LINQ

Language-INtegrated Query

Agenda

- What is LINQ?
- LINQ to Objects

What is LINQ?

- LINQ is a uniform programming model for any kind of data
  - And enables you to query and manipulate data with a consistent model that is independent from data sources
- LINQ defines a set of method names (called standard query operators, or standard sequence operators), along with translation rules from so-called query expressions to expressions using these method names, lambda expressions and anonymous types
- LINQ query:

```
var adultNames = from person in people
                 where person.Age >= 18
                 select person.Name;
```

Why Use LINQ?

- The two most common sources of non-OO information are relational databases and XML
- Rather than add relational or XML-specific features to our programming languages and runtime, with the LINQ project Microsoft have taken a more general approach and added general-purpose query facilities to the .NET Framework that apply to all sources of information
- Language-integrated query (LINQ) allows *query expressions* to benefit from:
  - rich metadata
  - compile-time syntax checking
  - static typing
  - IntelliSense

  that was previously available only to imperative code

Foundation

- Features in the language that is necessary to enable LINQ:
  - Generics
  - Anonymous methods
  - Implicit typing of local variables
  - Lambda expressions and expression trees
  - Extension methods

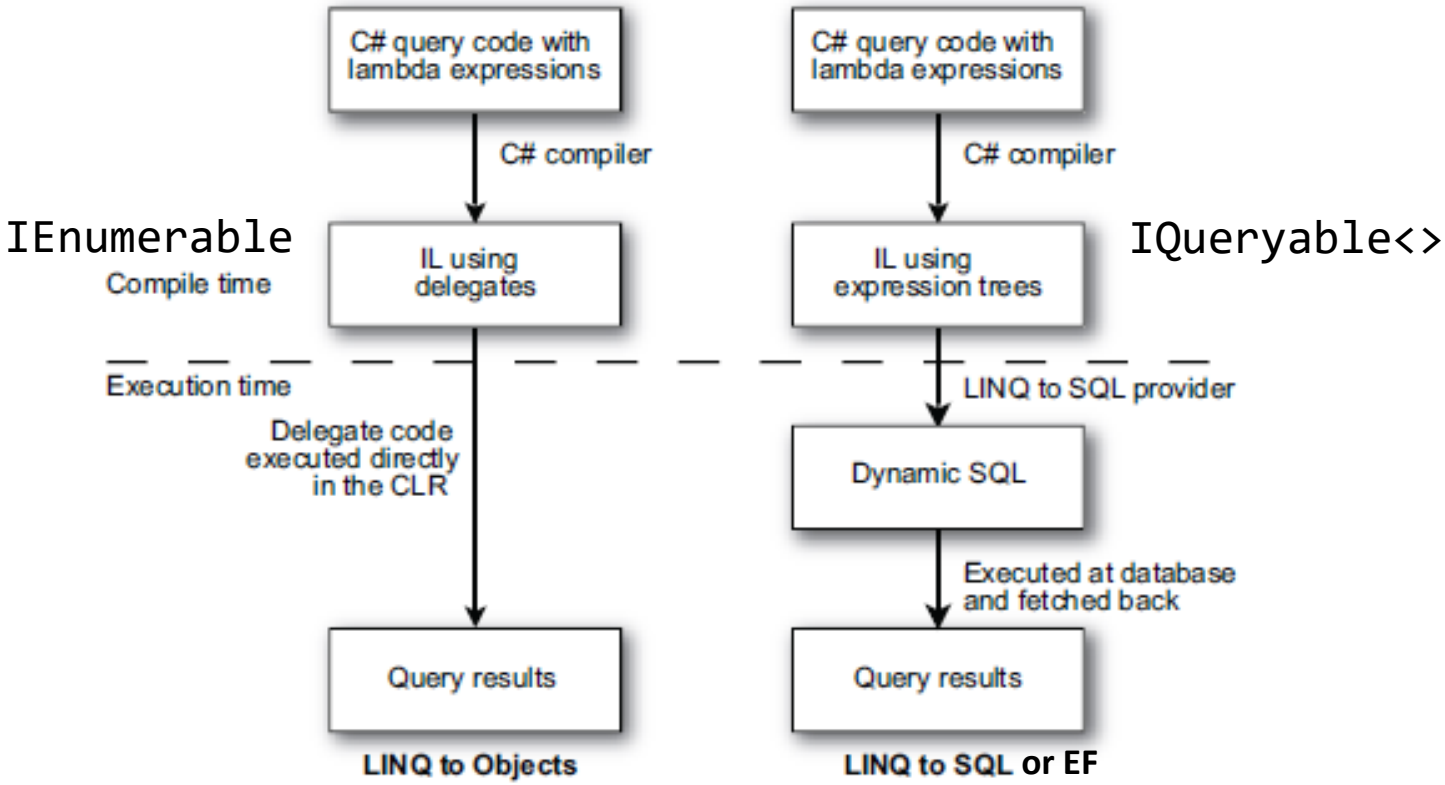
Expression Trees

- Expression trees represent code in a tree-like data structure, where each node is an expression
- When a lambda expression is assigned to a variable of type
  `Expression<TDelegate>`
  the compiler emits code to build an expression tree that represents the lambda expression
- The C# compiler can only generate expression trees from expression lambdas (single-line lambdas)
- Example:

```
Expression<Func<int, bool>> myLambda = num => num < 5;
```

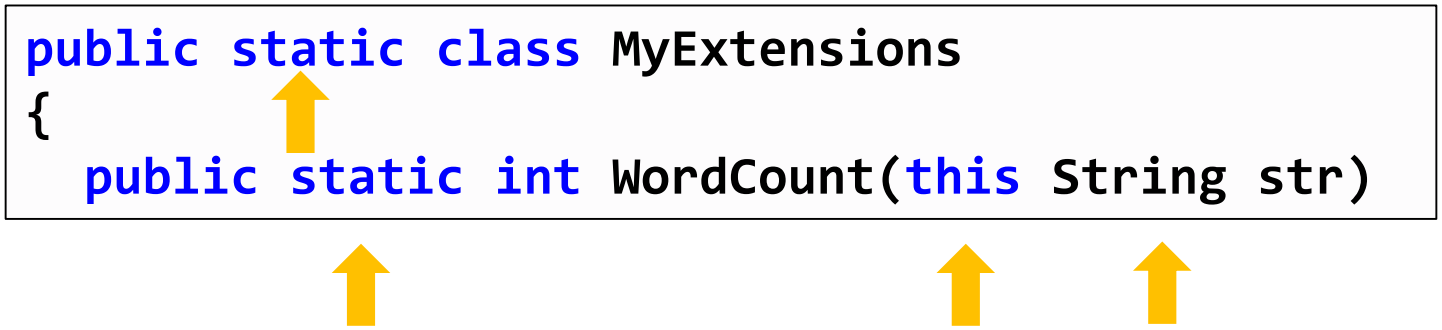
The Use of Expression Trees

- Both LINQ to Objects and LINQ to EF start with C# code and end with query results
- The ability to execute the code remotely as LINQ to EF does comes through expression trees



Extension Methods

- Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.
- Extension methods are a special kind of **static** method, but they are called as if they were instance methods on the extended type.
- Their first parameter specifies which type the method operates on, and the parameter is preceded by the **this** modifier.
- Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.



Using Extension Methods

- For client code written in C# there is no apparent difference between calling an extension method and the methods that are actually defined in a type.
- Extension methods are defined as static methods but are called by using instance method syntax.

```
string s = "Hello Extension Methods";
int i = s.WordCount();
```

- To enable extension methods for a particular type, just add a using directive for the namespace in which the methods are defined.

LINQ Foundation

- LINQ works on a sequence of elements
- A sequence is an instance of a class that implements the **IEnumerable<T>** interface (local query)
  or
  IQueryable<T> interface (remote query)

- A LINQ **query operator**, work on an input sequence and produce some output value
  - This output value could be an output sequence or single scalar value

- LINQ query operators that are implemented as extension methods in the static **System.Linq.Enumerable** class

LINQ TO OBJECTS

LINQ

- Defines a set of general purpose ***standard query operators*** that allow:
  - traversal,
  - filter, and
  - projection

  operations to be expressed in a direct yet declarative way in any .NET-based programming language.
- The standard query operators are defined as extension methods in the type **System.Linq.Enumerable**.
- Almost all standard query operators are defined in terms of the **IEnumerable<T>** interface.
- This means that every **IEnumerable<T>**-compatible information source gets the standard query operators simply by adding the following using statement in C#:

```
using System.Linq;
```

LINQ

- The developer is free to use:
  - named methods,
  - anonymous methods, or
  - lambda expressions

  with query operators.
- Lambda expressions have the advantage of providing the most direct and compact syntax for authoring.
- But more importantly:
  - lambda expressions can be compiled as either code or data, which allows lambda expressions to be processed at runtime by optimizers, translators, and evaluators.

LINQ Example

```
string[] names = { "Burke", "Connor", "Frank","Everett", "Albert", "George"};

var query = from s in names
            where s.Length == 5
            orderby s
            select s.ToUpper();

foreach (string item in query)
    Console.WriteLine(item);

// Is equivalent to
IEnumerable<string> query2 = names
                            .Where(s => s.Length == 5)
                            .OrderBy(s => s)
                            .Select(s => s.ToUpper());

foreach (string item in query2)
    Console.WriteLine(item);
```

Query Syntax: from

- Every query expression starts off in the same way - stating the source of a sequence of data:

```
from element in source
```

  - The ***element*** part is just an identifier
  - The ***source*** part is just a normal expression.

Query Syntax: select

- Query expressions always end with either a select clause or a group clause:

```
select expression
```

- The select clause is known as a ***projection***.

- Minimal (useless) query:

```
var query = from name in names
            select name;
```

Query Syntax: OrderBy

- The **OrderBy** and **OrderByDescending** operators can be applied to any information source and allow the user to provide a key extraction function that produces the value that is used to sort the results.

```
var s1 = names.OrderBy(s => s);
var s2 = names.OrderByDescending(s => s);
```

- OrderBy and OrderByDescending also accept an optional comparison function that can be used to impose a partial order over the keys.

```
var s3 = names.OrderBy(s => s.Length);
var s4 = names.OrderByDescending(s => s.Length);
```

Query Syntax: ThenBy

- To allow multiple sort criteria, both **OrderBy** and **OrderByDescending** return **OrderedSequence<T>** rather than the generic **IEnumerable<T>**.
- Two operators are defined only on **OrderedSequence<T>**, namely **ThenBy** and **ThenByDescending** which apply an additional (subordinate) sort criterion.

```
var s1 = names.OrderBy(s => s.Length).ThenBy(s => s);
```

Query Syntax: reverse

- **Reverse** simply enumerates over a sequence and yields the same values in reverse order.
- Unlike **OrderBy**, **Reverse** doesn't consider the actual values themselves in determining the order, rather it relies solely on the order the values are produced by the underlying source.

Query Syntax: …

Self study

LINQ Extensibility

- LINQ allows third parties to augment the set of standard query operators with new domain-specific operators that are appropriate for the target domain or technology.
  - LINQ to SQL
  - LINQ to Entities
  - LINQ to XML

  - LINQ to Google
  - LINQ to CSV
  - LINQ to Twitter
  - LINQ to …

References & Links

- http://linqsamples.com/
- Free book: **LINQ Succintly**
- LINQ: .NET Language-Integrated Query (Don Box, Anders Hejlsberg)
  http://msdn.microsoft.com/en-us/library/bb308959.aspx
- **.NET Framework Developer Center > Learn > LINQ**
  http://msdn.microsoft.com/en-us/netframework/aa904594.aspx
- **101 LINQ Samples**
  http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b
- C# in Depth
  http://csharpindepth.com/