# 2. How to use EF Core?

## Adding EF Core to an application (一)

- Adding EF Core to an application is a multistep process:
  1. Choose a database provider, such as Postgres, SQLite, or MS SQL Server.
  2. Install the EF Core NuGet packages.
  3. Design app's DbContext and entities that make up your data model.

11

## Adding EF Core to an application (二)

- Adding EF Core to an application is a multistep process:
  4. Register app's DbContext with the ASP.NET Core DI container.
  5. Use EF Core to generate a migration describing your data model.
  6. Apply the migration to the database to update the database's schema.

12

## Step 1: Choosing a Database Provider

- EF Core supports a range of databases by using a provider model.
- The modular nature of EF Core means that you can use the same high-level API to program against different underlying databases; EF Core knows how to generate the necessary implementation specific code and SQL statements.

13

## Step 2: Installing EF Core

- Adding support for a given database involves adding the correct NuGet package to .csproj file, such as the following:
  - PostgreSQL—Npgsql.EntityFrameworkCore.PostgreSQL
  - Microsoft SQL Server—Microsoft.EntityFrameworkCore.SqlServer
  - MySQL—MySql.Data.EntityFrameworkCore
  - SQLite—Microsoft.EntityFrameworkCore.SQLite

14

## Project: Install EF Core

- LocalDB is the zero-configuration developer version of SQL Server.
- This project use LocalDB provided by Visual Studio, install the following packages:
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.Relational
  - Microsoft.EntityFrameworkCore.SqlServer
  - Microsoft.EntityFrameworkCore.Tools

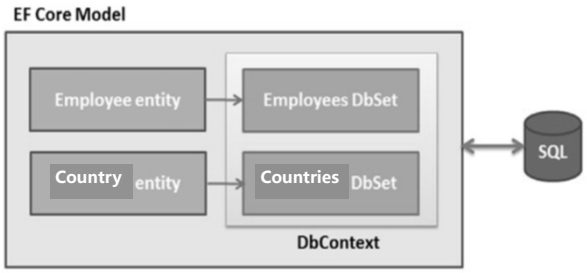
15

## Step3: Building Data Model（一）

- EF Core builds up its internal model of your database from the DbContext and entity models.
- EF Core heavily favors a convention over configuration approach in defining entities as POCO classes.
  - EF Core identifies the property with an Id suffix as the primary key of the table.

16

## Project: Design the Data Model




17

## Project: Define the Entity Class

```
public class Country
{
    public int CountryId { get; set; }
    public string CountryName { get; set; }
}
```


```
public class Employee
{
    public int EmployeeId { get; set; } //工号
    public string EmployeeName { get; set; } //姓名
    public string Title { get; set; } //职务
    public string Country { get; set; } //国籍
}
```

18

## Step3: Building Data Model（二）

- As well as defining the entities, define the DbContext for application.
- The DbContext is the heart of EF Core in application, used for all database calls.
  - The constructor receives a DbContextOptions<T> object, where T is the context class.
  - The constructor parameter will provide Entity Framework Core with the configuration information it needs to connect to the database server.

19

## Project: Define the DbContext Class


```
using Microsoft.EntityFrameworkCore; //DbContext所在名称空间

public class AppDbContext:DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options):
base(options)
    {
    }
    public DbSet<Employee> Employees { get; set; } //Employees数据表
    public DbSet<Country> Countries { get; set; }//Countriess数据表
}
```

20

## Step4: Registering a Data Context（一）

- As with any other service in ASP.Net Core, register AppDbContext with the dependency injection (DI) container.
- EF Core provides a generic AddDbContext<T> extension method for this purpose.

21

## Step4: Registering a Data Context (二)

- Entity Framework Core isn't tied to any specific database server.
- All of the functionality that is required for a specific database server is contained in a package called the database provider.
- Two steps to configure the database provider:
  - Set up the connection string;
  - Configure the application.

22

## Step 4: Registering a Data Context (三)

```
using Microsoft.EntityFrameworkCore;
WebApplicationBuillder builder = WebApplication.CreateBuilder(args);
var connString = builder.Configuration
        .GetConnectionString("DefaultConnection");

    Builder.Services.AddDbContext<AppDbContext>(
        options => options.UseSqlite(connString));

    WebApplication app = builder.Build();
    app.Run();
```

The connection string is taken from configuration, from the ConnectionStrings section.

Registers your app's DbContext by using it as the generic parameter

Specifies the database provider in the customization options for the DbContext.

23

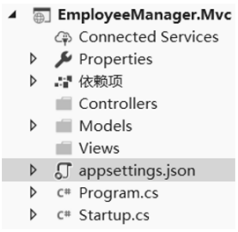
## Connection Strings

- A connection string gives a database provider the information it needs to connect to a specific database server.

| | |
|---|---|
| Server | • specifies the hostname for the server. |
| Database | • specifies the name of the database. |
| MultipleActiveResultSets | • determines whether a client can execute multiple active SQL statements on a single connection. |

24

## Project: Configure the Connection String




```
"ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\MSSQLLocalDB;
Database=EmployeeManager;
Trusted_Connection=True;
MultipleActiveResultSets=true"
}
```

25

## Project: Register the DbContext (一)

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>(
    options => options.UseSqlServer
    (builder.Configuration["ConnectionStrings:DefaultConnection"]));

var app = builder.Build();

app.UseStaticFiles();
app.MapDefaultControllerRoute();

app.Run();
```

**Program.cs** 26

## Project: Register the DbContext (二)

- The AddDbContext method creates a service for an Entity Framework Core context class.
- The method receives an options object that is used to select the database provider, which is done with the UseSqlServer method.
- The IConfiguration service is used to get the connection string for the database.

```
builder.Services.AddDbContext<AppDbContext>(
    options => options.UseSqlServer
    (builder.Configuration["ConnectionStrings:DefaultConnection"]));
```

27