LINQ Querys

SQL:

IF EXISTS (SELECT 1 FROM tblUsers WHERE Username = „whatever“)
// Return 1, wenn Username vorhanen ist, 0, wenn eben nicht

LINQ:
[crayon-679184badd0df380866198/]

My SQL Querys

eine Sammlung netter Querys für MySQL:

ifnull(expression,“) – Wenn expression NULL ist, dann zählt der rechte Wert.

Concat(‚a‘,123,(SELECT col from tab WHERE a = b)) – erstellt einen zusammengesetzten String aus allen teilen der Funktion

---

Eigenen Event erstellen

Ein Event oder deutsch Ereignis ist immer an einen Delegatentypen gebunden, welcher wiederum eine bestimmte Signatur von (Rückgabewert + Parameter).

Ungefähr, stellt es dieses Muster da:

Zuerst brauchen wir einen delegaten. Jetzt stellt sich die Frage, die Methoden, auf die der Delegat später verweist, wie soll die aussehen? soll sie einen Rückgabewert haben? Soll irgend ein Wert an die Methode übergeben werden? Ja? Dann brauchen wir Parameter. Um das noch ein wenig verständlicher zu machen, wenn wir eine for-Schleife haben und wollen, dass das Event nach jedem Schleifendurchgang immer abgefeuert wird, und uns den Wert liefert. Dann würden wir einen Delegaten mit einem Parameter vom Typ int z.B. machen.
[crayon-679184badd52a793738072/]

Dann braucht man das Event an sich. Dieser muss vom Typ des Delegaten sein.
[crayon-679184badd52e598886333/]

Jetzt muss gesagt werden, wann das Event abgefeuert werden muss. Und, wichtig, wir haben dem Delegaten einen Parameter (int a) angegeben. Dieser Parameter muss nun durch das Event gefüllt werden. Das machen wir, indem wir
[crayon-679184badd530694296830/]
Jetzt wird bei jedem Schleifendurchgang das Event abgefeuert.

Ok, das war unsere Quell klasse.

Jetzt erstellen wir ein Objekt dieser Klasse und weisen dem Event eine Methode zu, die ausgeführt werden soll, sobald das Event abgefeuert wird. Die Methode muss die selbe Signatur haben wie unser Delegat. Zum Schluss müssen wir die Methode in der Quell klasse ausführen und das Event wird abgefeuert und führt ebenso die Methode mit dem Parameter aus, die wir in der Ziel Methode eingegeben haben.
[crayon-679184badd531119152580/]
[crayon-679184badd532517898764/]

Download Beispielprojekt Eigener Event

Delegaten in .NET

Das .Net Framework beinhaltet bereits Delegaten, welche bestimmte Signaturen haben.

1. Der EventHandler

stellt einen Delegaten dar, der keinen Rückgabewert hat (void), und 2 Parameter hat. 1. Ist der object sender , welcher das Senderobject implementiert und EventArgs[] welcher evtl. Argumente beinhalten kann. Diesen Delegaten findet man, wenn man zu einem Event von Steuerelementen eine Methode aufruft.

2. Action<>

Der Action Delegat besitzt keinen Rückgabewert, wohl aber bis zu 16 Parameterelemente. Dieser Delegat kann z.B. dazu verwendet werden um eine Methode mit der Task Klasse aufzurufen und diese im neuen Thread zu starten.
[crayon-679184badd6f9089358488/]
Das geht natürlich auch mit anonymen Methoden
[crayon-679184badd6fd657961750/]
3. Func<>

Der Func Delegat ist dem Action<> Delegaten gleich, liefert im Gegensatz zu ihm aber einen Rückgabewert

---

Stopwatch – Messung von Zeit von bestimmten Algorithmen oder Operationen

Manchmal möchte man wissen, wie lange mein Computer braucht um einen bestimmten Code auszuführen. Hierzu könnte man natürlich DateTime.Now für die Operationen verwenden. Allerdings ist dieser Wert ungenau. Einen Genaueren Wert erhalten wir mit dem Stopwatch. Dieser funktioniert aus recht simpel:
[crayon-679184badd82d793833916/]
[crayon-679184badd830398908938/]

C# Vergleichsoperatoren ? : und ??

Oft findet man im Code ewig lange if – else Verzweigungen, die zuviel Platz einnehmen und irgendwann die Übersicht rauben.

Die beiden Operatoren ? : und ?? sehen merkwürdig aus, sind aber in der Handhabung sehr praktikabel.

Der ? : prüft ob der Wert true ist. Falls ja, zählt der linke Wert, wenn nicht, zählt der Wert rechts davon.
[crayon-679184badd98a097725777/]
Der ?? Operator prüft, ob der Wert null ist. Bei nein, zählt der linke Wert, bei ja der rechte.
[crayon-679184badd98e029026344/]

Das ganze in lang könnte so aussehen:

[crayon-679184badd98f728136268/]
[crayon-679184badd990011170886/]
Ab C# 6 soll noch das Feature hinzufügen, dass auch das „Elternobjekt“ geprüft werden kann, ob dieser nicht null ist.
[crayon-679184badd992457541000/]