# 小项目练习

## 1、使用 EF Core 来完成打开 MySQL 数据库、

创建数据表、

进行对象深度复制并将其存储到数据库的操作、

将数据库配置参数保存到本地配置文件。

### **1. 项目准备**

首先，你需要创建一个 C# 项目（例如控制台应用程序），并通过 NuGet 包管理器安装以下必要的库：

* Microsoft.EntityFrameworkCore：EF Core 的核心库。
* Pomelo.EntityFrameworkCore.MySql：用于与 MySQL 数据库交互的 EF Core 提供程序。
* Newtonsoft.Json：用于实现对象的深度复制。

### **2. 定义实体类**

定义要存储到数据库的实体类，这里以 Person 和 Address 为例，Person 类包含一个 Address 类型的引用，体现了实体之间的关联关系。

收起

csharp

using Newtonsoft.Json;

// 定义地址类public class Address{

public int Id { get; set; }

public string Street { get; set; }

public string City { get; set; }

public string ZipCode { get; set; }}

// 定义人员类，包含地址引用public class Person{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public Address Address { get; set; }

// 实现深度复制方法

public Person DeepCopy()

{

string json = JsonConvert.SerializeObject(this);

return JsonConvert.DeserializeObject<Person>(json);

}}

### **3. 创建 DbContext 类**

DbContext 类是 EF Core 与数据库交互的核心，它负责管理数据库连接、实体集合以及数据操作。

收起

csharp

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext{

public DbSet<Person> Persons { get; set; }

public DbSet<Address> Addresses { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// 配置数据库连接字符串，需根据实际情况修改

string connectionString = "server=localhost;user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;";

optionsBuilder.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));

}}

### MySQL 的AllowPublicKeyRetrieval 是什么？

AllowPublicKeyRetrieval 是 MySQL 连接字符串中的一个参数，主要用于处理 MySQL 客户端与服务器之间的公钥检索问题，下面从其作用、使用场景、安全风险以及配置示例几个方面详细介绍：

#### 作用

在 MySQL 8.0 及更高版本中，默认使用 caching\_sha2\_password 作为认证插件，这种认证方式在某些情况下需要客户端从服务器获取公钥来进行加密通信。AllowPublicKeyRetrieval 参数决定了客户端是否允许从服务器检索公钥。

#### 使用场景

* ****默认不允许情况****：当 AllowPublicKeyRetrieval 设置为 false（默认值）时，如果客户端需要公钥但没有提前配置，就会导致认证失败。例如，使用旧版本的 MySQL 连接器或者没有正确配置公钥的情况下，连接会报错。
* ****允许公钥检索****：将 AllowPublicKeyRetrieval 设置为 true，客户端在认证过程中如果需要公钥，会主动向服务器发起请求获取公钥，从而完成认证过程。这种方式适用于大多数需要使用 caching\_sha2\_password 认证插件的场景，尤其是在没有提前配置公钥的情况下。

#### 安全风险

将 AllowPublicKeyRetrieval 设置为 true 会带来一定的安全风险。因为允许客户端从服务器检索公钥可能会导致中间人攻击（MITM）的风险增加。攻击者可能会拦截客户端与服务器之间的通信，篡改公钥信息，从而窃取用户的敏感信息。因此，在生产环境中，如果不是必要情况，不建议将该参数设置为 true。

#### 配置示例

在不同的编程语言和框架中，配置 AllowPublicKeyRetrieval 参数的方式略有不同，以下是一些常见的示例：

#### **1.C# 使用 EF Core 连接 MySQL**

收起

csharp

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

string connectionString = "server=localhost;user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;";

optionsBuilder.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));

}}

#### **2.Java 使用 JDBC 连接 MySQL**

收起

java

import java.sql.Connection;import java.sql.DriverManager;import java.sql.SQLException;

public class MySQLConnectionExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database?AllowPublicKeyRetrieval=true";

String user = "root";

String password = "your\_password";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

System.out.println("Connected to the database!");

} catch (SQLException e) {

e.printStackTrace();

}

}}

#### 替代方案

为了避免使用 AllowPublicKeyRetrieval=true 带来的安全风险，可以采取以下替代方案：

* ****使用旧的认证插件****：将 MySQL 用户的认证插件改为 mysql\_native\_password，这样就不需要进行公钥检索。可以使用以下 SQL 语句修改用户的认证插件：

收起

sql

ALTER USER 'your\_user'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'your\_password';

* ****提前配置公钥****：在客户端配置文件中提前配置好服务器的公钥，避免在认证过程中进行公钥检索。不同的客户端和框架配置公钥的方式不同，需要参考相应的文档进行配置。

分享

在 C# 中使用 EF Core 连接 MySQL 时，如何配置其他连接参数？

除了 EF Core，还有哪些常见的.NET 数据库操作框架？

如何在生产环境中安全地配置 MySQL 的 AllowPublicKeyRetrieval 参数？

### **4. 配置 MySQL 数据库访问权限**

在 MySQL 中，你需要确保用户具有创建表和读写数据的权限。可以使用以下 SQL 语句为用户授予相应的权限：

收起

sql

-- 授予用户在指定数据库上的所有权限：

GRANT ALL PRIVILEGES ON your\_database.\* TO 'your\_username'@'localhost';

-- 刷新权限使更改生效：

FLUSH PRIVILEGES;

将 your\_database 替换为你实际使用的数据库名，your\_username 替换为你使用的 MySQL 用户名。

#### C# 配置 MySQL 数据库访问权限有哪些方式？权限有哪些？如何配置？

在 C# 中本身不能直接配置 MySQL 数据库的访问权限，但可以通过执行 SQL 语句或者调用 MySQL 管理工具相关的 API 来实现权限的配置。以下详细介绍相关信息：

#### 常见的权限类型

MySQL 提供了多种权限，以下是一些常见的权限及其作用：

##### ****数据操作权限****

* + SELECT：允许用户从表中查询数据。
  + INSERT：允许用户向表中插入新的数据记录。
  + UPDATE：允许用户修改表中的现有数据。
  + DELETE：允许用户从表中删除数据记录。

##### ****数据库结构操作权限****

* + CREATE：允许用户创建数据库、表、索引等数据库对象。
  + ALTER：允许用户修改数据库对象的结构，如修改表的列定义。
  + DROP：允许用户删除数据库、表等数据库对象。

##### ****用户和权限管理权限****

* + CREATE USER：允许创建新的 MySQL 用户。
  + GRANT OPTION：允许用户将自己拥有的权限授予其他用户。
  + REVOKE：允许用户撤销其他用户的权限。

#### 配置权限方式及具体操作

##### **1. 使用 MySqlConnector 执行 SQL 语句**

MySqlConnector 是一个用于在 .NET 应用程序中连接和操作 MySQL 数据库的开源库。以下是一个使用 MySqlConnector 执行 SQL 语句来配置 MySQL 数据库访问权限的示例：

收起

csharp

using MySqlConnector;using System;

class Program{

static void Main()

{

// 数据库连接字符串，使用具有足够权限的用户登录

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

try

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

// 打开数据库连接

connection.Open();

// 创建一个新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

using (MySqlCommand createUserCommand = new MySqlCommand(createUserQuery, connection))

{

createUserCommand.ExecuteNonQuery();

}

// 授予新用户对指定数据库的 SELECT 和 INSERT 权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

using (MySqlCommand grantPermissionsCommand = new MySqlCommand(grantPermissionsQuery, connection))

{

grantPermissionsCommand.ExecuteNonQuery();

}

// 刷新权限使更改生效

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

using (MySqlCommand flushPrivilegesCommand = new MySqlCommand(flushPrivilegesQuery, connection))

{

flushPrivilegesCommand.ExecuteNonQuery();

}

Console.WriteLine("用户创建和权限授予成功。");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

****代码解释****

1. ****连接数据库****：使用 MySqlConnection 类创建一个数据库连接对象，并打开连接。需要使用具有足够权限（如 root 用户）的用户登录，以便执行用户创建和权限授予操作。
2. ****创建新用户****：执行 CREATE USER 语句创建一个新的 MySQL 用户。
3. ****授予权限****：执行 GRANT 语句为新用户授予对指定数据库（your\_database）中所有表的 SELECT 和 INSERT 权限。
4. ****刷新权限****：执行 FLUSH PRIVILEGES 语句使权限更改立即生效。

##### 2. 调用 MySQL 管理工具的 API（较复杂，一般不常用）

一些 MySQL 管理工具提供了 API 供外部程序调用，通过调用这些 API 可以实现权限配置。但这种方式相对复杂，需要了解管理工具的 API 文档和使用方法。例如，某些云服务提供商提供的 MySQL 数据库管理服务，可能会有相应的 SDK 来管理数据库用户和权限。

**注意事项**

* ****权限管理安全****：在配置权限时，应遵循最小权限原则，只授予用户完成其工作所需的最少权限，以降低安全风险。
* ****异常处理****：在执行 SQL 语句时，可能会出现各种异常，如数据库连接失败、SQL 语法错误等，需要进行适当的异常处理。
* ****权限刷新****：每次修改用户权限后，都需要执行 FLUSH PRIVILEGES 语句使更改生效。

##### 1. MySQL.Data

* ****简介****：MySQL.Data 是 MySQL 官方提供的用于 .NET 平台的 [ADO.NET](https://ado.net/" \t "https://www.doubao.com/chat/_blank) 数据提供程序，它提供了一系列类和方法，用于在 C# 程序中连接、操作和管理 MySQL 数据库，也可用于配置数据库访问权限。
* ****使用示例****

收起

csharp

using System;using MySql.Data.MySqlClient;

class Program{

static void Main()

{

// 数据库连接字符串，使用具有足够权限的用户登录

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

try

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

// 打开数据库连接

connection.Open();

// 创建一个新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

using (MySqlCommand createUserCommand = new MySqlCommand(createUserQuery, connection))

{

createUserCommand.ExecuteNonQuery();

}

// 授予新用户对指定数据库的 SELECT 和 INSERT 权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

using (MySqlCommand grantPermissionsCommand = new MySqlCommand(grantPermissionsQuery, connection))

{

grantPermissionsCommand.ExecuteNonQuery();

}

// 刷新权限使更改生效

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

using (MySqlCommand flushPrivilegesCommand = new MySqlCommand(flushPrivilegesQuery, connection))

{

flushPrivilegesCommand.ExecuteNonQuery();

}

Console.WriteLine("用户创建和权限授予成功。");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

* ****特点****：
  + 官方支持，与 MySQL 数据库的兼容性好。
  + 功能全面，提供了丰富的类和方法来处理各种数据库操作。

##### **2. Dapper**

* ****简介****：Dapper 是一个轻量级的对象映射器，它本身不是专门针对 MySQL 的库，但可以与 MySqlConnector 或 MySQL.Data 结合使用，简化数据库操作代码。通过它可以更方便地执行 SQL 语句来配置 MySQL 数据库访问权限。
* ****使用示例****

收起

csharp

using System;using Dapper;using MySqlConnector;

class Program{

static void Main()

{

// 数据库连接字符串，使用具有足够权限的用户登录

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

try

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

// 打开数据库连接

connection.Open();

// 创建一个新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

connection.Execute(createUserQuery);

// 授予新用户对指定数据库的 SELECT 和 INSERT 权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

connection.Execute(grantPermissionsQuery);

// 刷新权限使更改生效

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

connection.Execute(flushPrivilegesQuery);

Console.WriteLine("用户创建和权限授予成功。");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

* ****特点****：
  + 性能高，在执行 SQL 查询和命令时速度快。
  + 代码简洁，通过扩展方法简化了数据库操作代码。

##### **3. Entity Framework Core with Pomelo**

* ****简介****：Entity Framework Core 是微软开发的一个跨平台的对象关系映射（ORM）框架，Pomelo.EntityFrameworkCore.MySql 是用于 MySQL 数据库的 EF Core 提供程序，使用它可以通过面向对象的方式来操作数据库，包括配置数据库访问权限。
* ****使用示例****

收起

csharp

using Microsoft.EntityFrameworkCore;

using System;

// 定义 DbContext 类

public class MyDbContext : DbContext{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

optionsBuilder.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));

}}

class Program{

static void Main()

{

try

{

using (var context = new MyDbContext())

{

// 创建一个新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

context.Database.ExecuteSqlRaw(createUserQuery);

// 授予新用户对指定数据库的 SELECT 和 INSERT 权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

context.Database.ExecuteSqlRaw(grantPermissionsQuery);

// 刷新权限使更改生效

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

context.Database.ExecuteSqlRaw(flushPrivilegesQuery);

Console.WriteLine("用户创建和权限授予成功。");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

* ****特点****：
  + 面向对象编程，通过实体类和 DbContext 来操作数据库，提高了开发效率。
  + 支持数据库迁移，方便进行数据库结构的版本控制。

#### C#可以在哪里配置 MySQL 数据库访问权限?

分享

在 C# 里，自身无法直接对 MySQL 数据库的访问权限进行配置，不过可以借助执行 SQL 语句或者调用 MySQL 管理工具的 API 来实现权限配置。下面详细介绍不同的配置途径：

##### 1.通过代码执行 SQL 语句配置

在 C# 代码里执行 MySQL 的 SQL 语句，从而完成数据库访问权限的配置。常用的库有 MySqlConnector 和 MySQL.Data。

##### **使用 MySqlConnector**

收起

csharp

using MySqlConnector;using System;

class Program{

static void Main()

{

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

try

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

// 创建新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

using (MySqlCommand createUserCommand = new MySqlCommand(createUserQuery, connection))

{

createUserCommand.ExecuteNonQuery();

}

// 授予权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

using (MySqlCommand grantPermissionsCommand = new MySqlCommand(grantPermissionsQuery, connection))

{

grantPermissionsCommand.ExecuteNonQuery();

}

// 刷新权限

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

using (MySqlCommand flushPrivilegesCommand = new MySqlCommand(flushPrivilegesQuery, connection))

{

flushPrivilegesCommand.ExecuteNonQuery();

}

Console.WriteLine("权限配置成功");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

##### **使用 MySQL.Data**

收起

csharp

using MySql.Data.MySqlClient;using System;

class Program{

static void Main()

{

string connectionString = "Server=localhost;Database=mysql;Uid=root;Pwd=your\_password;";

try

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

// 创建新用户

string createUserQuery = "CREATE USER 'newuser'@'%' IDENTIFIED BY 'newpassword';";

using (MySqlCommand createUserCommand = new MySqlCommand(createUserQuery, connection))

{

createUserCommand.ExecuteNonQuery();

}

// 授予权限

string grantPermissionsQuery = "GRANT SELECT, INSERT ON your\_database.\* TO 'newuser'@'%';";

using (MySqlCommand grantPermissionsCommand = new MySqlCommand(grantPermissionsQuery, connection))

{

grantPermissionsCommand.ExecuteNonQuery();

}

// 刷新权限

string flushPrivilegesQuery = "FLUSH PRIVILEGES;";

using (MySqlCommand flushPrivilegesCommand = new MySqlCommand(flushPrivilegesQuery, connection))

{

flushPrivilegesCommand.ExecuteNonQuery();

}

Console.WriteLine("权限配置成功");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

}}

##### **2.借助 MySQL 管理工具**

除了代码方式，还能使用 MySQL 管理工具来配置访问权限，这些工具可以独立于 C# 程序运行。

##### **MySQL Workbench**

这是 MySQL 官方推出的可视化管理工具，通过它能图形化地配置数据库访问权限，具体步骤如下：

1. 开启 MySQL Workbench 并连接到 MySQL 服务器。
2. 选择 “管理” -> “用户和权限”。
3. 在 “用户和权限” 窗口，点击 “添加账户” 创建新用户，或者选择已有用户进行权限修改。
4. 在 “登录信息” 选项卡设置用户名、主机和密码。
5. 在 “架构权限” 选项卡选择要授予权限的数据库和表，勾选相应权限。
6. 在 “管理权限” 选项卡可授予高级管理权限。
7. 配置完成后，点击 “应用” 保存更改。

##### **Navicat for MySQL**

Navicat 是一款功能强大的数据库管理工具，其配置权限的步骤如下：

1. 打开 Navicat 并连接到 MySQL 服务器。
2. 右键点击连接，选择 “用户管理”。
3. 在 “用户管理” 窗口，点击 “新建用户” 创建新用户，或选择已有用户修改权限。
4. 在 “常规” 选项卡设置用户名、主机和密码。
5. 在 “权限” 选项卡选择要授予的权限，可按数据库、表或具体操作进行设置。
6. 配置完成后，点击 “确定” 保存更改。

**注意事项**

* ****安全问题****：在配置权限时，要遵循最小权限原则，仅授予用户完成工作所需的最少权限，以此降低安全风险。
* ****权限刷新****：每次使用 SQL 语句修改用户权限后，都要执行 FLUSH PRIVILEGES 语句使更改立即生效。

### **5. 使用迁移创建数据表———失败(已解决)**

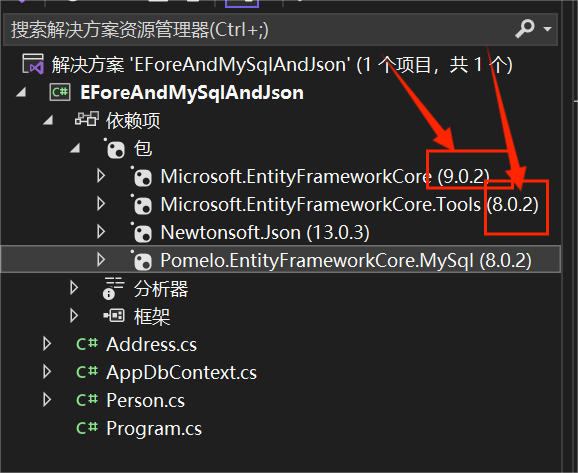
在包管理器控制台中执行以下命令来创建迁移文件并更新数据库，从而创建相应的数据表：

Add-Migration InitialCreate

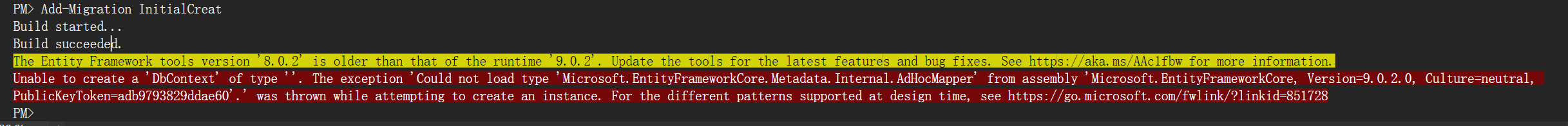
Update-Database

* Add-Migration InitialCreate：该命令会根据定义的实体类和 DbContext 生成一个迁移文件，记录数据库结构的变化。
* Update-Database：此命令会将迁移文件中的更改应用到数据库，创建相应的数据表。

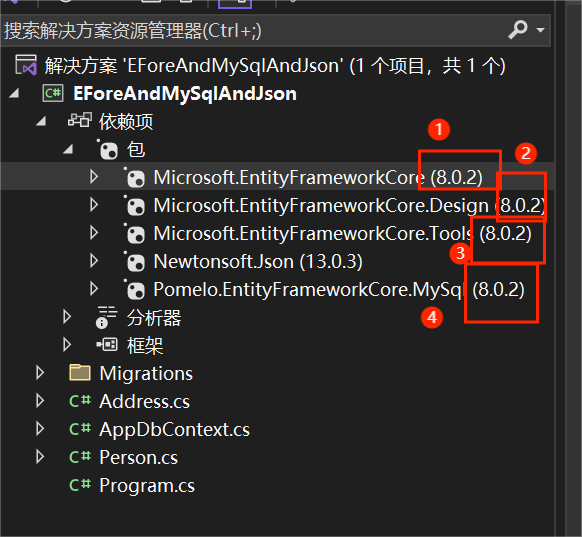
#### 失败原因：包版本不对



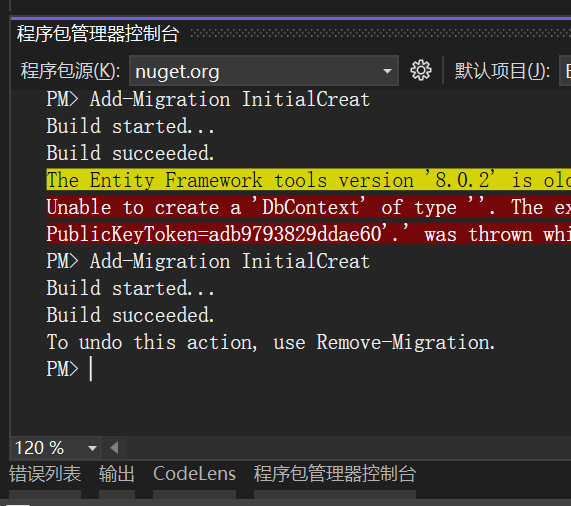
数据迁移报错，错误提示：



删除错误依赖包，统一4个依赖包版本：

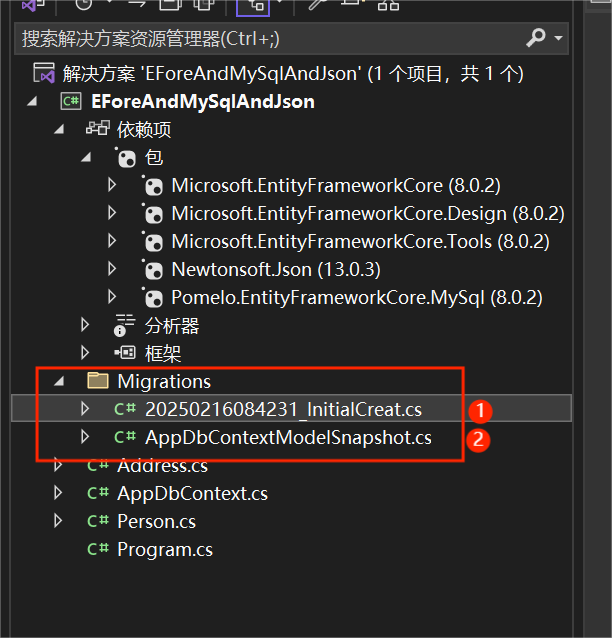


运行OK：

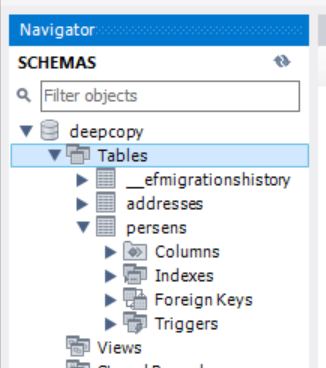
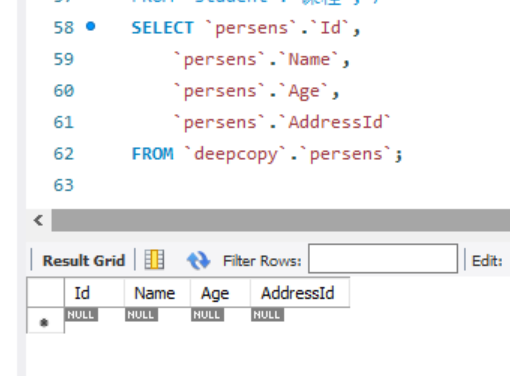


解决方案自动添加一个文件夹Migrations，里面新建了两个文件。

奇怪，第二遍测试，不显示这个文件夹。



#### 数据库查看结果：

 ---

### 5-1、在迁移过程中，如果数据库结构发生变化，如何更新迁移？

在使用 Entity Framework Core（EF Core）进行数据库迁移时，当数据库结构发生变化（例如添加、修改或删除实体类的属性，或者更改实体类之间的关系），需要更新迁移以反映这些变化。以下是详细的操作步骤和示例：

#### 1. 修改实体类

假设你已经有了一个简单的 Product 实体类和对应的 DbContext 类，现在需要对 Product 类进行修改，例如添加一个新的属性。

收起

csharp

// 原有的 Product 类public class Product{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }}

// 修改后的 Product 类，添加了 Description 属性public class Product{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public string Description { get; set; }}

#### 2. 创建新的迁移

在包管理器控制台（PMC）或命令行中执行以下命令,创建一个新的迁移，该迁移将记录数据库结构的变化：

##### **使用包管理器控制台（适用于 Visual Studio 项目）**

Add-Migration AddDescriptionToProduct

这里的 AddDescriptionToProduct 是迁移的名称，你可以根据实际的更改内容自定义迁移名称，以便清晰地描述这次迁移所做的更改。

##### **使用命令行（适用于 .NET Core 项目）**

收起

plaintext

dotnet ef migrations add AddDescriptionToProduct

执行该命令后，EF Core 会分析当前实体类的状态与上一次迁移时的状态之间的差异，并生成一个新的迁移文件。这个文件通常位于项目的 Migrations 文件夹中，包含了用于更新数据库结构的 SQL 语句。

#### 3. 应用新的迁移到数据库

创建新的迁移文件后，需要将其应用到数据库中，以更新数据库结构。同样可以使用包管理器控制台或命令行来完成：

##### **使用包管理器控制台**

收起

plaintext

Update-Database

##### **使用命令行**

收起

plaintext

dotnet ef database update

执行这些命令后，EF Core 会将新的迁移文件中的 SQL 语句应用到数据库，从而更新数据库结构，使其与当前实体类的定义保持一致。

#### 4. 回滚迁移（可选）

如果在应用迁移后发现问题，或者需要将数据库结构恢复到之前的状态，可以使用回滚操作。可以指定回滚到某个特定的迁移版本：

##### **使用包管理器控制台**

收起

plaintext

Update-Database <PreviousMigrationName>

其中 <PreviousMigrationName> 是你要回滚到的迁移的名称。

##### **使用命令行**

收起

plaintext

dotnet ef database update <PreviousMigrationName>

#### 5. 删除迁移（可选）

如果新创建的迁移还未应用到数据库，并且你想要删除它，可以使用以下命令：

##### **使用包管理器控制台**

收起

plaintext

Remove-Migration

##### 使用命令行

收起

plaintext

dotnet ef migrations remove

##### **注意事项**

* 迁移文件是版本化的，每次创建新的迁移时，EF Core 会记录迁移的顺序，确保数据库结构的更新是按照正确的顺序进行的。
* 在进行迁移操作之前，建议备份数据库，以防意外情况导致数据丢失或损坏。
* 不同的数据库可能对迁移操作有一些细微的差异，需要根据实际使用的数据库进行相应的调整。

### **6. 深度复制对象并保存到数据库**

以下是完整的代码示例，展示了如何创建原始对象、进行深度复制，并将复制后的对象保存到 MySQL 数据库。

收起

csharp

using System;

class Program{

static void Main()

{

// 创建原始对象

Address originalAddress = new Address

{

Street = "123 Elm St",

City = "Springfield",

ZipCode = "12345"

};

Person originalPerson = new Person

{

Name = "Alice",

Age = 30,

Address = originalAddress

};

// 深度复制对象

Person copiedPerson = originalPerson.DeepCopy();

using (var context = new AppDbContext())

{

try

{

// 将复制后的对象添加到 DbContext

context.Persons.Add(copiedPerson);

// 保存更改到数据库

int rowsAffected = context.SaveChanges();

if (rowsAffected > 0)

{

Console.WriteLine("数据插入成功！");

}

else

{

Console.WriteLine("数据插入失败。");

}

}

catch (Exception ex)

{

Console.WriteLine($"发生错误: {ex.Message}");

}

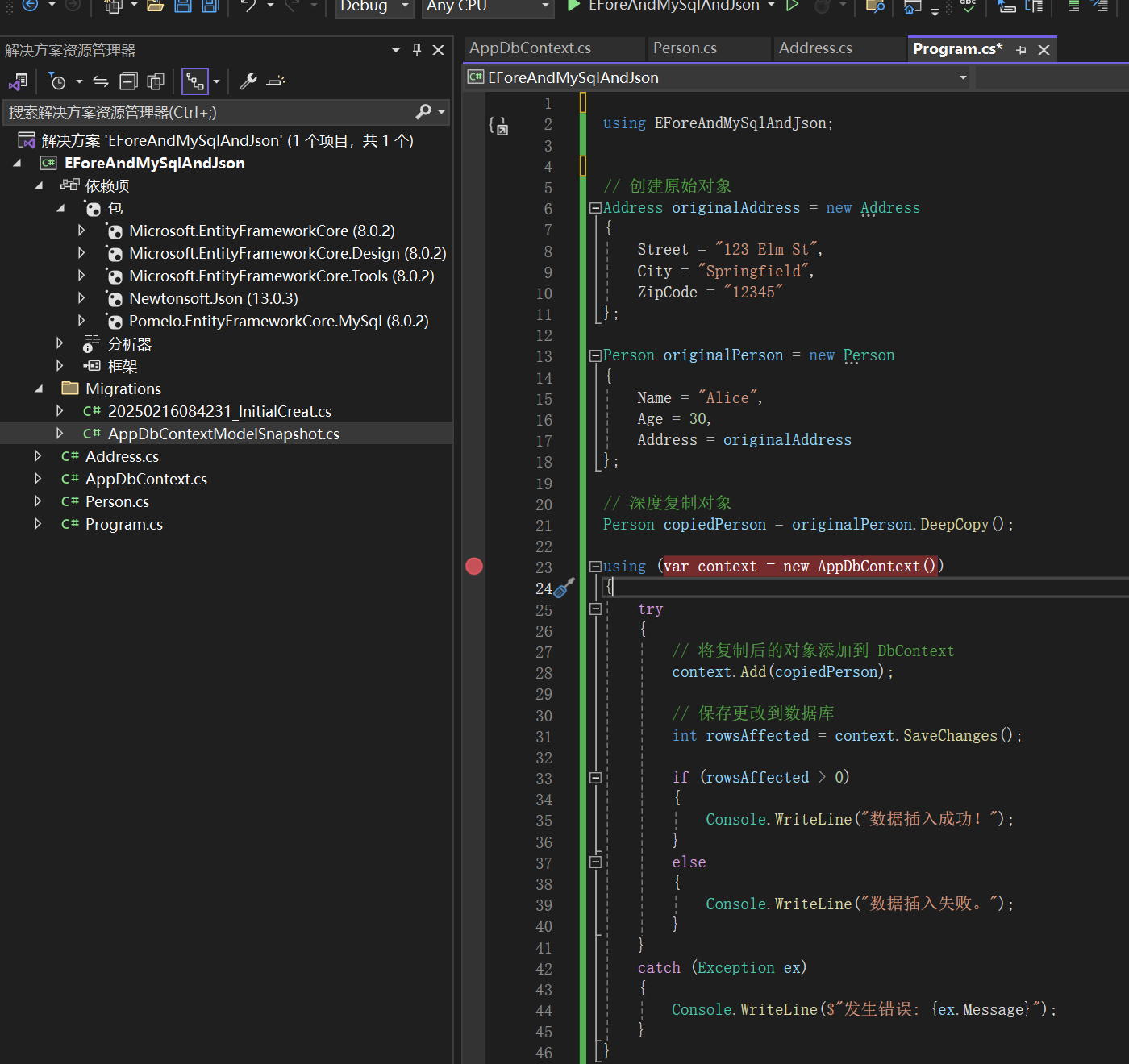
}

}}

**代码解释**

* ****实体类****：Person 和 Address 类定义了数据的结构，Person 类中的 Address 属性建立了与 Address 类的关联。
* **DbContext**配置****：AppDbContext 类继承自 DbContext，通过 OnConfiguring 方法配置 MySQL 数据库的连接字符串。
* ****深度复制****：Person 类中的 DeepCopy 方法使用 Newtonsoft.Json 库将对象序列化为 JSON 字符串，再反序列化为新对象，实现深度复制。
* ****数据插入****：创建 AppDbContext 实例，将复制后的 Person 对象添加到 Persons 集合中，调用 SaveChanges 方法将更改保存到数据库。

#### 6-1.在控制台实例化person和address，将数据写入数据库deepcopy的数据表Persons和Addresses。



### **7.创建本地配置文件**

在项目根目录下创建一个名为 appsettings.json 的文件，并添加以下内容：

{

"ConnectionStrings": {

"DefaultConnection": "server=localhost;user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;"

}}

请将 your\_password 替换为你的 MySQL 密码，your\_database 替换为你要使用的数据库名称。

注意事项

* 确保 MySQL 服务器正在运行，并且数据库用户具有创建表和插入数据的权限。
* 及时处理可能出现的异常，例如数据库连接失败、数据插入错误等，以保证程序的健壮性。
* 对于连接字符串中的 AllowPublicKeyRetrieval=true，在某些 MySQL 版本和配置下可能需要添加该参数以解决公钥检索问题。

#### 7-1.C#列举几个在项目根目录下创建一个名为 appsettings.json 的文件的方式

并添加以下内容：

{ "ConnectionStrings": { "DefaultConnection": "server=[localhost](https://localhost/" \t "https://www.doubao.com/chat/_blank);user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;" }}

分享

以下为你介绍几种在 C# 项目根目录下创建 appsettings.json 文件并添加指定内容的方式：

##### 方式一：手动创建

这是最为直接的方法，适用于你想要快速完成文件创建和内容添加的场景。

###### 步骤

1. 在 Visual Studio 或其他代码编辑器中打开你的项目。
2. 找到项目根目录（通常包含 .csproj 文件的目录）。
3. 在项目根目录下，右键点击空白处，选择 “添加” -> “新建项”。
4. 在弹出的 “添加新项” 对话框中，选择 “JSON 文件”，将文件名命名为 appsettings.json，然后点击 “添加”。
5. 在新创建的 appsettings.json 文件中，添加以下内容：

收起

json

{

"ConnectionStrings": {

"DefaultConnection": "server=localhost;user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;"

}}

将 your\_password 替换为你的 MySQL 数据库密码，your\_database 替换为你要使用的数据库名称。

##### **方式二：使用代码创建**

如果你希望通过代码自动创建并添加内容，可以使用以下示例代码。此方式适合在项目初始化脚本或自动化部署流程中使用。

using System.IO;

class Program{

static void Main()

{

// 定义 appsettings.json 文件的内容

string jsonContent = @"{

""ConnectionStrings"": {

""DefaultConnection"": ""server=localhost;user=root;password=your\_password;database=your\_database;AllowPublicKeyRetrieval=true;""

}

}";

// 获取项目根目录

string projectRoot = Directory.GetCurrentDirectory();

// 拼接 appsettings.json 文件的完整路径

string filePath = Path.Combine(projectRoot, "appsettings.json");

// 将内容写入文件

File.WriteAllText(filePath, jsonContent);

System.Console.WriteLine("appsettings.json 文件已创建并添加内容。");

}}

代码解释

1. ****定义 JSON 内容****：使用字符串变量 jsonContent 存储 appsettings.json 文件的内容，注意需要对双引号进行转义。
2. ****获取项目根目录****：使用 Directory.GetCurrentDirectory() 方法获取当前项目的根目录。
3. ****拼接文件路径****：使用 Path.Combine 方法将项目根目录和文件名拼接成完整的文件路径。
4. ****写入文件****：使用 File.WriteAllText 方法将 JSON 内容写入到 appsettings.json 文件中。

##### **方式三：使用模板引擎（如 Razor）**

如果你需要根据不同的环境动态生成 appsettings.json 文件的内容，可以使用 Razor 模板引擎。这种方式适合在复杂的配置管理场景中使用。

###### **步骤**

1. 安装 Microsoft.AspNetCore.Razor.Runtime NuGet 包。
2. 创建一个 Razor 模板文件，例如 appsettings.template.cshtml，内容如下：

收起

razor

{

"ConnectionStrings": {

"DefaultConnection": "server=localhost;user=root;password=@Model.Password;database=@Model.Database;AllowPublicKeyRetrieval=true;"

}

}

1. 使用以下代码生成 appsettings.json 文件：

收起

csharp

using Microsoft.AspNetCore.Razor.Language;using System.IO;

class Program{

class AppSettingsModel

{

public string Password { get; set; }

public string Database { get; set; }

}

static void Main()

{

// 定义模型数据

var model = new AppSettingsModel

{

Password = "your\_password",

Database = "your\_database"

};

// 读取 Razor 模板文件

string template = File.ReadAllText("appsettings.template.cshtml");

// 配置 Razor 引擎

var engine = RazorProjectEngine.Create(RazorConfiguration.Default);

// 解析模板

var document = engine.Process("appsettings.template.cshtml", template);

// 生成 C# 代码

var csharpDocument = document.GetCSharpDocument();

// 动态编译并执行模板

var result = csharpDocument.GeneratedCode

.Replace("@Model.Password", model.Password)

.Replace("@Model.Database", model.Database);

// 获取项目根目录

string projectRoot = Directory.GetCurrentDirectory();

// 拼接 appsettings.json 文件的完整路径

string filePath = Path.Combine(projectRoot, "appsettings.json");

// 将内容写入文件

File.WriteAllText(filePath, result);

System.Console.WriteLine("appsettings.json 文件已创建并添加内容。");

}}

代码解释

1. ****定义模型类****：创建 AppSettingsModel 类，用于存储数据库密码和数据库名称。
2. ****读取 Razor 模板文件****：使用 File.ReadAllText 方法读取 appsettings.template.cshtml 文件的内容。
3. ****配置 Razor 引擎****：使用 RazorProjectEngine.Create 方法创建 Razor 引擎。
4. ****解析模板并生成 C# 代码****：使用 engine.Process 方法解析模板，然后使用 document.GetCSharpDocument 方法生成 C# 代码。
5. ****动态替换模型数据****：将生成的 C# 代码中的 @Model.Password 和 @Model.Database 替换为实际的密码和数据库名称。
6. ****写入文件****：将替换后的内容写入到 appsettings.json 文件中。

#### 7-2.在 C# 中，将 appsettings.json 配置文件加载到数据库初始化代码中有多种常见的方式

以下为你详细介绍：

##### **方式一：使用 Microsoft.Extensions.Configuration 进行加载**

这是在 .NET Core 及 .NET 5+ 项目中最常用的方式，Microsoft.Extensions.Configuration 提供了强大的配置管理功能。

###### **步骤**

1. ****安装 NuGet 包****：确保项目中安装了 Microsoft.Extensions.Configuration.Json 包，你可以在包管理器控制台中执行以下命令进行安装：

收起

plaintext

Install-Package Microsoft.Extensions.Configuration.Json

1. ****编写代码加载配置文件并初始化数据库连接****：

收起

csharp

using Microsoft.Extensions.Configuration;using System;using System.IO;using MySqlConnector; // 假设使用 MySQL 数据库

class Program{

static void Main()

{

// 创建配置构建器

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)

.Build();

// 获取数据库连接字符串

string connectionString = configuration.GetConnectionString("DefaultConnection");

try

{

// 创建数据库连接

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

// 打开数据库连接

connection.Open();

Console.WriteLine("数据库连接已打开。");

// 这里可以执行数据库操作

}

}

catch (Exception ex)

{

Console.WriteLine($"数据库连接出错: {ex.Message}");

}

}}

###### **代码解释**

* ConfigurationBuilder：用于构建配置对象，通过 SetBasePath 方法设置配置文件的基础路径，AddJsonFile 方法添加 appsettings.json 文件。
* GetConnectionString：从配置中获取名为 DefaultConnection 的数据库连接字符串。
* MySqlConnection：使用获取到的连接字符串创建 MySQL 数据库连接，并打开连接。

##### **方式二：使用 JsonSerializer 手动解析（较基础的方式）**

这种方式不依赖于 Microsoft.Extensions.Configuration，直接使用 System.Text.Json 中的 JsonSerializer 手动解析 appsettings.json 文件。

#### **代码示例**

收起

csharp

using System;using System.IO;using System.Text.Json;using MySqlConnector;

class Program{

class AppSettings

{

public ConnectionStringsSection ConnectionStrings { get; set; }

}

class ConnectionStringsSection

{

public string DefaultConnection { get; set; }

}

static void Main()

{

try

{

// 读取 appsettings.json 文件内容

string json = File.ReadAllText("appsettings.json");

// 反序列化 JSON 内容到 AppSettings 对象

AppSettings settings = JsonSerializer.Deserialize<AppSettings>(json);

// 获取数据库连接字符串

string connectionString = settings.ConnectionStrings.DefaultConnection;

// 创建数据库连接

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

// 打开数据库连接

connection.Open();

Console.WriteLine("数据库连接已打开。");

// 这里可以执行数据库操作

}

}

catch (Exception ex)

{

Console.WriteLine($"出错: {ex.Message}");

}

}}

###### **代码解释**

* 定义 AppSettings 和 ConnectionStringsSection 类来表示 appsettings.json 文件的结构。
* 使用 File.ReadAllText 方法读取 appsettings.json 文件的内容。
* 使用 JsonSerializer.Deserialize 方法将 JSON 内容反序列化为 AppSettings 对象。
* 从反序列化后的对象中获取数据库连接字符串，并创建和打开数据库连接。

##### **方式三：使用依赖注入（适用于 [ASP.NET](https://asp.net/" \t "https://www.doubao.com/chat/_blank) Core 项目）**

在 [ASP.NET](https://asp.net/" \t "https://www.doubao.com/chat/_blank) Core 项目中，可以利用依赖注入机制将配置对象注入到需要使用的服务中。

###### **步骤**

1. ****配置依赖注入****：在 Startup.cs（.NET Core 3.1 及以前）或 Program.cs（.NET 5+）中进行配置。

收起

csharp

// Program.cs（.NET 6+ 示例）using Microsoft.AspNetCore.Hosting;using Microsoft.Extensions.Configuration;using Microsoft.Extensions.DependencyInjection;using Microsoft.Extensions.Hosting;using MySqlConnector;

var builder = WebApplication.CreateBuilder(args);

// 加载配置文件var configuration = builder.Configuration;

// 获取数据库连接字符串string connectionString = configuration.GetConnectionString("DefaultConnection");

// 注册数据库连接服务

builder.Services.AddScoped<MySqlConnection>(\_ => new MySqlConnection(connectionString));

var app = builder.Build();

// 示例：从服务容器中获取数据库连接using (var scope = app.Services.CreateScope()){

var connection = scope.ServiceProvider.GetRequiredService<MySqlConnection>();

try

{

connection.Open();

Console.WriteLine("数据库连接已打开。");

}

catch (Exception ex)

{

Console.WriteLine($"数据库连接出错: {ex.Message}");

}}

app.Run();

###### **代码解释**

* 使用 WebApplication.CreateBuilder 创建应用程序构建器，并通过 builder.Configuration 加载配置文件。
* 使用 GetConnectionString 方法获取数据库连接字符串。
* 使用 AddScoped 方法将 MySqlConnection 服务注册到依赖注入容器中。
* 在应用程序启动时，从服务容器中获取 MySqlConnection 实例并打开连接。

## 2、C#如何使用迁移创建SQL server数据表，详细步骤

在 C# 中使用 Entity Framework Core（EF Core）的迁移功能来创建数据表是一种常见且高效的方式。下面以一个简单的控制台应用程序为例，详细介绍使用迁移创建数据表的步骤。

### **1. 创建项目**

首先，打开 Visual Studio，创建一个新的 C# 控制台应用程序项目。

### **2. 安装必要的 NuGet 包**

EF Core 支持多种数据库，这里以 SQL Server 为例，需要安装以下 NuGet 包：

* Microsoft.EntityFrameworkCore.SqlServer：用于与 SQL Server 数据库交互的 EF Core 提供程序。
* Microsoft.EntityFrameworkCore.Tools：提供用于执行迁移命令的工具。

可以通过 NuGet 包管理器控制台执行以下命令进行安装：

收起

plaintext

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

### **3. 定义实体类**

实体类代表数据库中的表结构，以下是一个简单的 Product 实体类示例：

收起

csharp

public class Product{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }}

### **4. 创建 DbContext 类**

DbContext 类是 EF Core 与数据库交互的核心，它负责管理实体集合和数据库连接。

收起

csharp

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext{

public DbSet<Product> Products { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// 配置数据库连接字符串，根据实际情况修改

optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=YourDatabase;Trusted\_Connection=True;");

}}

### **5. 启用迁移功能**

在包管理器控制台中，确保项目上下文选择正确（即包含 DbContext 类的项目），然后执行以下命令启用迁移功能：

收起

plaintext

Add-Migration InitialCreate

* Add-Migration 是 EF Core 提供的命令，用于创建一个新的迁移。
* InitialCreate 是迁移的名称，你可以根据实际情况自定义。

执行该命令后，EF Core 会根据定义的实体类和 DbContext 生成一个迁移文件，该文件记录了数据库结构的初始状态。迁移文件通常位于项目的 Migrations 文件夹中，包含两个文件：一个是迁移的主文件，另一个是快照文件。

### **6. 应用迁移到数据库**

迁移文件生成后，需要将其应用到数据库中，以创建相应的数据表。在包管理器控制台中执行以下命令：

收起

plaintext

Update-Database

* Update-Database 命令会将最新的迁移文件应用到数据库，创建或更新数据库结构。

### **7. 验证数据表创建**

可以使用 SQL Server Management Studio（SSMS）或其他数据库管理工具连接到指定的数据库，查看是否成功创建了 Products 表。

### **完整代码示例**

收起

csharp

using Microsoft.EntityFrameworkCore;using System;

// 定义实体类public class Product{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }}

// 定义 DbContext 类public class ApplicationDbContext : DbContext{

public DbSet<Product> Products { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// 配置数据库连接字符串，根据实际情况修改

optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=YourDatabase;Trusted\_Connection=True;");

}}

class Program{

static void Main()

{

using (var context = new ApplicationDbContext())

{

try

{

// 确保数据库存在并应用最新迁移

context.Database.Migrate();

Console.WriteLine("数据库迁移成功！");

}

catch (Exception ex)

{

Console.WriteLine($"迁移过程中出现错误: {ex.Message}");

}

}

}}

### **注意事项**

* 迁移文件是版本化的，每次对实体类或 DbContext 进行更改后，都需要创建新的迁移并应用到数据库。
* 不同的数据库需要使用不同的 EF Core 提供程序，例如使用 MySQL 数据库需要安装 Pomelo.EntityFrameworkCore.MySql 包，并在 OnConfiguring 方法中使用 UseMySql 配置连接字符串。