**C# پیشرفته :**

در این سند نحوه شروع کار با تمرینات پیشرفته C# را شرح خواهیم داد.

سری LINQ :

Language Integrated Query ، LINQ، یک نحو منسجم برای پرس و جو از هر منبع داده ای را فراهم می کند. بیشتر از آن، کامپایلر و کتابخانه‌ها آن نحو سازگار را به پرس‌و‌جوهای بومی برای هر منبع داده متفاوت ترجمه می‌کنند. شما LINQ را یاد می گیرید و می توانید به طور موثر به داده ها از پایگاه داده ها، ذخیره سازی ابری، اشیاء موجود در حافظه، منابع داده JSON یا XML دسترسی داشته باشید. در این مجموعه، نحوه نوشتن پرس و جوهای LINQ را یاد خواهید گرفت. همچنین با مکانیسم‌هایی آشنا می‌شوید که LINQ برای ترجمه کد C# شما به پرس‌و‌جوهای بومی برای منابع داده خاص استفاده می‌کند.

سری Async :

برنامه های مدرن نیاز به پردازش ناهمزمان دارند. می توانید با سرویس ها در فضای ابری یا ماشین های دیگر تماس شبکه برقرار کنید. ممکن است فرآیندهای طولانی مدت را برای تجزیه و تحلیل مجموعه داده های بزرگ شروع کنید. زبان C# دارای ویژگی هایی است که استدلال و درک برنامه های همگام را آسان تر می کند. در این مجموعه، شما با async و wiat و کلاس Task آشنا خواهید شد. شما یاد خواهید گرفت که چگونه این ویژگی ها با هم کار می کنند تا به شما در نوشتن برنامه های ناهمزمان کمک کنند.

این داکیومنت شامل 8 قسمت می باشد که همراه با نمونه کد آن را بررسی می نماییم.

1. مقدمه ای بر (LINQ) :

LINQ مخفف Language-Integrated Query به معنای زبان پرس و جوی یکپارچه است که در دات نت 3.5 معرفی شد و به برنامه نویس اجازه می دهد که داده ها را از هر نوع منبع داده ای بدون نیاز به دانستن یک زبان دیگر، پرس و جو کند. پرس و جو، فرایند به دست آوردن داده از منبع داده است. LINQ پرس و جوی داده ای از منابع داده ای مختلف را بسیار راحت کرده است.

در این آموزش مرحله به مرحله با اصول LINQ آشنا می شوید.

نگاهی کلی بر عملگرهای LINQ :

* عملگرهای محدود کننده کلمه کلیدی where (Restriction operators) : این عملگرها توالی ورودی را برای تولید یک دنباله خروجی محدود یا فیلتر می کنند.
* عملگرهای انتخاب کلمه کلیدی select(Projection operators) : این عملگرها عناصر توالی خروجی را از عناصر ورودی انتخاب می کنند. این عناصر خروجی می توانند یکسان و یا انواع مختلف باشند.
* عملگرهای پارتیشن بندی (Partition operators) : از متدهای Take، Skip، TakeWhile و SkipWhile برای پارتیشن بندی دنباله ورودی می توانید استفاده کنید. می توانید قسمتی از دنباله ورودی را به عنوان دنباله خروجی دریافت کنید.
* عملگرهای مرتب سازی (Ordering operators) : کلمات کلیدی orderby ( به تنهایی برای مرتب سازی نزولی استفاده می شود) و کلمات کلیدی OrderBy, ThenBy, OrderbyDescending,ThenByDescending برای مرتب سازی خروجی داده ها استفاده می شود.
* عملگرهای گروه بندی (Grouping operators) : عملگرهای GroupBy و into یک دنباله را دسته بندی می کنند.
* عملگرهای مجموعه (Set Operators) : عملکردی را برای مقایسه چندین مجموعه داده ارائه می کنند. می توانید تقاطع، اتحاد، همه عناصر متمایز و تفاوت بین مجموعه ها را پیدا کنید. مانند کلمه کلیدی distinct
* عملگرهای تبدیل (Conversion operators) : اگر بخواهید یک مجموعه از نتیجه پرس و جو را به مجموعه دیگری تبدیل کنید از این عملگرها استفاده کنید.
* عملگرهای عنصر (Element operators) : در صورتی که بخواهید عناصر خروجی را بر اساس موقعیت آن عنصر در دنباله دریافت کنید می توانید از این عملگرها استفاده کنید. مانند متد های First, FirstOrDefault, Last, LastOrDefault, ElementAt
* ایجاد توالی (Generate sequences) : متدهای Range و Repeat دنباله هایی از اعداد صحیح را ایجاد می کنند. این ها می توانند دنباله های منبع پرس و جوها باشند.
* تعیین کمیت اعضا (Quantifying members) : متدهای Any و All برای عناصری که با یک شرط مطابقت دارند استفاده می شوند.
* عملگرهای جمع کننده (Aggregator operators) : تعدادی متد وجود دارد که محاسبات مقادیر را در یک دنباله انجام می دهد. مانند متد count و sum و... توجه داشته باشید که برخی از این روش ها مستلزم این هستند که دنباله ورودی از نوع عددی باشد.
* عملگرهای توالی (Sequence operations) : این متدها به جای عناصر یک دنباله، بر روی کل دنباله ها عمل می کنند. آنها توالی ها را مقایسه می کنند یا از ترکیب همه عناصر دنباله های جدیدی ایجاد می کنند. مانند متدهای SequenceEqual, Concat, Combine
* عملیات پیوند (Join operations) : این عملگرها، عملکردهای مشابه عملگرهای SQL Join را انجام می دهند. این ها با هر منبع داده LINQ کار می کنند. مانند متد join

1. فراگیری دستورات LINQ :

در این قسمت تمامی عملگرهای قید شده در قسمت مقدمه به صورت عملی در پروژه CsharpAdvanced اجرا می شود.

1. ترکیب و اجرای پرس و جوی در LINQ :

در این قسمت نیز، ادامه مبحث عملگرهای قید شده در قسمت مقدمه را به صورت عملی در پروژه CsharpAdvanced پیاده سازی کرده ایم.

1. Entity Framework و اتصال به منبع داده از راه دور در LINQ :

در این قسمت نحوه کار LINQ با Entity Framework را برای جستجو در منابع داده SQL می آموزیم.

* ایجاد پروژه
* نصب پکیج Microsoft.EntityFrameworkCore
* نصب پکیج Microsoft.EntityFrameworkCore.SqlServer
* نصب پکیج Microsoft.EntityFrameworkCore.Tools
* ساخت یک فولدر برای مدل های پروژه به اسم Models
* ساخت کلاس های مورد نیاز و تعریف پارامترها .
* ساخت کلاس EFContext که ارث بری شده از کلاس DBContext (این کلاس دارای متدهای تعریف کانکشن و و کوئری هاست) برای نحوه ساختار این کلاس به پروژه معرفی شده مراجعه کنید. ( connectionString به درستی تعریف شود.)
* اجرا دستور Add-Migration NewDatabaseدر قسمت package console manager .
* بعد از اجرا دستور بالا پوشه Migrations ایجاد می شود.
* مرحله بعدی ایجاد پایگاه داده با استفاده از Migrations مرحله قبل است.
* مجدد در قسمت package console manager دستور update-database –verbose اجرا می کنیم. دیتابیس ایجاد می شود.
* برای عملیات CRUD باید ابتدا یک instance از کلاس DBContext ایجاد کنیم.
* برای درج ردیف در پایگاه داده ایجاد شده و جدول مشخص باید از متد SaveChanges استفاده نماییم.
* برای حذف یک ردیف از پایگاه داده ایجاد شده و جدول مشخص باید از متد Remove و سپس متد SaveChanges استفاده نماییم.
* برای ویرایش یک ردیف در پایگاه داده ایجاد شده و جدول مشخص باید از متد Find و متد SaveChanges استفاده نماییم.
* در EF Core برای Select از دیتا در پایگاه داده ایجاد شده و جدول کافیست از data مورد نظر یک لیست ایجاد نماییم. ولی با استفاده از LINQ میتوانیم مستقیم Select مشابه دستورات SQL را اجرا کنیم.
* تمامی مثال های عملیات CRUD در کلاس Program ، پروژه معرفی شده قابل مشاهده است.

در پروژه نمونه CsharpAdvanced تمامی این عملگرهای قید شده در این سند را با فایل های کلاسی هم نام با نوع عملگر و کلمه کلیدی می توانید مشاهده کنید.

<https://github.com/BaharehSmaili/CsharpAdvanced>

1. مقدمه ای بر Async، Await و Tasks :

در این بخش، نحوه استفاده از async و await و task در برنامه نویسی غیرهمزمان در زبان C# با کمک چند مثال ساده بررسی می کنیم.

C# با استفاده از برخی توابع اصلی، کلاس ها و کلمات کلیدی رزرو شده از برنامه نویسی غیرهمزمان (asynchronous programming) پشتیبانی می کنند.

وقتی ما با Ui سروکار داریم و یک متد که زمان اجرای آن طولانی است (مثل خواندن یک فایل بزرگ و ذخیره آن در پایگاه داده) را در رویداد کلیک یک دکمه می گذاریم زمانی که روی آن دکمه کلیک شود رابط کاربری اپلیکیشن قفل شده و به اصطلاح هنگ می کند ، زیرا رابط کاربری و بقیه متد ها در برنامه نویسی همگام (Synchronous) در یک نخ (Thread) از CPU اجرا می شوند پس رابط کاربری تا زمانی که فعالیت متد پایان نیابد پاسخی به کاربر نمی دهد. برنامه نویسی ناهمگام در این شرایط بسیار کار آمد است ، زیرا در این روش رابط کاربری و متد ها به هم متکی نبوده و متد ها به صورت جداگانه اجرا می شوند.

به طور کلی با استفاده از برنامه نویسی ناهمگام می توانید میزان واکنش گرایی و سرعت نرم افزار های خود را به میزان قابل توجهی افزایش دهید. تکنیک های برنامه نویسی همگام و سنتی باعث می شود عملکرد نرم افزار پیچیده تر شود و این موضوع اشکال زدایی آن را سخت تر می کند.

کلمات کلیدی async  وawait  از نسخه 5 زبان سی شارپ اضافه شده اند. و این ویژگی ها می توانند تا میزان زیادی، برنامه نویسی غیز همزمان را ساده تر کنند.

مایکروسافت Task-based Asynchronous Pattern را برای اجرای برنامه‌نویسی غیرهمزمان در برنامه‌هایNET Framework. یاNET Core. با استفاده از کلمات کلیدی async و await و  Task یا کلاس Task<TResult> توصیه می‌کند.

Async  و await  دو برچسب هستند که مشخص می کنند در کدام بخش کد، پاسخ دهی باید بعد از اتمام کار از سر گرفته شود.

1. Asynchronous Task Composition :

متد async مقداری را به کد فراخوانی برمی گرداند، از async به همراه await و Task استفاده می کنیم.

کلمه کلیدی await منتظر متد async می ماند تا زمانی که مقداری را برگرداند. بنابراین thread مربوط به برنامه اصلی در آنجا متوقف می شود تا زمانی که مقدار بازگشتی را دریافت کند.

کلاس Task یک عملیات ناهمگام را نشان می دهد و کلاس عمومی Task<TResult> عملیاتی را نشان می دهد که می تواند یک مقدار را برگرداند. برای مثال از await Task.Delay(4000) استفاده می کنیم که عملیات async را شروع کرد که به مدت 4 ثانیه می‌خوابد و await یک thread را تا 4 ثانیه نگه می‌دارد.

در پروژه نمونه AsyncAwaitTasksSample تمامی این عملگرهای قید شده در این سند را با فایل های کلاسی هم نام با نوع عملگر و کلمه کلیدی می توانید مشاهده کنید.

<https://github.com/BaharehSmaili/AsyncAwaitTasksSample>

1. Handling Asynchronous Exceptions :

مدل برنامه نویسی ناهمزمان در سی شارپ به توسعه دهندگان این امکان را می دهد که با استفاده از قدرت هسته های CPU شما کدی بنویسند که مقیاس پذیر، پاسخگو و قابل اعتماد باشد. با این حال، یک چیزی که باید درک کرد این است که محدودیت‌هایی در مورد اینکه چگونه برنامه‌نویسان می‌توانند استثناها را در یک روش ناهمزمان مدیریت کنند وجود دارد.

Exception ها خطاهایی هستند که در زمانی که برنامه شما در حال اجرا است رخ می دهد. اگر Exception ها در کد شما به درستی مدیریت نشوند، رشته اصلی برنامه زمانی که یک Exception اتفاق بی افتد و اجرا شود، خاتمه می یابد. مدیریت Exception به عنوان فرآیند رسیدگی به Exception ها در کد برنامه شما تعریف می شود.

حالا چگونه با استفاده از Asynchronous این Exception ها را مدریت کنیم ؟

برنامه نویسی ناهمزمان (Asynchronous) به توسعه دهندگان این امکان را می دهد تا بدون توقف رشته اصلی یا اجرایی برنامه، عملیاتی با منابع فشرده انجام دهند. برنامه نویسی ناهمزمان (Asynchronous) چندین مزیت دارد. برای شروع، به بهبود پاسخگویی و مقیاس پذیری برنامه شما به طور قابل توجهی کمک می کند. با این حال، باید توجه داشت که انتشار Exception در کدهای همزمان و ناهمزمان متفاوت عمل می کند.

برنامه‌نویسی ناهمزمان (Asynchronous) شما را قادر می‌سازد برنامه‌هایی بسازید که هر دستور یا دستوری را مسدود نکند و به برنامه اجازه می‌دهد به اجرای دستورالعمل‌های دیگر ادامه دهد. در نتیجه، از سایر مزیت های برنامه نویسی ناهمزمان (Asynchronous) این است که می توانید برنامه هایی را توسعه دهید که مقیاس پذیرتر و پاسخگوتر باشند.

اگر یک Exception در یک متد ناهمزمان (Asynchronous) ایجاد شود که یک شی Task را برمی گرداند، Exception در یک نمونه از کلاس AggregateException پیچیده می شود و به متد فراخوانی ارسال می شود. نکته مهمی که در اینجا باید به آن توجه کرد این است که وقتی منتظر نمونه Task هستید، می‌توانید فقط اولین Exception را دریافت کنید، حتی اگر چندین Exception در روش ناهمزمان (Asynchronous) شما وجود داشته باشد. نمونه این مبحث در پروژه معرفی شده به صورت عملی در متد ThrowMultipleExceptionsAsync پیاده سازی شده است.

کلمه کلیدی await اولین Exception را از نمونه Task ایجاد می کند که به دلیل آن فقط یک Exception دریافت خواهید کرد. برای دریافت تمام Exception هایی که در متد async شما وجود دارد، باید از ویژگی Exception یک نمونه Task ایجاد و استفاده نمایید. نمونه این مبحث در پروژه معرفی شده به صورت عملی در متد ThrowMultipleAllExceptionsAsync پیاده سازی شده است.

[https://github.com/BaharehSmaili/HandlingAsynchronousExceptions](https://github.com/BaharehSmaili/AsyncAwaitTasksSample)

1. تشخیص ناهمزمان و چند نخی (Distinguish Asynchronous And Multi-Threading)