Memory Leak

Consumo della memoria causato dalla mancata deallocazione di variabili/risorse non più utilizzati dai processi.

Un programma rimanendo attivo, continua a allocare memoria finchè la memoria del sistema non viene completamente consumata.

- Rallentamento delle funzionalità
- Problemi di memoria su altri programmi
- Riavvio del sistema

Stack e Managed Heap

Value Type

- Dichiarati all’interno di una funzione: Stack
- Parametri di una funzione : Stack
- All’interno di una classe: *Heap*

Reference Type

- Sempre: Heap

Garbage Collector

Il Garbage Collector è un componente il cui ruolo è di liberare la memoria dagli oggetti non più utilizzati.

Gestisce gli oggetti allocate nel managed heap.

Agisce quando si ha necessità di avere maggiori risorse a disposizione:un oggetto può essere rimosso in una fase successive rispetto al suo inutilizzo.

Garbage Collector

Il funzionamento del Garbage Collector è il seguente:

1. Segna tutta la memoria allocata (heap) come “garbage”
2. Cerca i blocchi di memoria in uso e li marca come validi
3. Dealloca le celle non utilizzate
4. Compatta il managed heap

Generations del Managed Heap

Per ottimizzare il funzionamento del managed heap, esso viene diviso in tre aree, dette generations.

Generation 0:

- È la prima area in cui vengono allocati gli oggetti
- Capacità minore (ordine della cache)

Collections

- Definite in **System.Collections**
- Definite in **System.Collections.Generics**
- Differenti specializzazioni per **differenti utilizzi**

Collections

INTERFACE	DESCRIPTION
`IEnumerable<T>`	The interface `IEnumerable` is required by the `foreach` statement. This interface defines the method `GetEnumerator`, which returns an enumerator that implements the `IEnumerator` interface.
`ICollection<T>`	`ICollection<T>` is implemented by generic collection classes. With this you can get the number of items in the collection (`Count` property), and copy the collection to an array (`CopyTo` method). You can also add and remove items from the collection (`Add`, `Remove`, `Clear`).
`IList<T>`	The `IList<T>` interface is for lists where elements can be accessed from their position. This interface defines an indexer, as well as ways to insert or remove items from specific positions (`Insert`, `RemoveAt` methods). `IList<T>` derives from `ICollection<T>`.
`ISet<T>`	This interface is implemented by sets. Sets allow combining different sets into a union, getting the intersection of two sets, and checking whether two sets overlap. `ISet<T>` derives from `ICollection<T>`.
`IDictionary<TKey, TValue>`	The interface `IDictionary<TKey, TValue>` is implemented by generic collection classes that have a key and a value. With this interface all the keys and values can be accessed, items can be accessed with an indexer of type `key`, and items can be added or removed.
`ILookup<TKey, TValue>`	Similar to the `IDictionary<TKey, TValue>` interface, lookups have keys and values. However, with lookups the collection can contain multiple values with one key.
`IComparer<T>`	The interface `IComparer<T>` is implemented by a comparer and used to sort elements inside a collection with the `Compare` method.
`IEqualityComparer<T>`	`IEqualityComparer<T>` is implemented by a comparer that can be used for keys in a dictionary. With this interface the objects can be compared for equality.

Collections

COLLECTION	ADD	INSERT	REMOVE	ITEM	SORT	FIND
`List<T>`	O(1) or O(n) if the collection must be resized	O(n)	O(n)	O(1)	O (n log n), worst case O(n ^ 2)	O(n)
`Stack<T>`	`Push`, O(1), or O(n) if the stack must be resized	n/a	`Pop`, O(1)	n/a	n/a	n/a
`Queue<T>`	`Enqueue`, O(1), or O(n) if the queue must be resized	n/a	`Dequeue`, O(1)	n/a	n/a	n/a
`HashSet<T>`	O(1) or O(n) if the set must be resized	`Add` O(1) or O(n)	O(1)	n/a	n/a	n/a
`SortedSet<T>`	O(1) or O(n) if the set must be resized	`Add` O(1) or O(n)	O(1)	n/a	n/a	n/a
`LinkedList<T>`	`AddLast` O(1)	`Add After` O(1)	O(1)	n/a	n/a	O(n)
`Dictionary <TKey, TValue>`	O(1) or O(n)	n/a	O(1)	O(1)	n/a	n/a
`SortedDictionary <TKey, TValue>`	O(log n)	n/a	O(log n)	O(log n)	n/a	n/a
`SortedList <TKey, TValue>`	O(n) for unsorted data, O(log n) for end of list, O(n) if resize is needed	n/a	O(n)	O(log n) to read/ write, O(log n) if the key is in the list, O(n) if the key is not in the list	n/a	n/a

icubed

System.**Collection**

Contiene le seguenti Collections:

- ArrayList
- Stack
- Queue
- HashTable

Supportano solo tipizzazione debole

Demo

Collections

System.Collections.**Generic**

Contiene le seguenti Collections:

- List<T>
- Dictionary<Tkey, Tvalue>
- HashSet<T>
- Stack<T>
- Queue<T>
- …

Supportano Tipizzazione forte

Generics

- Consentono di scrivere classi e metodi **indipendenti dal tipo**
- Anzichè scrivere metodi differenti per tipi differenti, è possibile scrivere **un unico metodo**
- Anche per le **classi**, **metodi** ed **interfacce**

Generics

- Performance
  - Possiamo utilizzare **System.Collection** e **System.Collections.Generic**
  - Utilizzando i tipi a valore con classi non generiche sono necessarie operazioni di **boxing** e **unboxing**. Il tipo a valore è convertito in un tipo a referenza e viceversa
  - Utilizzando List<T> con il tipo int viene effettuata la **traduzione solo in fase di compilazione**. Non vengono eseguite operazioni boxing / unboxing

Generics

Type safety

- Se fossero utilizzati degli object come parametri potremmo trovarci in **situazioni non sicure** in termini di esecuzione
- Possiamo aggiungere stringhe, interi, ecc… **senza generare errori** in compilazione
- Con i Generics il compilatore **si accorge del tipo** che stiamo inserendo

Generics

- **Riuso** del codice
- Meno codice da **scrivere**
- Meno codice da **mantenere**!
- Scriviamo un metodo/classe che può essere utilizzato con **tipi differenti**

Generics

Crescita del codice

- Quanto viene compilata una Generics Class il compilatore crea una classe **per ogni tipo che deve gestire**
- Una classe Generics viene definita direttamente all'interno dell'assembly, istanziare una classe Generics con tipi specifici **non duplica** la classe nell'IL

Generics

Creare una classe generica

- Non è possibile assegnare null ad una classe generic

```
public class LinkedListNode<T>
    {
        public LinkedListNode(T value)
        {
            Value = value;
        }

        public T value { get; private set; }
        public LinkedListNode<T> Next { get; internal set; }
        public LinkedListNode<T> Prev { get; internal set; }
    }
```

Utilizzo della keyword **T**

Generics

Non possibile associare un **valore null** ad un Generics perché:

- Un tipo generics può essere istanziato come Value Types ed i Value Types non accettano valori null
- Un tipo generics **non è** un Reference Types!

Utilizzando **default(T)**

- Viene associato **null** a **Reference Types**
- Viene associato **0** a **Values Types**

Generics

E’ possibile definire **alcune regole** relativamente al tipo di Generics che deve essere utilizzato:

CONSTRAINT	DESCRIPTION
`where T: struct`	With a struct constraint, type `T` must be a value type.
`where T: class`	The class constraint indicates that type `T` must be a reference type.
`where T: IFoo`	Specifies that type `T` is required to implement interface `IFoo`.
`where T: Foo`	Specifies that type `T` is required to derive from base class `Foo`.
`where T: new()`	A constructor constraint; specifies that type `T` must have a default constructor.
`where T1: T2`	With constraints it is also possible to specify that type `T1` derives from a generic type `T2`.

Generics

E’ possibile **combinare** multipli constraint:

```
public class MyClass<T>
  where T : IFoo, new()
{
  //...
```

E’ possibile utilizzare **l’ereditarietà**.
Un tipo Generics può **implementare un’interfaccia Generics**:

```
public class LinkedList<T> : IEnumerable<T>
{
  //...
```

Generics

Metodi Generici

- Come per le classi è possibile **definire metodi Generics**
- Anche in questo caso è possibile **aggiungere Costraints**

```
public static decimal Accumulate<TAccount>(IEnumerable<TAccount> source)
  where TAccount : IAccount
```

Grazie!

Ricordate il feedback!

© 2021 iCubed Srl

iCubed s.r.l.

Piazza Durante, 8 20131 MILANO

Phone: +39 02 57501057

P.IVA 07284390965