Промышленное программирование

Лекция 15. Интроспекция и рефлексия

Основные определения и типы

Интроспекция — способность программы исследовать тип или свойства объекта во время работы программы.

Рефлексия — способность компьютерной программы изучать и модифицировать свою структуру и поведение во время выполнения.

```
namespace System.Reflection;
```

Assembly

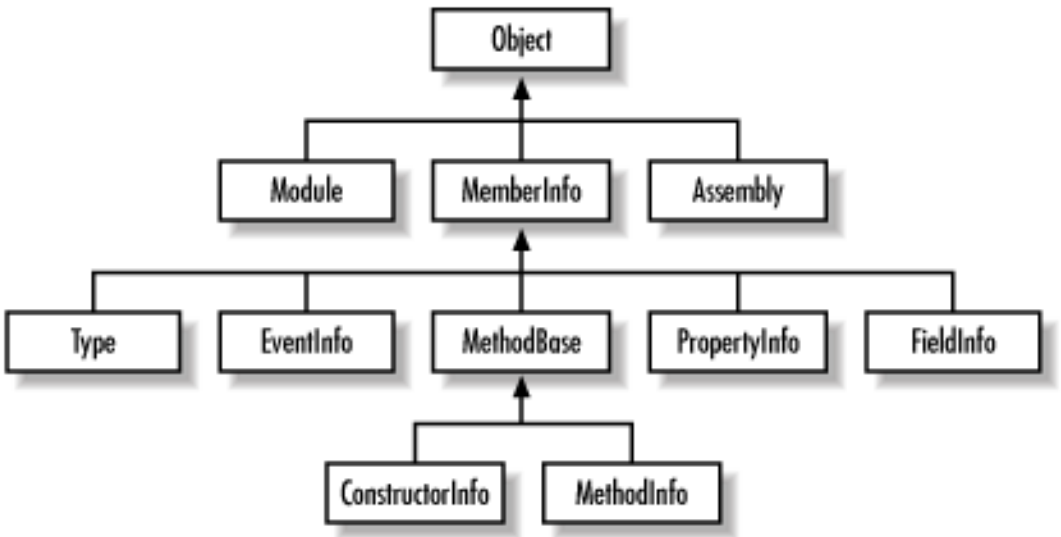
Type

FieldInfo

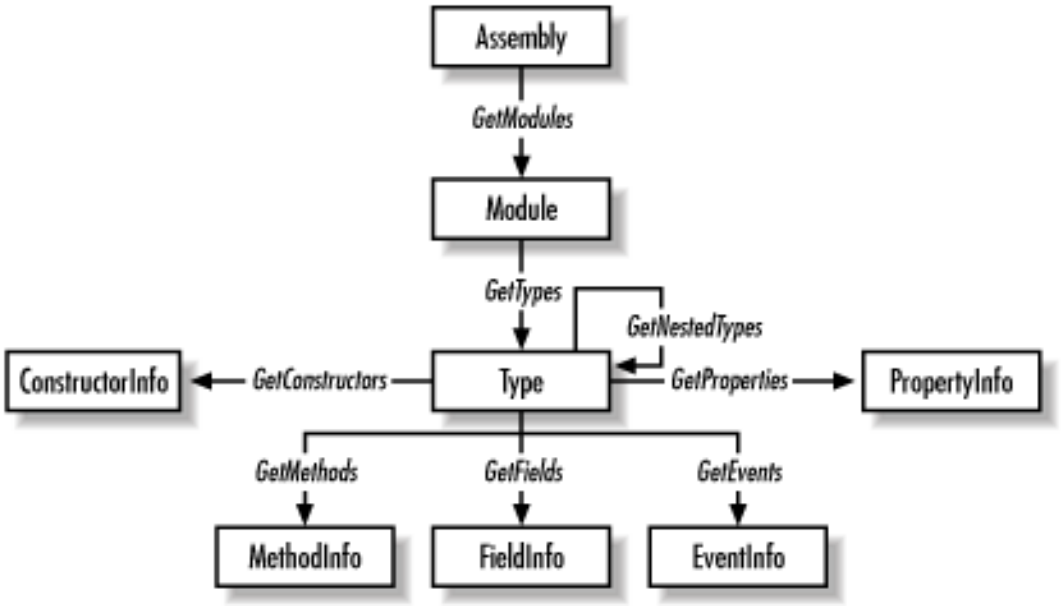
PropertyInfo

ConstructorInfo

MethodInfo



Навигация между рефлексивными объектами



Как получить объекты типа Assembly и Type

```
using System;
using System.Reflection;

Console.WriteLine(Assembly.GetEntryAssembly());
Console.WriteLine(Assembly.GetCallingAssembly());
Console.WriteLine(Assembly.GetExecutingAssembly());

var a = Assembly.LoadFrom("path-to-assembly.dll");

foreach (var type in a.GetTypes())
    Console.WriteLine(type.FullName);
```

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

int valObj = 5;
var refObj = new Dictionary<string, int>();
Console.WriteLine(valObj.GetType());
Console.WriteLine(refObj.GetType());

var intType = typeof(int);
var refType = typeof(Dictionary<string, int>);
Console.WriteLine(intType);
Console.WriteLine(refType);

GenericMethod(valObj);
GenericMethod(refObj);
static void GenericMethod<T>(T? obj)
{
    Console.WriteLine(typeof(T));
}
```

Методы Type для поиска конструктора

```
class Type
{
    public ConstructorInfo[] GetConstructors();
    public ConstructorInfo? GetConstructor(Type[] types);
    public ConstructorInfo? GetConstructor(BindingFlags bindingAttr, Type[] types);
    …
}

enum BindingFlags
{
    Default = 0,
    DeclaredOnly = 2,
    Instance = 4,
    Static = 8,
    Public = 16,
    NonPublic = 32,
    …
}
```

Пример вызова конструктора

```
using System;


var customerType = typeof(Customer);
var customerConstructor =
    customerType.GetConstructor(new Type[] { typeof(string), typeof(string) })
    ?? throw new InvalidOperationException();
var customer = (Customer)customerConstructor.Invoke(new object?[] { "Sergey", "Ivanov" });
Console.WriteLine(customer);

var defaultCustomer = Activator.CreateInstance<Customer>();
Console.WriteLine(defaultCustomer);


class Customer
{
    public string FirstName { get; set; } = "?";
    public string LastName { get; set; } = "?";

    public Customer() { }

    public Customer(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public override string ToString()
    {
        return $"{FirstName} {LastName}";
    }
}
```

Методы Type для поиска метода и примеры вызова метода

```
class Type
{
    public MethodInfo[] GetMethods();
    public MethodInfo? GetMethod(string name);
    public MethodInfo? GetMethod(string name, BindingFlags bindingAttr);
    public MethodInfo? GetMethod(string name, BindingFlags bindingAttr, Type[] types);
    public MethodInfo? GetMethod(string name, Type[] types);
    …
}
```

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


var list = new List<int>();

var method = list.GetType().GetMethod("Add");
method?.Invoke(list, new object?[] { 1 });
method?.Invoke(list, new object?[] { 2 });
method?.Invoke(list, new object?[] { 3 });

Print(list);


static void Print<T>(IEnumerable<T> items)
{
    foreach (var item in items)
        Console.WriteLine(item);
}
```

```
using System;


var method = typeof(int).GetMethod("Parse", new[] { typeof(string) });
var value = method?.Invoke(null, new object[] { "42" });

Console.WriteLine($"{value?.GetType()} {value}");
```

Методы Type для поиска свойства и пример использования

```
class Type
{
    public PropertyInfo[] GetProperties();
    public PropertyInfo? GetProperty(string name);
    public PropertyInfo? GetProperty(string name, BindingFlags bindingAttr);
    public PropertyInfo? GetProperty(string name, Type? ReturnType);
    public PropertyInfo? GetProperty(string name, Type[] types);
    public PropertyInfo? GetProperty(string name, Type? returnType, Type[] types);
    …
}
```

```
using System;


var customer = new Customer { FirstName = "Sergey", LastName = "Ivanov" };

var property = customer.GetType().GetProperty("FirstName");

Console.WriteLine(property?.GetValue(customer));

property?.SetValue(customer, "Ivan");

Console.WriteLine(property?.GetValue(customer));


class Customer
{
    public string FirstName { get; init; } = string.Empty;
    public string LastName { get; init; } = string.Empty;
}
```

Методы Type для поиска поля и пример использования

```
class Type
{
    public FieldInfo[] GetFields();
    public FieldInfo? GetField(string name);
    public FieldInfo? GetField(string name, BindingFlags bindingAttr);
    …
}
```

```
using System;
using System.Reflection;


var counter = new Counter();

var field = counter.GetType().GetField("_value", BindingFlags.Instance | BindingFlags.NonPublic);

Console.WriteLine(field?.GetValue(counter));

field?.SetValue(counter, 42);

Console.WriteLine(field?.GetValue(counter));


class Counter
{
    private int _value = 13;

    public int Value => _value;
}
```

Обобщённые классы в рефлексии

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

var map = new Dictionary<string, int>();



var mapType = map.GetType();
Console.WriteLine(mapType);
foreach (Type arg in mapType.GetGenericArguments())
    Console.WriteLine(arg);

// System.Collections.Generic.Dictionary`2[System.String,System.Int32]
// System.String
// System.Int32



var genericMapType = mapType.GetGenericTypeDefinition(); // typeof(Dictionary<,>)
var inverseMapType = genericMapType.MakeGenericType(typeof(int), typeof(string));
var inverseMap = Activator.CreateInstance(inverseMapType);

Console.WriteLine(genericMapType);
Console.WriteLine(inverseMapType);
Console.WriteLine(inverseMap);

// System.Collections.Generic.Dictionary`2[TKey, TValue]
// System.Collections.Generic.Dictionary`2[System.Int32, System.String]
// System.Collections.Generic.Dictionary`2[System.Int32, System.String]
```

Атрибуты и пример сериализации

```
using System;
using System.Reflection;
using System.Text.Json.Nodes;

var customer = new Customer { FirstName = "Sergey", MiddleName = "Alexandrovich", LastName = "Ivanov" };
Console.WriteLine(Serialize(customer));

static JsonObject Serialize(object obj)
{
    var json = new JsonObject();
    foreach (var prop in obj.GetType().GetProperties())
    {
        if (prop.GetCustomAttribute<IgnoreAttribute>() is not null)
            continue;
        var value = prop.GetValue(obj) ?? throw new InvalidOperationException();
        if (prop.PropertyType == typeof(string))
            json[prop.Name] = (string)value;
        else
            json[prop.Name] = Serialize(value);
    }
    return json;
}

[AttributeUsage(AttributeTargets.Property)]
class IgnoreAttribute : Attribute { }

class Customer
{
    public string FirstName { get; init; } = string.Empty;
    [Ignore] public string MiddleName { get; init; } = string.Empty;
    public string LastName { get; init; } = string.Empty;
}
```