Теория типов и типизация в .NET

Содержание лекции

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

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

Типы в математике (1)

Чистой системой типов называется семейство лямбда-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»)

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).

Преимущества типизации

Модель предметной области лучше структурирована, существует иерархия сортов элементов

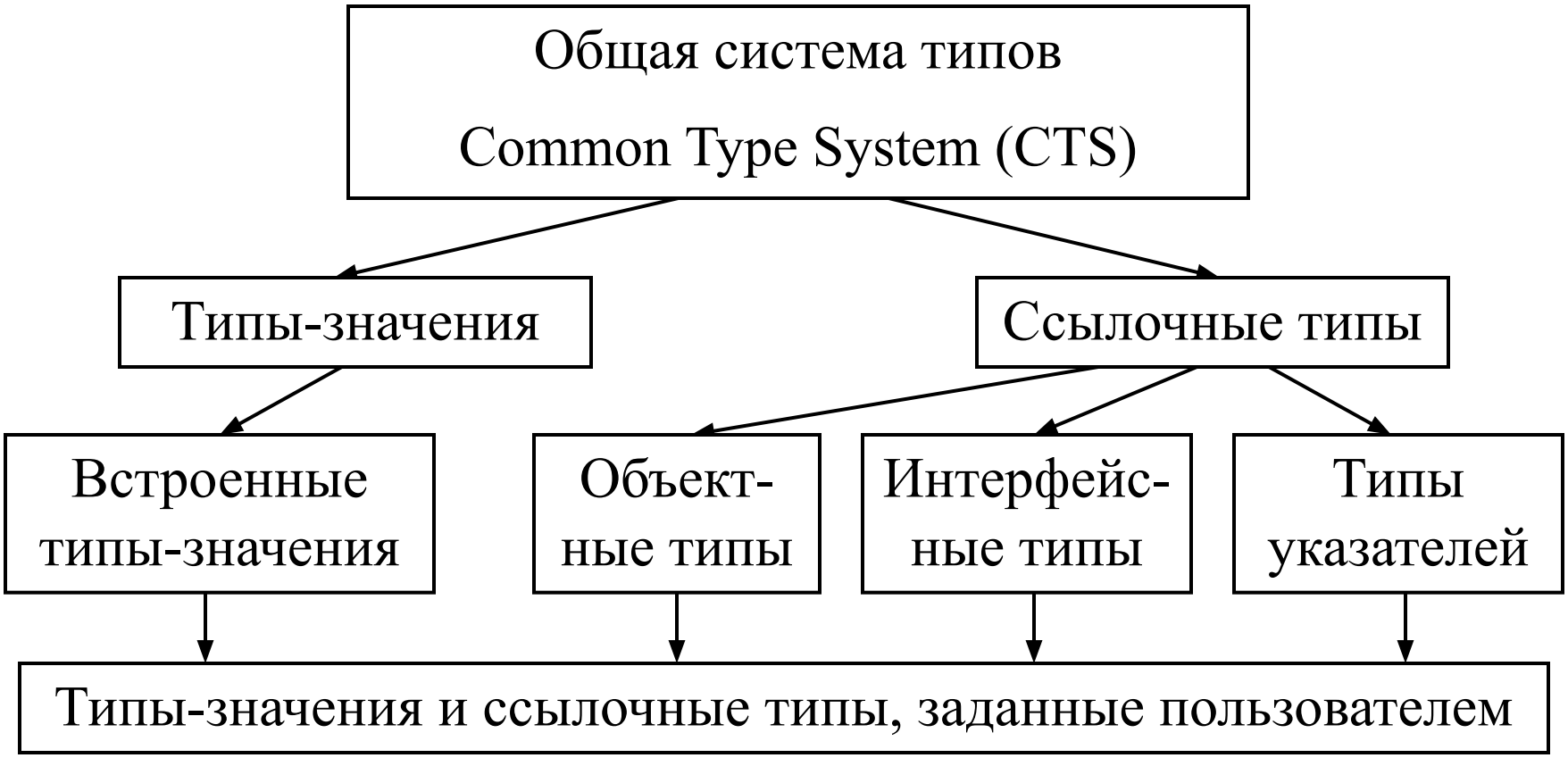
Манипулирование элементами более целенаправленно, разнородные элементы обрабатываются различным образом, однородные - единообразно

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

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

Иерархия типов в .NET



Управление типами в CTS

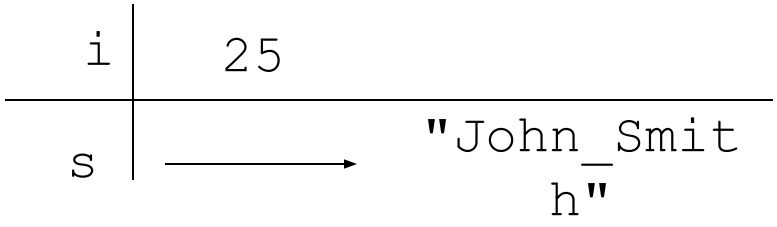
- Типы могут использоваться после инициализации (с учетом метода вызова, свойств get и set и т.д.)
- Над типами могут совершаться преобразования (как явным, так и неявным образом)
- Типы объединяются в совокупности (пространства имен, файлы, сборки)
- Важнейшими подкатегориями системы типизации в .NET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

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";
```



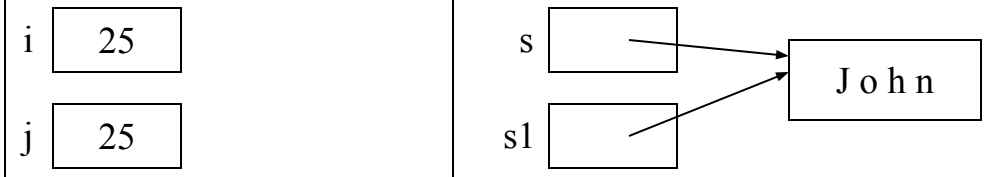
Ссылочные типы и типы-значения в .NET и C# (2)

- 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();`

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

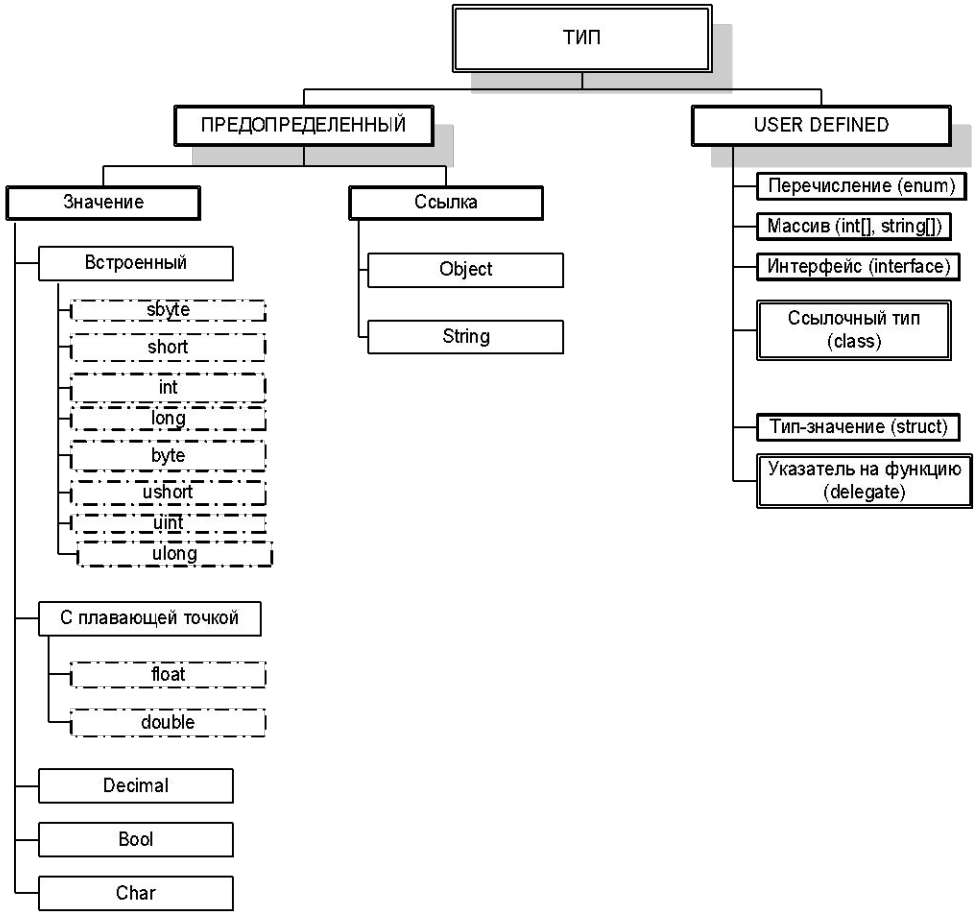
Пример `int i = 25; string s = “John"; int j = i; string s1 = s;`



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)

Иерархия типов языка C# (фрагмент)



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
```

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

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
    }
  }
}
```

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
```

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

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***)

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