Patron Prototype

Introducción

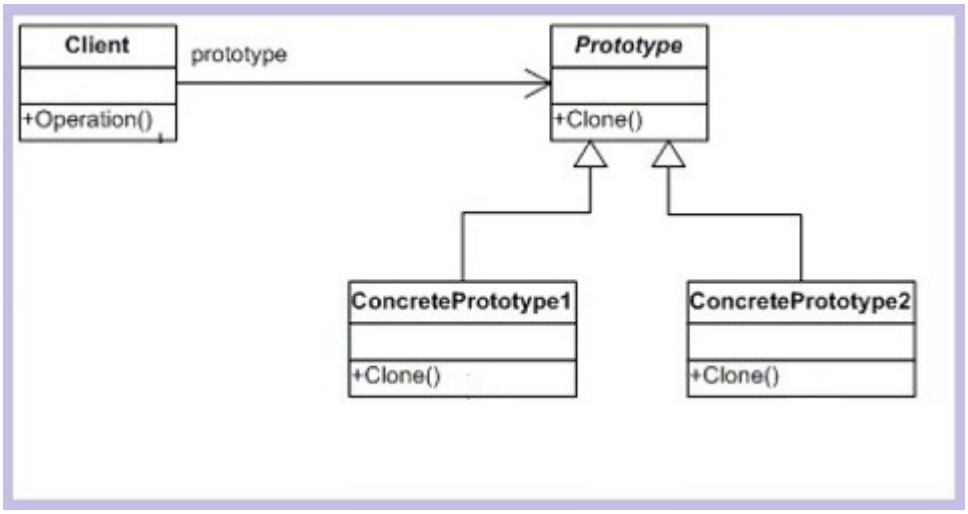
Este artículo habla sobre el patrón prototipo, cuando se debe utilizar. a continuación, vamos a ver cómo el patrón prototipo puede ser implementado en C #. Además, este artículo se discutirá acerca de la copia superficial y profunda copia en C # y cómo podemos implementar la interfaz ICloneable para implementar el patrón prototipo de una manera optimizada .Net.

Fondo

Hay muchas situaciones en nuestras aplicaciones en las que queremos tener copia de un objeto a partir del contexto y luego proceder con esta copia con un conjunto independiente de las operaciones. modelo prototipo es especialmente útil en estos escenarios donde podemos simplemente copiar un objeto de contexto de aplicación actual y luego proceder con ello independiente del objeto original.

GoF define el patrón prototipo como "Especificar el tipo de objetos para crear mediante una instancia prototípica, y crear nuevos objetos copiando este prototipo ." por lo que para visualizar este patrón veamos el diagrama de clases de este patrón.

Prototype: Se trata de una interfaz o clase abstracta que define el método para clonar en sí.
ConcretePrototype: Esta es la clase concreta que va a clonarse.
Client: El objeto de la aplicación que necesita la copia clonada del objeto.



```
using System;
namespace DoFactory.GangOfFour.Prototype.Structural
{

// MainApp test application
class MainApp
{

static void Main()
{
  // Create two instances and clone each
  ConcretePrototype1 p1 = new ConcretePrototype1("I");
  ConcretePrototype1 c1 = (ConcretePrototype1)p1.Clone();
  Console.WriteLine ("Cloned: {0}", c1.Id);
  ConcretePrototype2 p2 = new ConcretePrototype2("II");
  ConcretePrototype2 c2 = (ConcretePrototype2)p2.Clone();
  Console.WriteLine ("Cloned: {0}", c2.Id);
  // Wait for user
  Console.Read();
}

}
// "Prototype"
abstract class Prototype
{
private string id;
// Constructor
public Prototype(string id)
{
this.id = id;
}
// Property
public string Id
{
get{ return id; }
}
public abstract Prototype Clone();
}
// "ConcretePrototype1"
class ConcretePrototype1 : Prototype
{
// Constructor
public ConcretePrototype1(string id) : base(id)
{
}
public override Prototype Clone()
{
```

```
// Shallow copy
return (Prototype)this.MemberwiseClone();
}
}
// "ConcretePrototype2"
class ConcretePrototype2 : Prototype
{
// Constructor
public ConcretePrototype2(string id) : base(id)
{
}
public override Prototype Clone()
{
// Shallow copy
return (Prototype)this.MemberwiseClone();
}
}
}
```