Ключевое слово var

Одно важное предупреждение: почти невозможно говорить о ключевом слове var и выведении неявных типов без демонстрации инициализации объектов или анонимных типов. Точно так же практически невозможно обсуждать инициализацию объектов или анонимные типы, не упоминая ключевого слова var. Все эти три расширения языка C# тесно связаны между собой.

Прежде чем описать в деталях каждое из этих средств языка - поскольку каждое из них описывает себя в терминах других - давайте взглянем на них всех вместе. Рассмотрим следующий оператор:

C#

```
var mySpouse = new { FirstName = "Alex", LastName = "Erohin" };
```

В этом примере объявляется переменная по имени mySpouse с использованием ключевого слова var. Ей присваивается значение анонимного типа, инициализированного с помощью новых средств инициализации объектов. Эта единственная строка кода пользуется преимуществами ключевого слова var, анонимных типов и инициализации объектов.

Короче говоря, ключевое слово var позволяет вывести тип данных локальной переменной на основе типа данных, которыми он инициализирован. Анонимные типы позволяют во время компиляции создавать новые типы данных — классы. В соответствии со словом анонимные, эти новые типы данных не имеют имен. Нельзя обычным образом создать анонимный тип данных, поскольку неизвестно, какие переменные-члены он содержит, и невозможно узнать, какие члены он содержит, если неизвестны их типы.

И, наконец, типы этих членов могут быть ясны только после инициализации. Средство инициализации объектов обрабатывает все эти тонкости.

На основе этой строки кода компилятор создаст новый анонимный класс, содержащий два общедоступных члена типа string: первый получит имя FirstName, второй — LastName.

Ключевое слово var неявно типизированной локальной переменной

С появлением в C# анонимных типов возникла новая проблема. Если создается переменная анонимного типа, то переменной какого типа ее можно присваивать? Рассмотрим в качестве примера следующий код:

C#

```
// Этот код не компилируется
??? unnamedTypeVar = new {firstArg = 1, secondArg = "Alex"};
```

Какой тип переменной должен быть объявлен для unnamedTypeVar? Это проблема. В Microsoft решили ее, создав ключевое слово var. Это новое ключевое слово информирует компилятор о том, что он должен неявно вывести тип переменной из инициализатора переменной. Это значит, что *переменная, объявленная с ключевым словом var должна иметь инициализатор*!

Пропуск инициализатора приводит к возникновению ошибки компиляции, например:

C#

```
var name;
```

Здесь компилятор сообщит об ошибке:

> *Implicitly-typed local variables must be initialized*
> *Неявно типизированная локальная переменная должна быть инициализирована*

Поскольку эти переменные подвергаются статической проверке типов во время компиляции, инициализатор необходим, чтобы компилятор мог неявно вывести тип из него. Попытка присвоить значение другого типа где нибудь в коде вызовет ошибку компиляции:

Пройди тесты

C# C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
var name = "Alex"; // Пока все хорошо
name = 1;       // Ошибка!
Console.WriteLine(name);
```

Этот код компилироваться не будет, потому что тип переменной name неявно выводится как string; однако производится попытка присвоить ей целочисленное значение. Как видите, компилятор обязывает переменную сохранять первоначальный тип.

Если вернуться к исходному примеру кода с назначением анонимного типа с использованием ключевого слова var, то код с дополнительной строкой для вывода значения переменной будет выглядеть так, как показано ниже:

C#

```
var unnamedTypeVar = new { firstArg = 1, secondArg = "Alex" };
Console.WriteLine(unnamedTypeVar.firstArg + ". " + unnamedTypeVar.secondArg);
```

Как видите, применение ключевого слова var обеспечивает статический контроль типа, а также гибкость поддержки анонимных типов. Это станет очень важно, когда речь пойдет об операциях проекции типов.

В примерах, приведенных до сих пор, использование ключевого слова var было обязательным, поскольку альтернативы не существовало. Если переменной необходимо присвоить объект анонимного типа, то такая переменная должна обязательно быть объявлена с ключевым словом var.

Однако допускается указывать var при каждом объявлении переменной, если только она правильно инициализируется. Из соображений сопровождаемости рекомендуется этим приемом не злоупотреблять. Понятно, что разработчики всегда должны знать тип данных, с которым они работают, но если вы помните действительный тип данных сейчас, то будет ли это так, когда вы вернетесь к этому коду полгода спустя? Как насчет других разработчиков, которые впоследствии могут заняться сопровождением?

В целях сопровождаемости кода избегайте использования ключевого слова var лишь потому, что это удобно. Применяйте его только там, где это необходимо, например, в случае присваивания переменной объекта анонимного типа.

Выражения инициализации объектов и коллекций

Из-за потребности в динамических типах данных, которые позволяют создавать анонимные типы, возникает необходимость изменения способа инициализации объектов и коллекций. Поскольку выражения теперь представлены в виде лямбда-выражений или деревьев выражений, инициализация объектов и коллекций упростилась.

Инициализация объектов

Инициализация объектов позволяет указывать значения инициализации для общедоступных полей и свойств класса во время создания экземпляра. Для примера рассмотрим следующий код:

C#

```
public class Address
  {
    public string address;
    public string city;
    public string state;
    public string postalCode;
  }
```

До появления средства инициализации объектов в C#, при отсутствии специализированного конструктора, объект типа Address нужно было инициализировать так, как показано ниже:

C#

```
Address address = new Address();
address.address = "105 Elm Street";
address.city = "Atlanta";
address.state = "GA";
address.postalCode = "30339";
```

Такой подход был бы очень неудобным в лямбда-выражениях. Предположим, что нужно запросить значения из источника данных и с помощью операции Select спроецировать определенные члены на объект Address:

Пройди тесты

C# C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
// Этот код не скомпилируется
IEnumerable<Address> addresses = somedatasource
  .Where(a => a.State = "GA")
  .Select(a => new Address(???)???};
```

Просто отсутствовал бы удобный способ инициализировать члены вновь сконструированного объекта Address. Но не беспокойтесь! На помощь пришла инициализация объектов. Вы можете возразить, что проблема решается простым созданием конструктора, который принимал бы все эти инициализирующие значения при конструировании экземпляра объекта. Да, это так - иногда. Но это сопряжено со сложностями. И как быть в случае анонимного типа? Не проще ли создавать экземпляры объектов, как показано ниже:

C#

```
Address address = new Address
      {
        address = "105 Elm Street",
        city = "Atlanta",
        state = "GA",
        postalCode = "30339"
      };
```

Это может решить проблему в лямбда-выражении. Кроме того, помните, что возможности инициализации объектов могут применяться где угодно, а не только в запросах LINQ.

При использовании инициализации объекта компилятор создает экземпляр объекта с помощью конструктора класса без параметров, после чего инициализирует именованные члены указанными значениями. Любые члены, которые не указаны, получают значения по умолчанию, принятые для своих типов данных.

Инициализация коллекций

Поскольку одной только инициализации объектов не достаточно, кто-то в Microsoft должен был сказать: "А как насчет коллекций?". Инициализация коллекций позволяет указывать инициализирующие значения для коллекции — как это делается для объектов. Пример инициализации коллекции приведен ниже:

C#

```
List<string> presidents = new List<string> { "Adams", "Arthur", "Buchanan" };
      foreach (string president in presidents)
      {
        Console.WriteLine(president);
      }
```

В дополнение к использованию инициализации коллекций с LINQ, инициализированные коллекции часто удобно создавать в коде, где не присутствуют запросы LINQ.

Анонимные типы

Создание API-интерфейса на уровне языка для обобщенных запросов данных было затруднено в C# из-за нехватки возможностей создания новых типов данных во время компиляции. Если необходимо, чтобы запросы данных извлекали первоклассные элементы на уровне языка, для этого язык должен обладать возможностью создания первоклассных элементов данных уровня языка, которыми в C# являются классы. Поэтому спецификация языка C# теперь включает возможность динамического создания новых безымянных классов и объектов этих классов. Классы такого рода известны как *анонимные типы*.

Анонимный тип не имеет имени и генерируется компилятором на основе инициализации создаваемого экземпляра объекта. Поскольку класс не имеет имени типа, любые переменные, присваиваемые объекту анонимного типа, должны иметь какой-то способ объявления. Именно в этом состоит предназначение ключевого слова var в C#.

Анонимный тип незаменим при проектировании (projecting) новых типов данных с использованием операций Select или SelectMany. Без анонимных типов при вызове этих операций должны были всегда существовать предопределенные именованные классы. Было бы очень неудобно создавать именованные классы для каждого запроса. Выше обсуждался следующий код создания экземпляра и инициализации объекта:

C#

```
Address address = new Address
      {
        address = "105 Elm Street",
        city = "Atlanta",
        state = "GA",
        postalCode = "30339"
      };
```

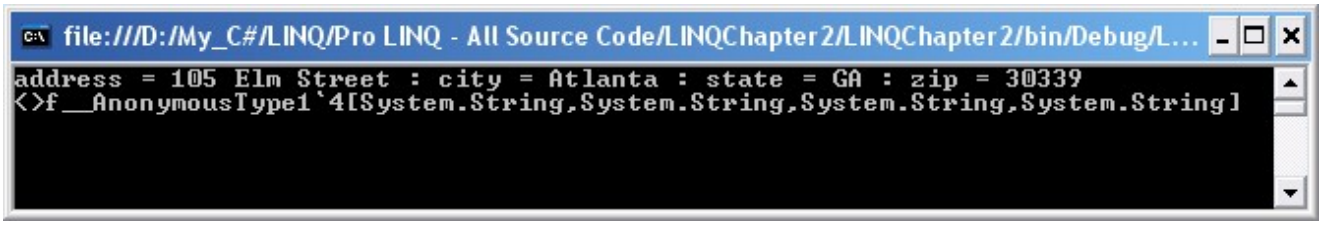
Если бы вместо использования именованного класса Address возникло желание применить анонимный тип, нужно было бы просто опустить имя класса. Однако тогда не получится сохранить вновь созданный объект в переменной типа Address, потому что он уже не будет переменной типа Address. В таком случае он относился бы к сгенерированному типу, который известен только компилятору. Поэтому пришлось бы изменить также и тип переменной address. Здесь снова на помощь приходит ключевое слово var:

C#

```
var address = new
      {
        address = "105 Elm Street",
        city = "Atlanta",
        state = "GA",
        postalCode = "30339"
      };

      Console.WriteLine("address = {0} : city = {1} : state = {2} : zip = {3}",
        address.address, address.city, address.state, address.postalCode);

      Console.WriteLine("{0}", address.GetType().ToString());
```



Назад (1_3.php)	3	4	5	Вперед (1_5.php)