    public void Hello()
```

```
        {
            Console.WriteLine($"Hello,{Name}");
        }
    }
```

Şimdi ise Reflection kullanarak runtime'da yani kod çalışıyor iken Person classının Methodlarına erişmeye çalışalım.

```
Type type = typeof(Person);
```

`typeof(Person)` => `Person` sınıfının tipini (Type objesi) alınır.

`Type` sınıfı, C#'ta bir nesnenin veya sınıfın yapısını temsil eder.

`Person` sınıfının yansımasını (`reflection`) kullanmak üzere bu tip bilgisi alınıyor.

```
MethodInfo[] methods = type.GetMethods();
foreach (var method in methods)
{
    Console.WriteLine("Method: " + method.Name);
}
```

MethhodInfo türünde methods isimli bir dizi oluşturduk ve bu dizinin elemanları type objesinin methodları olarak belirlendi.

Daha sonra foreach kullanarak bu methodları yazdırdık.

`MethodInfo[]` dizisi, her metodun bilgilerini içerir (isim, parametreler vs.).

```
Method: get_Name
Method: set_Name
Method: Hello
Method: GetType
Method: ToString
Method: Equals
Method: GetHashCode
```

Kodumuzu run ettiğimizde böyle bir çıktı elde edeceğiz. Görüldüğü üzere methodlar teker teker listelendi. Bazı methodlar C# da classlara default olan methodlardır. Örneğin daha iyi anlaşılabilmesi açısından classımıza bir property ve bir method daha ekelyelim ve çıktıya bakalım.

```
public class Person
{
    public string? Name { get; set; }
    public decimal Salary { get; set; }

    public void Hello()
    {
        Console.WriteLine($"Hello,{Name}");
    }
    public void Calculate()
    {
        Console.WriteLine($"Hello,{Salary}");
    }

}
```

```
Method: get_Name
Method: set_Name
Method: get_Salary
Method: set_Salary
Method: Hello
Method: Calculate
Method: GetType
Method: ToString
Method: Equals
Method: GetHashCode
```

Eklediğimiz method ve propertyle ait get ve set methodları da eklendi.

Şimdi ise runtime da Person sınıfından bir nesne oluşturup değer ataması yapalım.

```
object personInstance = Activator.CreateInstance(type);
```

`new Person();` ama reflection kullanılarak yapılır.

Runtime da Person classından personInstance isimli bir instance oluşturduk.

```
PropertyInfo prop = type.GetProperty("Name");
prop.SetValue(personInstance, "Süleyman");
```

PropertyInfo türünde prop isimli bir değişken oluşturduk ve Name propertyisine atadık. Daha sonra SetValue ile Propertymize bir değer atadık.

`GetProperty("Name")` ⇒ `Person` sınıfındaki `Name` özelliğini bulur.

`PropertyInfo` , o özelliğin bilgilerini içerir.

Dinamik oluşturulan `personInstance` nesnesinin `Name` özelliğine `"Süleyman"` değerini atar.

person.Name = "Süleyman"; Bu anlama gelir fakat reflection kullanarak yapılmıştır.

```
MethodInfo HelloMethod = type.GetMethod("Hello");
HelloMethod.Invoke(personInstance, null);
```

`GetMethod("SayHello")` ⇒ `SayHello` metodunu bulur.

`Invoke(...)` ⇒ bu metodu çağırır. (Bu yapı unity de çokça kullanılır)

`null` ⇒ metodun parametresi olmadığı için boş gönderildi.

```
Method: get_Name
Method: set_Name
Method: get_Salary
Method: set_Salary
Method: Hello
Method: Calculate
Method: GetType
Method: ToString
Method: Equals
Method: GetHashCode
Hello,Süleyman
```

Kodumuzun çıktısı bu şekilde görüntülenmektedir. Aynı şekilde Salary prop'una runtime da değer atayıp Calculate fonksiyonunu çağırabiliriz.

```
using ReflectionOrnek;
using System.Reflection;

Type type = typeof(Person);


MethodInfo[] methods = type.GetMethods();
foreach (var method in methods)
{
    Console.WriteLine("Method: " + method.Name);
}


object personInstance = Activator.CreateInstance(type);


PropertyInfo prop = type.GetProperty("Name");
prop.SetValue(personInstance, "Süleyman");



MethodInfo HelloMethod = type.GetMethod("Hello");
HelloMethod.Invoke(personInstance, null);
PropertyInfo salary = type.GetProperty("Salary");
salary.SetValue(personInstance, (decimal)80000.00);

MethodInfo calculate = type.GetMethod("Calculate");
calculate.Invoke(personInstance,null);
```

```
Method: get_Name
Method: set_Name
Method: get_Salary
Method: set_Salary
Method: Hello
Method: Calculate
Method: GetType
Method: ToString
Method: Equals
Method: GetHashCode
Hello,Süleyman
Hello,80000

C:\Users\suley\source\repos\ReflectionOrnek\ReflectionOrnek\bin\Debug\net8.0\ReflectionOrnek.exe (process 4680) exited w
ith code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Reflection kavramını bu şekilde ele aldık. Daha detaylı bir inceleme için YouTube kanalımda bir video da çekeceğim. Aşağıdaki linkten kanalıma ulaşabilirsiniz.

Herkese Kolay Gelsin !

👇

https://www.youtube.com/@suleymanmeral53/videos

Yazilim Gelistirme Yazılım Kodlama

Edit profile

Written by Süleyman Meral

1 Follower · 1 Following

Software Engineering Student/ Backend Developer

No responses yet

Süleyman Meral

What are your thoughts?

More from Süleyman Meral