بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

مدیریت کردن روابط موجودیت (Entity)

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

بارگذاری با تاخیر (Lazy Loading) در مقابل بارگذاری حریصانه(Eager Loading)

باید به این نکته توجه کرد که وقتی از EF وابسته به دیتابیس استفاده می کنیم چگونگی بارگذاری EF را متوجه شویم. همچنین بهتر است که دستورات SQL که EF بر پایه آنها ساخته شده را ببینید. برای ردگیری (trace) SQL، قطعه کد زیر را به BookServiceContext اضافه کنید:

```
public BookServiceContext() : base("name=BookServiceContext")
{
    // New code:
    this.Database.Log = s => System.Diagnostics.Debug.WriteLine(s);
}
```

اگر شما یک درخواست GET به /api/books بفرستید، داده JSON به صورت زیر بر می گردد:

```
{
    "BookId": 1,
    "Title": "Pride and Prejudice",
    "Year": 1813,
    "Price": 9.99,
    "Genre": "Comedy of manners",
    "AuthorId": 1,
    "Author": null
  },
```

همانطور که می بینید خاصیت Author با اینکه یک مقدار معتبر AuthorId دارد، مقدار null گرفته است. دلیل این موضوع برای آن است که EF روابط موجودیت های Author را بارگذاری نکرده است. trace دستورات SQL که در زیر ، این موضوع را تائید می کند:

```
SELECT
    [Extent1].[BookId] AS [BookId],
    [Extent1].[Title] AS [Title],
```

```
    [Extent1].[Year] AS [Year],
    [Extent1].[Price] AS [Price],
    [Extent1].[Genre] AS [Genre],
    [Extent1].[AuthorId] AS [AuthorId]
    FROM [dbo].[Books] AS [Extent1]
```

این SELECT از جدول book گرفته شده و به جدول author مربوط نمی شود. برای دیدن reference، در اینجا متدی در کلاس BooksController وجود دارد که لیستی از book را بر می گرداند.

```
  public IQueryable<Book> GetBooks()
{
    return db.Books;
}
```

حالا می خواهیم ببینیم چطور مقدار نویسنده را به عنوان قسمتی از داده JSON برگردانیم. سه راه برای بارگذاری داده های وابسته در EF وجود دارد: بارگذاری حریصانه(Eager Loading)، بارگذاری با تاخیر (Lazy Loading)، بارگذاری صریح (Explicit Loading). هر کدام از این روش ها دستورالعمل هایی دارند که درک آنها برای فهمیدن کارایی آنها نیاز است.

Eager Loading

با eager loading ، EF موجودیت های وابسته را به عنوان دستورات اولیه دیتابیس بارگذاری می کند. برای اجرای eager loading از **System.Data.Entity.Include** استفاده کنید.

```
public IQueryable<Book> GetBooks()
{
    return db.Books
        // new code:
        .Include(b => b.Author);
}
```

این کد به EF اطلاع می دهد که دارای داده Author در دستورات می باشد. اگر شما این تغییرات را انجام دهید و برنامه را اجرا کنید داده JSON به صورت زیر می باشد:

```
{
    "BookId": 1,
    "Title": "Pride and Prejudice",
    "Year": 1813,
    "Price": 9.99,
    "Genre": "Comedy of manners",
    "AuthorId": 1,
    "Author": {
```

```
      "AuthorId": 1,
      "Name": "Jane Austen"
    }
  },
```

ردگیری log به ما نشان می دهد EF بین جداول Book و Author دستور Join را اجرا کرده است:

```
{
    "BookId": 1,
    "Title": "Pride and Prejudice",
    "Year": 1813,
    "Price": 9.99,
    "Genre": "Comedy of manners",
    "AuthorId": 1,
    "Author": {
      "AuthorId": 1,
      "Name": "Jane Austen"
    }
  },
```

Lazy Loading

با استفاده از Lazy loading، EF به صورت خودکار entity وابسته را زمانی که خاصیت navigation برای آن entity بصورت غیر مرجع باشد بارگذاری می شود. برای فعال کردن lazy loading خاصیت navigation را مجازی (Virtual)کنید. برای مثال، درون کلاس Book:

```
public class Book
{
    // (Other properties)
    // Virtual navigation property
    public virtual Author Author { get; set; }
}
```

حالا کد زیر را در نظر بگیرید:

```
var books = db.Books.ToList();  // Does not load authors
var author = books[0].Author;   // Loads the author for books[0]
```

زمانی که lazy loading فعال می شود، دسترسی به خاصیت Author در book[0] برای Author باعث تبدیل EF به دستورات دیتابیس می شود.

EF هر بار یک دستور نیاز دارد تا وابستگی entity را بازیابی کند. در واقع، lazy loading برای objectهایی که شما serial کرده اید مناسب می باشد. برای مثال، دستورات SQL زمانی که EF لیست book را که با lazy

loading فعال کرده است serial می کند. شما می توانید ببینید که EF سه دستور جدا را برای سه author ساخته است.

```
SELECT
    [Extent1].[BookId] AS [BookId],
    [Extent1].[Title] AS [Title],
    [Extent1].[Year] AS [Year],
    [Extent1].[Price] AS [Price],
    [Extent1].[Genre] AS [Genre],
    [Extent1].[AuthorId] AS [AuthorId]
    FROM [dbo].[Books] AS [Extent1]
SELECT
    [Extent1].[AuthorId] AS [AuthorId],
    [Extent1].[Name] AS [Name]
    FROM [dbo].[Authors] AS [Extent1]
    WHERE [Extent1].[AuthorId] = @EntityKeyValue1
SELECT
    [Extent1].[AuthorId] AS [AuthorId],
    [Extent1].[Name] AS [Name]
    FROM [dbo].[Authors] AS [Extent1]
    WHERE [Extent1].[AuthorId] = @EntityKeyValue1
SELECT
    [Extent1].[AuthorId] AS [AuthorId],
    [Extent1].[Name] AS [Name]
    FROM [dbo].[Authors] AS [Extent1]
    WHERE [Extent1].[AuthorId] = @EntityKeyValue1
```

Eager loading باعث می شود EF بتواند دستورات Join بسیار پیچیده بسازد. همچنین شاید شما به روابط entity ها برای یک زیرمجموعه کوچک از داده نیاز داشته باشید و همین باعث تاثیرپذیر تر شدن lazy loading شود.

یکی از راههای چشم پوشی از مشکلات serial کردن این است که DTO ها (data transfer object) را به جای object های entity سریال کنید.

Explicit loading

Explicit loading بسیار شبیه lazy loading می باشد به جز آنکه صراحتا وابستگی داده را در کد دریافت می کند و به طور خودکار در زمان دسترسی به خاصیت navigation اتفاق نمی افتد. Explicit loading کنترل بیشتری برای بارگذاری وابستگی داده به شما می دهد ولی به کدهای اضافی احتیاج دارد.

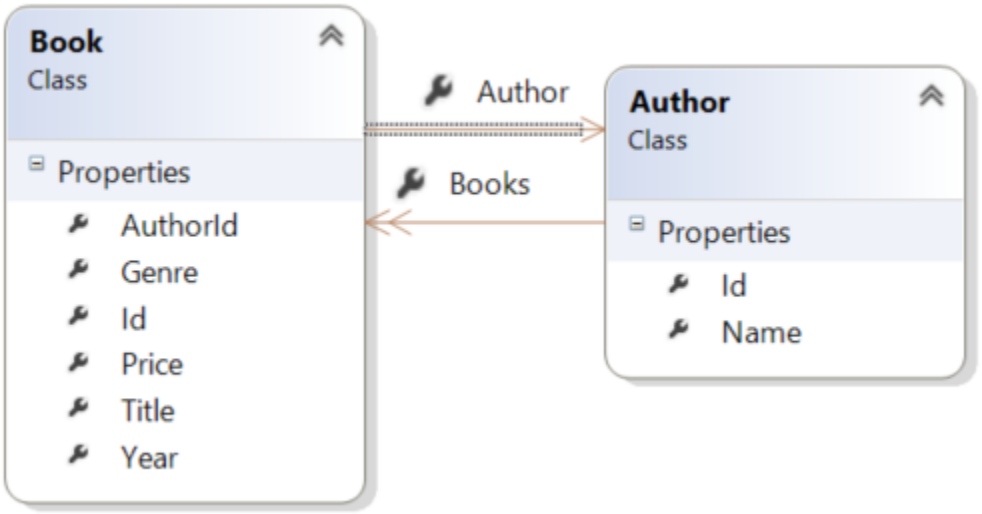
خاصیت های جهت دهی و مراجع دایره ای(Circular References)

وقتی ما model های book و author را تعریف کردیم، یک خاصیت navigation به کلاس book برای رابطه book-Author دادیم اما خاصیت navigation در مسیر های دیگر را تعریف نکردیم.

اگر خاصیت navigation را با کلاس Author متناظر کنیم چه اتفاقی می افتد؟

```
public class Author
{
    public int AuthorId { get; set; }
    [Required]
    public string Name { get; set; }
    public ICollection<Book> Books { get; set; }
}
```

متاسفانه این باعث یک مشکل در زمانی که شما model ها را serial کردید می شود. اگر شما داده وابسته را بارگذاری کنید، یک گراف دایره ای ساخته اید.



وقتی JSON یا XML سعی می کنند گراف را سریال کنند، دچار خطا می شوند. دو فرمت سریال کننده پیام، خطای متفاوتی دارند که در زیر نمونه خطای JSON را می بینید:

```
{
  "Message": "An error has occurred.",
  "ExceptionMessage": "The 'ObjectContent`1' type failed to serialize the response body
for content type
      'application/json; charset=utf-8'.",
```

```
  "ExceptionType": "System.InvalidOperationException",
  "StackTrace": null,
  "InnerException": {
    "Message": "An error has occurred.",
    "ExceptionMessage": "Self referencing loop detected with type
'BookService.Models.Book'.
        Path '[0].Author.Books'.",
    "ExceptionType": "Newtonsoft.Json.JsonSerializationException",
    "StackTrace": "..."
     }
}
```

مثال زیر هم نمونه خطای XML می باشد:

```
<Error>
  <Message>An error has occurred.</Message>
  <ExceptionMessage>The 'ObjectContent`1' type failed to serialize the response body for
content type
    'application/xml; charset=utf-8'.</ExceptionMessage>
  <ExceptionType>System.InvalidOperationException</ExceptionType>
  <StackTrace />
  <InnerException>
    <Message>An error has occurred.</Message>
    <ExceptionMessage>Object graph for type 'BookService.Models.Author' contains cycles
and cannot be
      serialized if reference tracking is disabled.</ExceptionMessage>
    <ExceptionType>System.Runtime.Serialization.SerializationException</ExceptionType>
    <StackTrace> ... </StackTrace>
  </InnerException>
</Error>
```

یکی از راه حل ها استفاده از DTO است که در مقاله بعد آن را توضیح خواهیم داد. روش دیگر نیز تنظیم کردن JSON و XML برای مدیریت چرخه گراف می باشد.