عنوان: آشنایی با الگوی IOC یا Inversion of Control (واگذاری مسئولیت)
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۹/۲۵ ۱۱:۴۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns

کلاس Kid را با تعریف زیر در نظر بگیرید. هدف از آن نگهداری اطلاعات فرزندان یک شخص خاص می‌باشد:

```
namespace IOCBeginnerGuide
{
  class Kid
  {
      private int _age;
      private string _name;

      public Kid(int age, string name)
      {
          _age = age;
          _name = name;
      }

      public override string ToString()
      {
          return "KID's Age: " + _age + ", Kid's Name: " + _name;
      }
  }
}
```

اکنون کلاس والد را با توجه به اینکه در حین ایجاد این شیء، فرزندان او نیز باید ایجاد شوند؛ در نظر بگیرید:

```
using System;

namespace IOCBeginnerGuide
{
  class Parent
  {
      private int _age;
      private string _name;
      private Kid _obj;

      public Parent(int personAge, string personName, int kidsAge, string kidsName)
      {
          _obj = new Kid(kidsAge, kidsName);
          _age = personAge;
          _name = personName;
      }

      public override string ToString()
      {
          Console.WriteLine(_obj);
          return "ParentAge: " + _age + ", ParentName: " + _name;
      }
  }
}
```

و نهایتا مثالی از استفاده از آن توسط یک کلاینت:

```
using System;

namespace IOCBeginnerGuide
{
  class Program
  {
      static void Main(string[] args)
      {
          Parent p = new Parent(35, "Dev", 6, "Len");
```

```
            Console.WriteLine(p);

            Console.ReadKey();
            Console.WriteLine("Press a key...");
        }
    }
}
```

که خروجی برنامه در این حالت مساوی سطرهای زیر می‌باشد:

```
KID's Age: 6, Kid's Name: Len
ParentAge: 35, ParentName: Dev
```

مثال فوق نمونه‌ای از الگوی طراحی ترکیب یا composition می‌باشد که به آن Object Dependency یا Object Coupling نیز گفته می‌شود. در این حالت ایجاد شیء والد وابسته است به ایجاد شیء فرزند.

مشکلات این روش:
-1 با توجه به وابستگی شدید والد به فرزند، اگر نمونه سازی از شیء فرزند در سازنده‌ی کلاس والد با موفقیت روبرو نشود، ایجاد نمونه‌ی والد با شکست مواجه خواهد شد.
-2 با از بین رفتن شیء والد، فرزندان او نیز از بین خواهند رفت.
-3 هر تغییری در کلاس فرزند، نیاز به تغییر در کلاس والد نیز دارد (اصطلاحا به آن Dangling Reference هم گفته می‌شود. این کلاس آویزان آن کلاس است!).

چگونه این مشکلات را برطرف کنیم؟
بهتر است کار وهله سازی از کلاس Kid به یک شیء، متد یا حتی فریم ورک دیگری واگذار شود. به این واگذاری مسئولیت، delegation و یا IOC - inversion of control نیز گفته می‌شود.

بنابراین IOC می‌گوید که:
-1 کلاس اصلی (یا همان Parent) نباید به صورت مستقیم وابسته به کلاس‌های دیگر باشد.
-2 رابطه‌ی بین کلاس‌ها باید بر مبنای تعریف کلاس‌های abstract باشد (و یا استفاده از interface ها).

تزریق وابستگی یا Dependency injection
برای پیاده سازی IOC از روش تزریق وابستگی یا dependency injection استفاده می‌شود که می‌تواند بر اساس constructor injection ، setter injection و یا interface-based injection باشد و به صورت خلاصه پیاده سازی یک شیء را از مرحله‌ی ساخت وهله‌ای از آن مجزا و ایزوله می‌سازد.

مزایای تزریق وابستگی‌ها:
-1 گره خوردگی اشیاء را حذف می‌کند.
-2 اشیاء و برنامه را انعطاف پذیرتر کرده و اعمال تغییرات به آن‌ها ساده‌تر می‌شود.

روش‌های متفاوت تزریق وابستگی به شرح زیر هستند:

تزریق سازنده یا constructor injection :
در این روش ارجاعی از شیء مورد استفاده، توسط سازنده‌ی کلاس استفاده کننده از آن دریافت می‌شود. برای نمونه در مثال فوق از آنجائیکه کلاس والد به کلاس فرزندان وابسته است، یک ارجاع از شیء Kid به سازنده‌ی کلاس Parent باید ارسال شود.
اکنون بر این اساس تعاریف، کلاس‌های ما به شکل زیر تغییر خواهند کرد:

```
//IBuisnessLogic.cs
namespace IOCBeginnerGuide
{
  public interface IBuisnessLogic
  {
  }
}
```

```
//Kid.cs
namespace IOCBeginnerGuide
{
  class Kid : IBuisnessLogic
  {
      private int _age;
      private string _name;

      public Kid(int age, string name)
      {
          _age = age;
          _name = name;
      }

      public override string ToString()
      {
          return "KID's Age: " + _age + ", Kid's Name: " + _name;
      }
  }
}
```

```
//Parent.cs
using System;

namespace IOCBeginnerGuide
{
  class Parent
  {
      private int _age;
      private string _name;
      private IBuisnessLogic _refKids;

      public Parent(int personAge, string personName, IBuisnessLogic obj)
      {
          _age = personAge;
          _name = personName;
          _refKids = obj;
      }

      public override string ToString()
      {
          Console.WriteLine(_refKids);
          return "ParentAge: " + _age + ", ParentName: " + _name;
      }
  }
}
```

```
//CIOC.cs
using System;

namespace IOCBeginnerGuide
{
  class CIOC
  {
      Parent _p;

      public void FactoryMethod()
      {
          IBuisnessLogic objKid = new Kid(12, "Ren");
          _p = new Parent(42, "David", objKid);
      }

      public override string ToString()
```

```
        {
            Console.WriteLine(_p);
            return "Displaying using Constructor Injection";
        }
    }
}
```

```
//Program.cs
using System;

namespace IOCBeginnerGuide
{
  class Program
  {
      static void Main(string[] args)
      {
          CIOC obj = new CIOC();
          obj.FactoryMethod();
          Console.WriteLine(obj);

          Console.ReadKey();
          Console.WriteLine("Press a key...");
      }
  }
}
```

توضیحات:
ابتدا اینترفیس IBuisnessLogic ایجاد خواهد شد. تنها متدهای این اینترفیس در اختیار کلاس Parent قرار خواهند گرفت.
از آنجائیکه کلاس Kid توسط کلاس Parent استفاده خواهد شد، نیاز است تا این کلاس نیز اینترفیس IBuisnessLogic را پیاده سازی کند.
اکنون سازنده‌ی کلاس Parent بجای ارجاع مستقیم به شیء Kid ، از طریق اینترفیس IBuisnessLogic با آن ارتباط برقرار خواهد کرد.
در کلاس CIOC کار پیاده سازی واگذاری مسئولیت وهله سازی از اشیاء مورد نظر صورت گرفته است. این وهله سازی در متدی به نام Factory انجام خواهد شد.
و در نهایت کلاینت ما تنها با کلاس IOC سرکار دارد.

معایب این روش:
- در این حالت کلاس business logic، نمی‌تواند دارای سازنده‌ی پیش فرض باشد.
- هنگامیکه وهله‌ای از کلاس ایجاد شد دیگر نمی‌توان وابستگی‌ها را تغییر داد (چون از سازنده‌ی کلاس جهت ارسال مقادیر مورد نظر استفاده شده است).

تزریق تنظیم کننده یا Setter injection
این روش از خاصیت‌ها جهت تزریق وابستگی‌ها بجای تزریق آن‌ها به سازنده‌ی کلاس استفاده می‌کند. در این حالت کلاس Parent می‌تواند دارای سازنده‌ی پیش فرض نیز باشد.

مزایای این روش:
- از روش تزریق سازنده بسیار انعطاف پذیرتر است.
- در این حالت بدون ایجاد وهله‌ای می‌توان وابستگی اشیاء را تغییر داد (چون سر و کار آن با سازنده‌ی کلاس نیست).
- بدون نیاز به تغییری در سازنده‌ی یک کلاس می‌توان وابستگی اشیاء را تغییر داد.
- تنظیم کننده‌ها دارای نامی با معناتر و با مفهوم‌تر از سازنده‌ی یک کلاس می‌باشند.

نحوه‌ی پیاده سازی آن:
در اینجا مراحل ساخت Interface و همچنین کلاس Kid با روش قبل تفاوتی ندارند. همچنین کلاینت نهایی استفاده کننده از IOC

نیز مانند روش قبل است. تنها کلاس‌های IOC و Parent باید اندکی تغییر کنند:

```
//Parent.cs
using System;

namespace IOCBeginnerGuide
{
  class Parent
  {
      private int _age;
      private string _name;

      public Parent(int personAge, string personName)
      {
          _age = personAge;
          _name = personName;
      }

      public IBuisnessLogic RefKID {set; get;}

      public override string ToString()
      {
          Console.WriteLine(RefKID);
          return "ParentAge: " + _age + ", ParentName: " + _name;
      }
  }
}
```

```
//CIOC.cs
using System;

namespace IOCBeginnerGuide
{
  class CIOC
  {
      Parent _p;

      public void FactoryMethod()
      {
          IBuisnessLogic objKid = new Kid(12, "Ren");
          _p = new Parent(42, "David");
          _p.RefKID = objKid;
      }

      public override string ToString()
      {
          Console.WriteLine(_p);
          return "Displaying using Setter Injection";
      }
  }
}
```

همانطور که ملاحظه می‌کنید در این روش یک خاصیت جدید به نام RefKID به کلاس Parent اضافه شده است که از هر لحاظ نسبت به روش تزریق سازنده با مفهوم‌تر و خود توضیح دهنده‌تر است. سپس کلاس IOC جهت استفاده از این خاصیت اندکی تغییر کرده است.

ماخذ

نظرات خوانندگان

نویسنده: gg
تاریخ: ۱۳۸۸/۱۲/۰۷ ۱۹:۲۹:۵۶

خوب بود . موضوع پروژه منم همین است . خوشحال میشم بازم در این مورد مطلب بنوسید.