Подсказки типа для object-переменных

admin
February 4 edited February 23

В object-переменной может храниться значение любого типа. Но можно явно задавать тип, который может хранить такая переменная. В первую очередь это нужно для ускорения работы reflection-стрелок, чтобы интерпретатор, заранее зная тип объекта, мог найти его члены при загрузке кода.
Подсказки типа делаются с помощью машинописного обратного апострофа (`) и константы с типом. Обратный апостроф на русской клавиатуре находится на букве Ё.

```
const object Int32=int;
`Int32 x=5; //на самом деле x - переменная типа object
WriteLine(x);
x=10L; //Ошибка при загрузке кода:  Can't put value of type 'long' into variable of type 'object/hint:System.Int32'.
```

В тексте ошибки мы видим, что тип x - 'object/hint:System.Int32'. Т.е. это object с подсказкой типа Int32. Чтобы присвоить такой переменной любое значение нужно использовать оператор :=, который не будет делать конвертирования (как обычно он делает), а просто запишет значение. Понятно, что делать так можно только, если вы точно знаете, что тип значения подходит для переменной.
Пример выше бессмысленный т.к. в OverScript есть тип int. А вот, например, типа uint (UInt32) нет. Напишем пример с этим типом.

```
const object UInt32=typeof("System.UInt32");
`UInt32 x=`UInt32\5, y=`UInt32\4; //то же, что и `UInt32 x=(`UInt32)5, y=(`UInt32)4;
WriteLine(x+y); //9 //x+y - это op_Addition(x, y)

static `UInt32 op_Addition(`UInt32 x, `UInt32 y){ //это перегрузка оператора сложения для `UInt32
    return `UInt32\(x.ToLong()+y.ToLong()); //складываем как long и приводим к `UInt32
}
```

Приведение типа в этом примере работает с конвертированием. Привести можно и так:

```
`UInt32 x=5.As(`UInt32);  //вернёт null, если не получится. Можно и так: `UInt32 x=5.As(UInt32);
```

Посмотрим ещё пример, демонстрирующий преимущество переменных с подсказкой типа перед обычными object-переменными при рефлекшене через стрелку (->) :

```
const object Point=typeof("System.Drawing.Point, System.Drawing");
`Point p=Point.Create(10, 20); //интерпретатор знает, что в p должен находиться объект типа Point
WriteLine(p->X); //10
WriteLine(Expr(p->X)); //@TGetValue(#Int32 X#,int,p)
//А теперь посмотрим, что будет без подсказки типа
```

```
object p2=Point.Create(10, 20); //интерпретатор не знает, объект какого типа в p2
WriteLine(p2->X); //10
WriteLine(Expr(p2->X)); //@GetMemberValue(p2,"X")
```

Видим, что для p->X интерпретатор использует более быструю функцию TGetValue, которая получает значение заранее найденного поля/свойства. А вот для p2->X используется GetMemberValue, которая ищет поле/свойство при каждом вызове, что замедляет работу примерно на 30%.
В примере выше есть строка:

```
`Point p=Point.Create(10, 20);
```

Можно написать и так:

```
`Point p=`Point.Create(10, 20);
```

В ней используется оператор =, а не := потому, что Create умеет возвращать object с подсказкой типа, если передаваемый ей тип - константа или базовый тип. Сейчас только несколько функций умеют так. В будущем я сделаю, чтобы все базовые функции, возвращающие object, по возможности возвращали с подсказкой типа. Тогда можно будет всегда использовать =, что снизит риск присваивания переменной некорректного значения. Если передаваемый функции Create тип не константа, то нужно использовать приведение типа либо оператор :=, который не будет делать проверки соответствия типов:

```
const object Point=typeof("System.Drawing.Point, System.Drawing");
object type=Point; //хоть Point и константа, type будет обычным object-ом
`Point p=`Point\type.Create(10, 20); //можно так: `Point p=(`Point)type.Create(10, 20);
//или так:
`Point p:=type.Create(10, 20);
```

Оператор := работает быстрее, поэтому в высоконагруженных операциях лучше использовать его.

Подсказки типа для массивов

```
const object UInt32=typeof("System.UInt32");
`UInt32[] arr={`UInt32\10, `UInt32\20, `UInt32\30};
WriteLine(arr); //System.Object[]
```

Как видим, массив имеет тип object[].

```
const object UInt32=typeof("System.UInt32");
//`UInt32[] arr=`UInt32[]\Array(UInt32, 10, 20, 30); //ошибка потому, что Array вернёт массив UInt32[], а не object[]!
`UInt32[] arr=`UInt32[]\Array(UInt32, 10, 20, 30).ToObjectArray();
WriteLine(Join(' ', arr)); //10 20 30
```

Приведение типа в данном случае проверяет типы элементов в массиве object[]. Поэтому, массив нужно сначала конвертировать в object[] (это делает функция ToObjectArray).
Для массивов элементов ссылочных типов возможно приведение без конвертирования в object[].

```
const object String=typeof("System.String");
`String[] arr=`String[]\new string[]{"hello", "world"};
WriteLine(arr); //System.String[]
//А если просто инициализируем, то получим object[]:
`String[] arr2={"hello", "world"};
WriteLine(arr2); //System.Object[]
```

Формально arr это object[]-переменная, но она ссылается на массив string[]. Это возможно потому, что C#, на котором написан OverScript, позволяет такое. Думаю, что многие про это не знают. Можете проверить на C#: `object[] arr = new string[]{ "hello", "world" };`.

Закрепим: массив с подсказкой типа - это либо object[]-массив, либо массив конкретного типа, если этот тип ссылочный. Если элементы значимого типа, то массив всегда должен быть типа object[].
Приведение типа с подсказкой типа ничего не конвертирует, а просто проверяет типы и возвращает сам массив. Я не стал делать поддержку конвертирования, чтобы не было путаницы, где приведение возвращает ссылку на оригинальный массив, а где создаёт новый массив.
Функция Is при проверке с подсказкой типа проверяет массив по принципу, изложенному выше.

```
const object String=typeof("System.String");
string[] strArr=new string[]{"hello", "world"};
object[] objArr=new object[]{"hello", "world"};
WriteLine(strArr.Is(`String[])); //True
WriteLine(objArr.Is(`String[])); //True

`String[] arr=`String[]\strArr; //это string[]
`String[] arr2=`String[]\objArr; //это object[]
WriteLine(arr.Is(`String[])); //True
WriteLine(arr2.Is(`String[])); //True
```

Теперь для массивов значимых типов:

```
const object Int32=typeof("System.Int32");
int[] intArr=new int[]{10, 20, 30};
object[] objArr=new object[]{10, 20, 30};
WriteLine(intArr.Is(`Int32[])); //False //потому, что не object[]!
WriteLine(objArr.Is(`Int32[])); //True

//`Int32[] arr=`Int32[]\intArr; //ошибка!
`Int32[] arr=`Int32[]\intArr.ToObjectArray();
`Int32[] arr2=`Int32[]\objArr;
```

```
WriteLine(arr.Is(`Int32[])); //True
WriteLine(arr2.Is(`Int32[])); //True
```

Есть нюанс:

```
const object Int32=typeof("System.Int32");
WriteLine(`Int32[]); //System.Int32[]
```

Здесь `` `Int32[] `` возвращает тип Int32[], а не object[]. Но в `arr.Is(`Int32[])` проверяется, является ли массив массивом object[] с элементами типа Int32. То же самое и для **функции As**:

```
const object Int32=typeof("System.Int32");
object[] arr={10, 20, 30};
`Int32[] arr2=arr.As(`Int32[]); //мут нужно именно As(`Int32[]), а не As(Int32[]).
WriteLine(arr2); //System.Object[]
int[] arr3=new int[]{10, 20, 30};
arr2=arr3.As(`Int32[]); //As вернёт null потому, что arr3 не object[]
WriteLine(arr2==null); //True
```

Для массива ссылочных значений:

```
const object StringBuilder=typeof("System.Text.StringBuilder");
object[] sbArr={StringBuilder.Create("hello"), StringBuilder.Create("world")};
`StringBuilder[] arr=sbArr.As(`StringBuilder[]);
WriteLine(arr+": "+Join(' ', arr)); //System.Object[]: hello world

object sbArr2=StringBuilder.Array(StringBuilder.Create("hello"), StringBuilder.Create("world")); //это массив StringBuilder[], а не object[]
arr=sbArr2.As(`StringBuilder[]); //сработает, т.к. переменная object[] может ссылаться на массивы со  значениями ссылочных типов (StringBuilder - ссылочный)
WriteLine(arr+": "+Join(' ', arr)); //System.Text.StringBuilder[]: hello world
```

Напомню, что `StringBuilder.Array(...)` - это то же самое, что и `Array(StringBuilder, ...)`, как и `StringBuilder.Create(...)` - это `Create(StringBuilder, ...)`. Так со всеми функциями делать можно.

Как я показывал выше, любой массив легко превратить в массив object[] с помощью функции ToObjectArray, которая к тому же умеет конвертировать значения в указанный тип:

```
const object UInt32=typeof("System.UInt32");
byte[] arr={10, 20, 30};
`UInt32[] arr2=`UInt32[]\arr.ToObjectArray(UInt32); //ToObjectArray конвертирует byte-значения в UInt32
WriteLine(Join(' ', arr2)); //10 20 30
```

Вместо ToObjectArray можно использовать ConvertArray:

```
`UInt32[] arr2=`UInt32[]\arr.ConvertArray(UInt32).ConvertArray(object);
```

Если объект, из которого нужно сделать массив, не массив, но реализует IEnumerable, то нужно использовать функцию ToArray:

```
`UInt32[] arr2=`UInt32[]\someCollection.ToArray(UInt32).ToObjectArray();
```

Возможно, у кого-то возник вопрос, почему, например, `ToArray(UInt32)`, а не `ToArray(`UInt32)`? Можно и так и так писать. Оба варианта возвращают одно значение. А вот писать UInt32[] нельзя, т.к. интерпретатор пример это за обращение к элементу массива с неуказанным индексом. В приведении типа нужно всегда писать с backtick-ом (`). Лучше всегда с ним писать, чтобы сразу было видно, что это не просто переменная, а константа с типом.
