LINQ

Language Integrated Query

LINQ

- LINQ (Language Integrated Query) is a set of classes and methods that enable you to access data that is stored in a variety of places and formats. The LINQ framework is the standard for accessing data in managed languages.

- A query is an expression that retrieves data from a data source. Different data sources have different native query languages, for example SQL for relational databases and XQuery for XML. Developers must learn a new query language for each type of data source or data format that they must support. LINQ simplifies this situation by offering a consistent C# language model for kinds of data sources and formats. In a LINQ query, you always work with C# objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, .NET collections, and any other format when a LINQ provider is available.

Three Parts of a Query Operation

All LINQ query operations consist of three distinct actions:

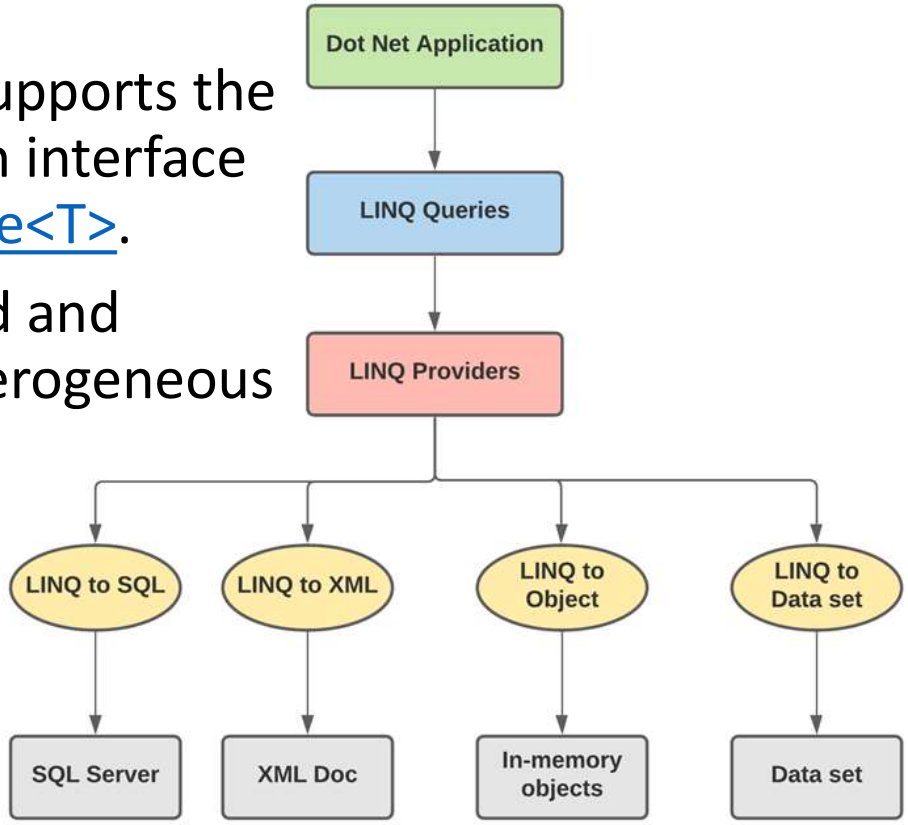
1. Obtain the data source.
2. Create the query.
3. Execute the query

The Data Source

A LINQ data source is any object that supports the generic IEnumerable<T> interface, or an interface that inherits from it, typically IQueryable<T>.

LINQ presents to programmers a unified and consistent API for data access from heterogeneous data sources, such as:

- In-memory object graphs
- Active Directory entries
- Flickr pictures and XML
- SQL Server



The Query

The query specifies what information to retrieve from the data source or sources. Optionally, a query also specifies how that information should be sorted, grouped, and shaped before being returned. A query is stored in a query variable and initialized with a query expression.

Query Execution

- **Deferred Execution**

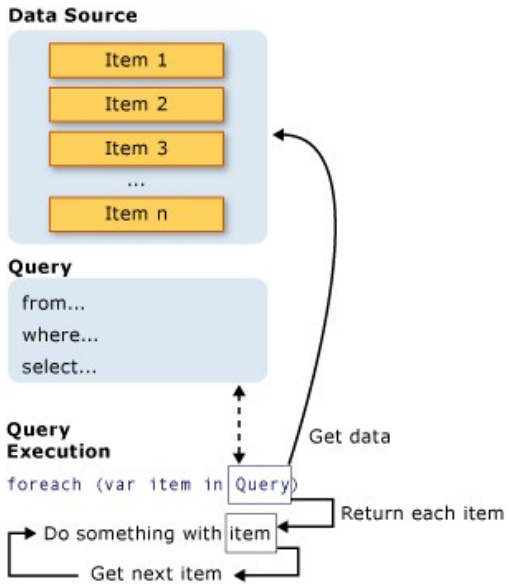
The query variable itself only stores the query commands. The actual execution of the query is deferred until you iterate over the query variable in a foreach statement.

- **Forcing Immediate Execution**
- To force immediate execution of any query and cache its results, you can call the ToList or ToArray methods.

you can also force execution by putting the foreach loop immediately after the query expression.

The complete query operation

- In LINQ, the execution of the query is distinct from the query itself. In other words, you don't retrieve any data by creating a query variable.



LINQ Operators

- The LINQ Operators are nothing but a set of extension methods used to write the LINQ Query.
- These LINQ extension methods provide many useful features we can apply to the data source. Some of the features are filtering the data, sorting the data, grouping the data, etc.

Projection Operators

- **Select**: Projects each element of a sequence into a new form.

- **SelectMany**: Projects each sequence element to an IEnumerable<T> and flattens the resulting sequences into one sequence.

Filtering Operators

- **Where**: Filters a sequence of values based on a predicate.

- **OfType**: Filters the elements of an array based on a specified type.

Partitioning Operators

- **Take:** Returns a specified number of contiguous elements from the start of a sequence.
- **Skip:** Bypasses a specified number of elements in a sequence and then returns the remaining elements.
- **TakeWhile:** Returns elements from a sequence as long as a specified condition is true.
- **SkipWhile:** Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

Ordering Operators

- **OrderBy:** Sorts the elements of a sequence in ascending order according to a key.
- **OrderByDescending:** Sorts the elements of a sequence in descending order according to a key.
- **ThenBy:** Performs a subsequent ordering of the elements in a sequence in ascending order.
- **ThenByDescending:** Performs a subsequent ordering of the elements in a sequence in descending order.
- **Reverse:** Inverts the order of the elements in a sequence.

Grouping Operators

- **GroupBy:** Groups the elements of a sequence according to a specified key selector function.

Set Operators

- **Distinct:** Removes duplicate elements from a sequence.
- **Union:** Produces the set union of two sequences.
- **Intersect:** Produces the set intersection of two sequences.
- **Except:** Produces the set difference of two sequences.
- **Concat :**concatenate two sequences into one sequence of the same type.

Join Operators

- **Join:** Joins two sequences based on matching keys.
- **GroupJoin:** Groups elements from a sequence based on a key and joins them with elements from another sequence.

Conversion Operators

- **AsEnumerable:** Casts an IEnumerable to an IEnumerable<T>.
- **ToArray:** Converts a sequence to an array.
- **ToList:** Converts a sequence to a List<T>.
- **ToDictionary:** Converts a sequence to a Dictionary<TKey, TValue> based on a key selector function.

Element Operators

- **First:** Returns the first element of a sequence.
- **FirstOrDefault:** Returns the first element of a sequence or a default value if no element is found.
- **Last:** Returns the last element of a sequence.
- **LastOrDefault:** Returns the last element of a sequence or a default value if no element is found.
- **Single:** Returns the only element of a sequence and throws an exception if there is not exactly one element in the sequence.
- **SingleOrDefault:** Returns the only element of a sequence or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
- **ElementAt:** Returns the element at a specified index in a sequence.
- **ElementAtOrDefault:** Returns the element at a specified index in a sequence or a default value if the index is out of range.

Quantifier Operators

- **Any:** Determines whether any element of a sequence satisfies a condition.
- **All:** Determines whether all elements of a sequence satisfy a condition.
- **Contains:** Determines whether a sequence contains a specified element.

Aggregate Operators

- **Count:** Counts the elements in a sequence.
- **LongCount:** Counts the elements in a sequence, returning the count as a long.
- **Sum:** Computes the sum of a sequence of numeric values.
- **Min:** Returns the minimum value in a sequence.
- **Max:** Returns the maximum value in a sequence.
- **Average:** Computes the average of a sequence of numeric values.
- **Aggregate:** Applies an accumulator function over a sequence.

Equality Operators

- **SequenceEqual:** Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.

Generation Operators

- **Empty:** Returns an empty IEnumerable<T> with the specified type argument.
- **Repeat:** Generates a sequence that contains one repeated value.
- **Range:** Generates a sequence of integral numbers within a specified range.

Special Operators

- **DefaultIfEmpty Operators:** This operator returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.