Теория типов
и типизация в .NET

Комментарий к слайду

В данной лекции будут рассмотрены вопросы, относящиеся к истории развития, идеологии, математическому основанию и обзору возможностей типизированной комбинаторной логики и теории типов – математической формализации, моделирующей типы выражений в языках программирования.

Современные языки программирования и .NET: II семестр
Лекция 6: Теория типов и типизация в .NET

Содержание лекции

Комментарий к слайду

В ходе лекции будут рассмотрены важнейшие научные исследования, относящиеся к эволюции подходов к математическому моделированию существенного для любого подхода к программированию аспекта, а именно, типа.

Прежде всего, будут сформулированы интуитивное и более строгое определения типа.

Затем будет представлена классификация систем типизации в языках программирования с указанием места в ней изучаемых в курсе языков.

Далее будут подробно исследованы особенности системы типизации Common Type System в среде Microsoft .NET.

Особое внимание будет уделено реализации механизмов организации и управления типами, а именно, преобразованиям типов и пространствам имен.

Иерархия типов Microsoft .NET будет проиллюстрирована примерами описаний типов данных для случаев типов-значений и ссылочных типов.

Лекция завершится обзором литературы для более глубокого исследования материала.

Важнейшие работы в области типизации

1960-е – Р. Хиндли (Roger Hindley) исследовал типизацию в комбинаторной логике для моделирования языков функционального программирования со строгой типизацией

1969 – Р. Хиндли исследовал полиморфные системы типов

1978 – Р. Милнер (Robin Milner) предложил расширенную систему полиморфной типизации для ML

Комментарий к слайду

Напомним ход эволюции теорий, лежащих в основе современного подхода к типизации и теории типов.

В 60-х г.г. Р. Хиндли (Roger Hindley) исследовал типизацию в комбинаторной логике. При этом основной проблемой, стоящей перед исследователем, было моделирование языков функционального программирования со строгой типизацией, к каковым, в частности, относятся изучаемые нами языки SML и C#.

Позднее, в конце 60-х г.г., тот же ученый исследовал полиморфные системы типов, т.е. такие системы типов, в которых возможны параметризованные функции или функции, имеющие переменный тип.

Наконец, в 70-х г.г. Р. Милнер (Robin Milner) предложил практическую реализацию расширенной системы полиморфной типизации для языка функционального программирования ML, давшего начало языку программирования SML.

Общие сведения о типизации

Тип (сорт) – относительно устойчивая и независимая совокупность элементов, которую можно выделить во всем рассматриваемом множестве (предметной области).

Задать тип T (как и любое множество) возможно:

1) явным перечислением элементов;
2) формализацией общих свойств элементов предметной области $d \in D$ посредством ***индивидуализирующей*** предикатной ***функции*** Ψ, значение которой истинно, если элемент принадлежит данному типу и ложно в противном случае: $T = \{d: D|\Psi\}$

Комментарий к слайду

Перейдем к изложению начальных сведений из теории типов и типизации языков программирования.

В математике принято называть типами (или, иначе, сортами) относительно устойчивые и независимые совокупности элементов, которые можно выделить во всем рассматриваемом множестве (предметной области). Заметим, что разделение элементов предметной области на типы или сорта во многом является условным и носит субъективный характер, т.к. зависит от эксперта в этой области.

Тип, подобно множеству, может определяться двояко.

Во-первых, возможно определение типа посредством явного перечисления всех элементов, принадлежащих типу (заметим, что такой подход применяется и в математике, и в программировании, где существуют так называемые перечислимые типы).

Другим способом определения типа T является формализация общих свойств тех элементов d из предметной области D, которые объединяются в этот тип, посредством задания индивидуализирующей предикатной функции Ψ, значение которой истинно, если элемент принадлежит данному типу и ложно в противном случае:

$$T = \{d: D|\Psi\}.$$

Типы в математике (1)

Чистой системой типов называется семейство ламбда-исчислений, в которых каждый элемент характеризуется тройкой <S, A, R>, где:

S – подмножество констант, называемых сортами;

A – множество аксиом вида c:s, где c-константа, s-сорт;

R – множество троек сортов, определяющих возможные функциональные пространства и их сорта для системы

Приписывание ламбда-терму M ***типа*** T обозначим как

#M |— T (читается: «ламбда-терм M имеет тип T»)

Комментарий к слайду

При более формальном подходе к теории типов и типизации в связи с исчислением ламбда-конверсий следует определить чистую систему типов.

Чистой системой типов называется семейство ламбда-исчислений, в которых каждый элемент характеризуется тройкой <S, A, R>, где:

S – подмножество констант, называемых сортами;

A – множество аксиом вида c : s, где c является константой, а s является сортом;

R – множество троек сортов, определяющих возможные функциональные пространства и их сорта для системы.

Далее введем обозначение, характеризующее то обстоятельство, что тот или иной объект является типизированным, или, иначе говоря, что тому или иному объекту приписан тип.

В частности, для ламбда-терма M приписывание ему типа T обозначим как

$$\#M \vdash T$$

и будем в таком случае говорить, что ламбда-терм M имеет тип T.

Типы в математике (2)

Система типов формируется следующим образом:

1) задается множество ***базисных*** типов δ_1, δ_2, …;
2) всякий базисный тип считается типом;
3) если a и b считаются типами, то функция из a в b считается типом и имеет тип $a \to b$.

В основе теории типов лежит принцип ***иерархичности***: производные типы содержат базисные как подмножества.

Это справедливо и для языков программирования (аналогично строятся иерархии классов в ООП).

Комментарий к слайду

При более общем подходе (который верен и для математики, и для программирования), система типов формируется следующим образом.

Во-первых, задается множество базисных типов (обозначим их символами d_1, d_2, и так далее).

Во-вторых, примем соглашение, что всякий базисный тип считается типом.

Наконец, условимся, что если a и b считаются типами, то функция из a в b также считается типом и при этом имеет тип $a \to b$.

Заметим, что в основе теории типов лежит принцип иерархичности, который заключается в том, что производные типы содержат базисные как подмножества.

Этот принцип построения остается справедливым и для языков программирования. В частности, иерархии классов в объектно-ориентированных языках программирования формируются аналогично вышеприведенному построению математической системы типов.

Преимущества типизации

Модель предметной области лучше структурирована, существует иерархия сортов элементов

Манипулирование элементами более целенаправленно, разнородные элементы обрабатываются различным образом, однородные - единообразно

В случае строгой типизации несоответствия типов фиксируются до выполнения программы, гарантируя отсутствие смысловых ошибок и безопасность кода

Комментарий к слайду

Рассмотрев введение в математическую теорию типов и подходы к типизации в языках программирования, представим в концентрированном виде те преимущества, которые отличают языки программирования и формальные теории с типами.

Прежде всего, отметим то бесспорное преимущество типизированных исчислений, что при таком подходе моделируемая предметная область области лучше структурирована, чем в том случае, если отсутствует сегментация на типы. Типизация структурирует предметную область по иерархическому принципу.

Введение типизации облегчает и упорядочивает не только восприятие, но и управление предметной областью. Манипулирование типизированными элементами носит более целенаправленный характер, причем появляется возможность обрабатывать разнородные сущности предметной области различным образом, а однородные (или, точнее, однотипные) – единообразно.

Перейдем к языкам программирования и практике проектирования и реализации программных систем. В случае построения языка программирования по принципу строгой типизации несоответствия типов фиксируются до начала этапа выполнения программы (на этапе контроля соответствия типов в ходе трансляции), что гарантирует отсутствие семантических (смысловых) и логических ошибок и безопасность программного кода.

Классификация систем типизации

1. Строгая
2. Сильная
3. Слабая
4. Полиморфная
5. С контролем соответствия типов во время компиляции
6. С контролем соответствия типов во время выполнения

Комментарий к слайду

Напомним классификацию систем типизации в языках программирования.

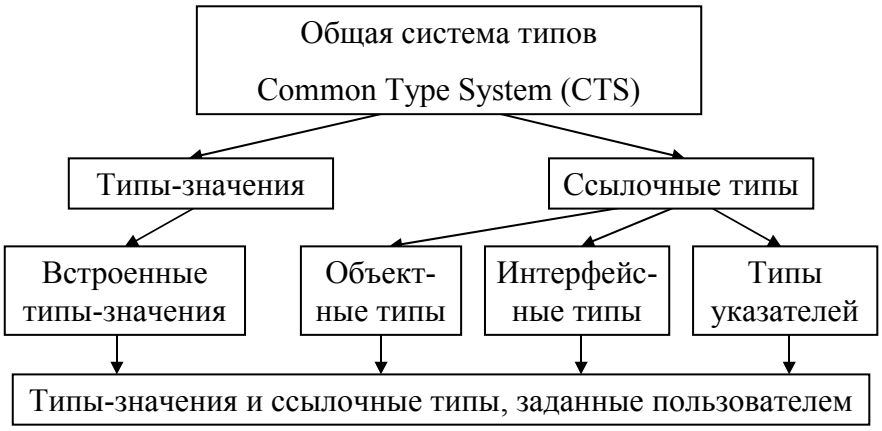
Исторически наиболее распространенной для языков программирования является строгая типизация. При таком формировании системы типов в языке в любой момент существования любого языкового объекта существует однозначное соответствие между объектом и его типом. Другими словами, можно запрограммировать функцию, определяющую тип объекта, подобную ранее рассмотренной нами функции `typeof` языка программирования C#. Строго типизированными являются классические императивные языки программирования Pascal, FORTRAN, PL/1 и др. Отметим, что классический вариант языка программирования C не является строго типизированным.

Сильная типизация необходима для обеспечения корректности связывания переменных со значениями до выполнения программы. Таким образом, каждое выражение языка программирования с сильной типизацией, по успешном завершении анализа корректности типизации является корректно типизированным. Примером языка программирования с сильной типизацией может служить SML. Использование механизма выводимости типов гарантирует сильную типизацию даже при отсутствии строгости. Язык программирования, не имеющий сильной системы типизации, может быть назван языком со слабой типизацией.

Еще одним важным видом систем типизации языков программирования является полиморфная типизация. При таком подходе допустимы выражения переменного типа (скажем, функция упорядочения списка с неопределенным типом элементов).

Процедура контроля соответствия типов (type-checking) может быть реализована как во время компиляции (compile time), т.е. более безопасным образом, так и во время выполнения (run time) программы, что потенциально менее безопасно для программного кода.

Иерархия типов в .NET



Комментарий к слайду

Рассмотрим более подробно обобщенную систему типизации, принятую в .NET.

Как очевидно из схемы, Common Type System представляет собой иерархию, в которой стрелки указывают в сторону уменьшения общности типа.

Типы иерархии CTS подразделяются на ссылочные типы (reference type) и типы-значения (value type).

Особенностями ссылочных типов являются необходимость использования указателей на типизированные объекты, а также централизованное хранение и освобождение памяти («сборка мусора»).

Одной из характеристик типов-значений является то обстоятельство, что они не участвуют в наследовании. Кроме того, типы-значения копируются при присваивании значения.

В свою очередь, ссылочные типы могут принимать одну из трех форм:

1) объектные типы (object type);
2) интерфейсные типы (interface type);
3) типы указателей (pointer type).

В случае отсутствия стандартного типа нужной формы, пользователь имеет возможность конструирования собственного типа, который может быть как ссылочным типом, так и типом-значением.

Система CTS обеспечивает безопасную типизацию, т.е. гарантирует отсутствие побочных эффектов (переполнение оперативной памяти компьютера, некорректное преобразование типов и т.д.).

Управление типами в CTS

- Типы могут использоваться после инициализации (с учетом метода вызова, свойств get и set и т.д.)
- Над типами могут совершаться преобразования (как явным, так и неявным образом)
- Типы объединяются в совокупности (пространства имен, файлы, сборки)
- Важнейшими подкатегориями системы типизации в .NET являются ссылочные типы (reference type) и типы-значения (value type)

Комментарий к слайду

Исследуем особенности управления типами в системе типизации Common Type System технологии Microsoft .NET.

Из соображений безопасности программного кода Microsoft .NET типы Common Type System, также как объекты и классы, могут использоваться после инициализации. При этом необходимо учитывать особенности вызывающего метода, а также определенных методов доступа `get` и изменения `set`.

Подобно конкретизации значений типов при определении значения типовой переменной, над элементами типов могут осуществляться преобразования из одного типа данных в другой (например, из строки в число или наоборот). При этом такие преобразования могут инициироваться как программистом (т.е. происходить явно), так и системой программирования (т.е. происходить неявным образом).

Иерархия типов в соотнесении с Microsoft .NET может быть представлена в форме той или иной совокупности (в частности, в виде пространства имен, файла или сборки).

Как уже отмечалось, наиболее крупными и важными подкатегориями иерархической системы типизации в Microsoft.NET являются ссылочные типы (reference type) и типы-значения (value type).

Ссылочные типы и типы-значения в .NET и C# (1)

- Типы-значения:
  - непосредственно содержат объекты данных;
  - не могут быть пустыми (null)
- Ссылочные типы:
  - содержат ссылки на объекты данных;
  - могут быть пустыми (null)

Пример:

```
int i = 25;
string s = "John_Smith";
```

Комментарий к слайду

Рассмотрим более подробно особенности реализации ссылочных типов и типов-значений в языке программирования C#.

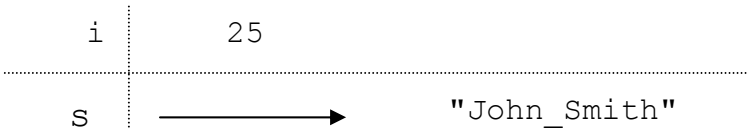
Прежде всего, типы-значения, в отличие от ссылочных типов, содержат непосредственно объекты данных. Кроме того, типы-значения не могут быть пустыми (т.е. принимать значение `null`).

Ссылочные типы, в противоположность типам-значениям, содержат не сами объекты, а лишь ссылки на них и могут принимать значение `null`.

В качестве иллюстрации особенностей реализации ссылочных типов и типов-значений в языке программирования C# рассмотрим следующий фрагмент программ на языке C#:

```
int i = 25;
string s = "John_Smith";
```

В первой строке программы происходит означивание целочисленной константы (т.е. типа-значения), а во второй – строковой (т.е. типа-ссылки). В результате при выполнении первой строки программы происходит связывание со значением в памяти, а во втором – со ссылкой, т.е. указателем на область памяти, как на схеме.



Ссылочные типы и типы-значения в .NET и C# (2)

- Типы-значения:
  - элементарные: `int i; float x;`
  - перечислимые: `enum State { Off, On }`
  - структурные: `struct Point {int x,y;}`
- Ссылочные типы:
  - корневые: `object`
  - строковые: `string`
  - классы: `class Foo: Bar, IFoo {...}`
  - интерфейсы: `interface IFoo: IBar {...}`
  - массивы: `string[] a = new string[10];`
  - делегаты: `delegate void Empty();`

Комментарий к слайду

Проиллюстрируем иерархию системы типов Common Type System программной среды Microsoft .NET примерами из языка программирования C#, конкретизирующими подтипы для типов-значений и ссылочных типов:

При этом типы-значения распадаются на следующие подтипы:

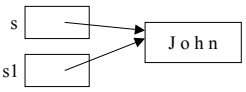
- элементарные (в частности, целочисленные и вещественные):
  `int i; float x;`
- перечислимые (в частности, моделирующие переключатели):
  `enum State { Off, On }`
- структурные (в частности, моделирующие точки на плоскости):
  `struct Point {int x,y;}`

В свою очередь, ссылочные типы подразделяются на следующие подтипы:

- корневой подтип (указатели на произвольные объекты в иерархии типов в Common Type System):
  `object`
- строковые (указатели на строки символов):
  `string`
- классы (указатели на объекты типа `class`):
  `class Foo: Bar, IFoo {...}`
- интерфейсы (указатели на объекты типа `interface`):
  `interface IFoo: IBar {...}`
- массивы (в частности, указатели на строки из 10 символов):
  `string[] a = new string[10];`
- делегаты (усовершенствованные указатели на функцию):
  `delegate void Empty();`

Сопоставление ссылочных типов и типов-значений

	Типы-значения	Ссылочные типы
Переменная содержит	значение	ссылку на значение
Переменная хранится	в стеке	в куче
Значение по умолчанию	0, false, '\0'	null
Оператор присваивания	копирует значение	копирует ссылку
Пример	int i = 25; int j = i;	string s = “John"; string s1 = s;



Комментарий к слайду

Ранее в ходе изложения настоящего курса уже обсуждались механизмы boxing и unboxing, реализованные в различных средах разработки программного обеспечения Microsoft .NET. Напомним, что эти механизмы используются для преобразования типов выражений языков программирования из ссылочных в типы-значения и обратно.

Рассмотрим более подробно реализацию двух основных семейств типов данных, а именно, ссылочных типов и типов-значений, применительно к языку программирования C#. Для определенности рассмотрим случай одного из простейших объектов языка программирования C#, а именно, переменной.

В соответствии с названиями, переменная в случае использования типов-значений содержит собственно значение, а при использовании ссылочных типов – не само значение, а лишь ссылку (указатель) на него.

Местом хранения переменной, определенной как тип-значение, является стек, а определенной как ссылочный тип – «куча» (последнее необходимо для динамического выделения и освобождения памяти для хранения переменной произвольным образом).

Значением, которым переменная инициализируется по умолчанию (необходимость выполнения этого требования диктуется идеологией безопасности Microsoft .NET) в случае определения посредством типа-значения является 0 (для целого или вещественного типа данных), false (для логического типа данных), '\0' (для строкового типа данных), а в случае определения посредством ссылочного типа – значение пустой ссылки null.

При выполнении оператора присваивания в случае переменной-значения копируется значение, а в случае переменной-ссылки – ссылка.

Приведенный далее пример иллюстрирует различия в реализации типов-ссылок и значений.

Элементарные типы языка C# (в сопоставлении с языком SML)

C#	CTS	SML	Диапазон
sbyte	System.SByte	---	-128 .. 127
byte	System.Byte	byte	0 .. 255
short	System.Int16	int	-32768 .. 32767
ushort	System.UInt16	word	0 .. 65535
long	System.Int64	---	-2^{63} .. $2^{63}-1$
float	System.Single	real	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	---	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	---	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	bool	true, false
char	System.Char	char	Символ (в коде unicode)

Комментарий к слайду

Перейдем к более подробному рассмотрению основных типов, входящих в состав Common Type System (CTS) Microsoft .NET. При этом прикладное исследование языков программирования произведем в форме сопоставления отображений фрагментов систем типизации языков программирования SML и C# в систему типов CTS.

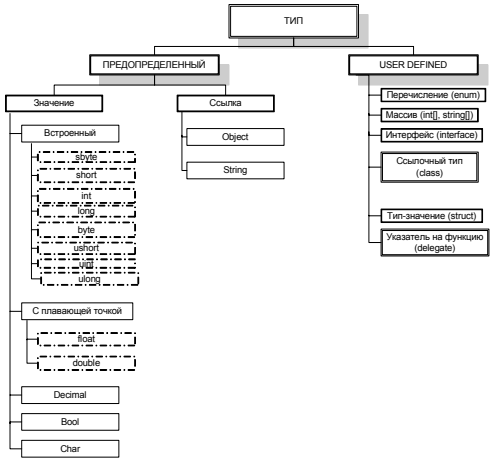
Для обеспечения большей наглядности сопоставления сведем отображения в следующую таблицу:

C#	CTS	SML	Диапазон
sbyte	System.SByte	---	-128 .. 127
byte	System.Byte	byte	0 .. 255
short	System.Int16	int	-32768 .. 32767
ushort	System.UInt16	word	0 .. 65535
long	System.Int64	---	-263 .. 263-1
float	System.Single	real	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	---	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	---	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	bool	true, false
char	System.Char	char	Символ (в коде unicode)

Даже из предварительного анализа таблицы видно, что система типизации языка программирования C# значительно богаче по сравнению с языком программирования SML. Можно заметить, что всякому типу языка программирования SML соответствует некоторый тип языка программирования C#, и их названия зачастую совпадают или являются подобными друг другу.

Наконец, отметим, что все без исключения типы обоих языков программирования однозначно отображаются в систему типизации Microsoft .NET, верхним иерархическим элементом которой является пространство имен System.

Иерархия типов языка C# (фрагмент)



Комментарий к слайду

Данная схема иллюстрирует перечень классов Common Type System среды проектирования и реализации программного обеспечения Microsoft .NET применительно к языку программирования C#.

В частности, все многообразие типов можно разделить на предопределенные (заранее заданные системой программирования) и определенные пользователем (user defined).

К последним относятся перечисления, массивы, классы, интерфейсы и делегаты (указатели на функцию).

Предопределенные типы делятся на ссылочные типы (объекты и символьные строки) и типы-значения (встроенные – короткие и длинные целые со знаком и без знака, а также числа с плавающей точкой – с одинарной и двойной точностью).

Соглашения о преобразовании типов в .NET

- Неявные преобразования (происходят автоматически, всегда успешно завершаются, без потери точности)
- Явные преобразования (требуют вызова, могут завершаться ошибкой, а также приводить к потере точности)
- Оба типа преобразований могут быть инициированы пользователем

```
int x = 25;
long y = x;                          // неявное
short z = (short)x;                  // явное
double d = 3.141592536;
float f = (float)d;                  // явное
long l = (long)d;                    // явное
```


Комментарий к слайду

Как уже отмечалось, над элементами типов могут осуществляться преобразования из одного типа данных в другой (например, из строки в число или наоборот). При этом такие преобразования могут инициироваться как программистом (т.е. происходить явно), так и системой программирования (т.е. происходить неявным образом).

Неявные преобразования инициируются Common Type System и происходят автоматически. При этом результат неявного преобразования всегда успешен и не приводит к потере точности.

Явные преобразования инициируются программистом или пользователем приложения, а следовательно, требуют явного вызова и при этом могут завершаться ошибкой, а также приводить к потере точности.

Приведем ряд примеров преобразований типов на языке программирования C#:

```
int x = 25;
long y = x;                    // неявное
short z = (short)x;            // явное
double d = 3.141592536;
float f = (float)d;            // явное
long l = (long)d;              // явное
```

Заметим, что система Common Type System среды Microsoft .NET обеспечивает безопасную типизацию, т.е. гарантирует отсутствие побочных эффектов (переполнение оперативной памяти компьютера, некорректное преобразование типов и т.д.). Заметим также, что как явные, так и неявные преобразования типов могут инициироваться пользователем.

Пространства имен в .NET (1)

- Пространство имен (namespace) предоставляет средства уникальной идентификации типа
- Дает возможность логически структурировать систему типов
- Пространства имен могут объединять сборки
- Пространства имен могут быть вложенными
- Между пространствами имен и файлами нет однозначного соответствия
- Полное имя типа содержит все пространства имен

Комментарий к слайду

Как показали исследования системы типизации Common Type System среды проектирования и реализации программных систем Microsoft .NET, эта иерархия имеет внушительный объем.

С целью повышения эффективности организации описания типов и манипулирования ими в программном обеспечении, работающем под управлением среды Microsoft .NET, вводится понятие пространства имен (namespace).

Пространством имен будем называть механизм среды Microsoft .NET, предназначенный для идентификации типов объектов языков программирования и среды реализации.

Значение механизма пространств имен состоит в том, что появляется возможность логической структуризации системы типизации Common Type System в среде разработки приложений Microsoft .NET.

Описания пространств имен по аналогии описаниями типов данных размещаются в файлах.

Перечислим основные свойства, которыми характеризуются пространства имен в среде Microsoft .NET:

1) пространства имен могут объединять различные сборки;
2) пространства имен могут быть вложенными друг в друга;
3) между пространствами имен и файлами не существует однозначного соответствия (т.е. отображения, переводящего название пространства имен в имя файла);
4) полное имя типа должно содержать все необходимые пространства имен.

Пространства имен в .NET (2)

Пример описания пространств имен
(в комментариях приведены полные имена):

```
namespace N1 {           // N1
  class C1 {             // N1.C1
    class C2 {           // N1.C1.C2
    }
  }
  namespace N2 {         // N1.N2
    class C2 {           // N1.N2.C2
    }
  }
}
```


Комментарий к слайду

Для иллюстрации применения механизма пространств имен в среде программирования Microsoft .NET приведем развернутый пример описания пространств имен на языке программирования C#:

```
namespace N1 {       // N1
  class C1 {         // N1.C1
    class C2 {            // N1.C1.C2
    }
  }
  namespace N2 {     // N1.N2
    class C2 {       // N1.N2.C2
    }
  }
}
```

Рассмотренный пример содержит описания пространств двух имен: пространства `N1` с описанием классов `C1` и `C2` и пространства `N2` с описанием класса `C2`.

Заметим, что в комментариях к каждой строке программы на языке C# приведены полные наименования пространств имен. Так, для обращения к классу `C2`, описанному в пространстве имен `N1`, нужно использовать полное имя `N1.C1.C2`, а для обращения к классу `C2`, описанному в пространстве имен `N2` – полное имя `N1.N2.C2`.

Таким образом, при адекватном употреблении полных квалификационных наименований пространств имен удается избежать коллизии обозначений типов.

Пространства имен в .NET (3)

- Оператор `using`:
- позволяет использовать типы без указания полного имени;
- может использоваться с полным именем;
- также используется для указания альтернативных имен (alias).

```
using N1;
C1 a;             // Имя N1. является неявным
N1.C1 b;          // Полное имя
C2 c;             // Ошибка: имя C2 не определено
N1.N2.C2 d;       // Один из (под)классов C2
C1.C2 e;          // Еще один из (под)классов C2
using C1 = N1.N2.C1;
using N2 = N1.N2;
C1 a;             // Соответствует имени N1.N2.C1
N2.C1 b;          // Соответствует имени N1.N2.C1
```


Комментарий к слайду

Очевидно, что при проектировании и реализации масштабных программных комплексов используется весьма значительное количество идентификаторов и риск коллизии обозначений при использовании полных квалификационных имен многократно возрастает.

Оказывается, что непременное использование полных имен типов в среде программирования Microsoft .NET является избыточным требованием.

Для экономии трудозатрат и во избежание коллизий обозначений при разработке крупных программных систем в языке программирования C# предусмотрен оператор `using`, к рассмотрению которого и переходим.

Для иллюстрации применения механизма ликвидации коллизии обозначений типов в пространствах имен среды программирования Microsoft .NET посредством оператора `using` приведем развернутый пример на языке программирования C#:

```
using N1;
C1 a;               // Имя N1. является неявным
N1.C1 b;            // Полное имя
C2 c;               // Ошибка: имя C2 не определено
N1.N2.C2 d;         // Один из (под)классов C2
C1.C2 e;            // Еще один из (под)классов C2
using C1 = N1.N2.C1;
using N2 = N1.N2;
C1 a;               // Соответствует имени N1.N2.C1
N2.C1 b;            // Соответствует имени N1.N2.C1
```

Заметим, что оператор `using` позволяет использовать типы как с без указанием полного имени, так и без него, а также для указания альтернативных имен (alias).

Пространства имен и ссылки

- В Visual Studio для проекта могут быть указаны ссылки
- Каждая ссылка идентифицирует уникальную сборку
- Передается C# компилятору по ссылке (`/r` или `/reference`)
  `csc HelloWorld.cs /reference:System.WinForms.dll`
- Пространства имен дают возможность сокращенного именования на уровне языка программирования
  - (устраняют необходимость многократного повторения полного имени)
- Ссылки указывают на сборку для использования в проекте

Комментарий к слайду

Еще одним аспектом проектирования и реализации крупных программных комплексов под управлением среды Microsoft .NET является потенциальная коллизия имен типов в рамках масштабных проектов.

Для решения этой проблемы в среде разработки программного обеспечения Microsoft Visual Studio для каждого из проектов проекта могут быть указаны так называемые ссылки, каждая из которых идентифицирует уникальную сборку – самодостаточную единицу для компиляции и выполнения в рамках программного проекта.

Ссылки передаются компилятору в ходе трансляции программы на том или ином языке программирования под управлением среды Microsoft .NET по ссылке (при этом используется опция `/r` или `/reference`). Скажем, команда на компиляцию файла `HelloWorld` с исходным текстом программы на языке C# со ссылкой на пространство имен `System.WinForms.dll` будет иметь вид:

```
csc HelloWorld.cs /reference:System.WinForms.dll
```

Как видно из приведенного примера, механизм ссылок реализует идентификацию сборки для использования в программном проекте.

Подводя итоги обсуждения механизмов управления пространствами имен среды проектирования и реализации программного обеспечения Microsoft .NET, отметим, что пространства имен предоставляют возможность сокращенного именования типов объектов на уровне языка программирования. Таким образом, устраняется необходимость многократного повторения полного имени объекта и существенно уменьшается риск коллизии обозначений, что значительно экономит трудозатраты при проектировании и реализации сложных программных систем и комплексов.

Преимущества типизации на платформе .NET

1. Использование централизованной, унифицированной системы типизации Common Type System (CTS)
2. Строгое соответствие между примитивными типами языков программирования и базовыми классами .NET; встроенная поддержка примитивных типов большинством компиляторов для .NET
3. Явное разделение на ***ссылочные типы*** (используются через указатель; централизованно хранятся и освобождаются) и ***типы-значения*** (не участвуют в наследовании; при присваивании значения копируются)
4. Гибкий и надежный механизм преобразования типов-значений в ссылочные (***boxing***) и обратно (***unboxing***)

Комментарий к слайду

Подводя итоги рассмотрения основных аспектов теории типов и типизации в языках программирования, можно сделать следующие выводы.

Во-первых, теории с типами и языки программирования с типизацией имеют значительно более высокую вычислительную мощность, а, следовательно, более высокую теоретическую и прикладную значимость.

Во-вторых, технологическая платформа .NET обеспечивает ряд несомненных дополнительных преимуществ перед другими известными платформами в отношении системы типов.

В частности, технология .NET использует централизованную, унифицированную, интегрированную систему типизации Common Type System (CTS), общую для всех языков программирования, реализуемых на данной платформе.

Кроме того, в рамках технологической платформы .NET обеспечивается строгое, однозначное соответствие между примитивными типами языков программирования и базовыми классами .NET. Большинством компиляторов для языков программирования, которые реализованы для платформы .NET, имеют встроенную поддержку примитивных типов.

Целям безопасности типизации также служит явное разделение на ссылочные типы и типы-значения, а также гибкий и надежный механизм преобразования типов-значений в ссылочные (известный также под названием boxing) и обратно (известный также под названием unboxing).

Библиография

1. Curry H.B., Feys R. Combinatory logic, vol.I, North Holland, Amsterdam, 1958
2. Hindley J.R., Seldin J.P. Introduction to combinators and λ-calculus. London Mathematical Society Student Texts, 1, Cambridge University Press, 1986
3. Milner R. A theory of type polymorphism in programming languages. Journal of Computer and System Science, 17(3):348-375, 1978
4. Hindley J.R. The principal type-scheme of an object in combinatory logic. Trans. Amer. Math. Soc., 146:29-60, 1969

Комментарий к слайду

К сожалению, в рамках времени, отведенных на одну лекцию, можно лишь в общих чертах охарактеризовать такую многоаспектную и гибкую теорию, как теория типов. Мы ограничились рассмотрением лишь наиболее существенных аспектов формализации типизации в языках программирования, включая краткое знакомство с приписыванием типа, выводимостью типов и системой типизации, реализованной в .NET.

Для более детального ознакомления с особенностями, достижениями и проблемами в области теории типов и типизации в языках программирования рекомендуется следующий список литературы:

1. Curry H.B., Feys R. Combinatory logic, vol.I, North Holland, Amsterdam, 1958

2. Hindley J.R., Seldin J.P. Introduction to combinators and λ-calculus. London Mathematical Society Student Texts, 1, Cambridge University Press, 1986

3. Milner R. A theory of type polymorphism in programming languages. Journal of Computer and System Science, 17(3):348-375, 1978

4. Hindley J.R. The principal type-scheme of an object in combinatory logic. Trans. Amer. Math. Soc., 146:29-60, 1969

Кратко остановимся на источниках. Работа [1] является энциклопедией формальной системы комбинаторной логики, которая формализует типизацию в языках программирования. Работа [2] связывает ламбда-исчисление и комбинаторную логику. Работа [3] описывает систему полиморфной типизации для языков программирования. Работа [4] посвящена вопросам моделирования типизации объектов языков программирования в терминах комбинаторной логики.