本快速入门介绍了如何配置IdentityServer以使用EntityFramework（EF）作为此数据的存储机制

# **IdentityServer4.EntityFramework组件**

IdentityServer项目通过添加对IdentityServer4.EntityFramework Nuget包的引用。

# **使用SqlServer**

鉴于EF的灵活性，您可以使用任何EF支持的数据库。

# **数据库Schema更改和使用EF迁移**

IdentityServer4.EntityFramework包包含从IdentityServer的模型映射的实体类。

# **使用EF工具进行迁移**

我的做法：

新建两个DbContext继承PersistedGrantDbContext和ConfigurationDbContext

// IEPersistedGrantDbContext

public class IEPersistedGrantDbContext : PersistedGrantDbContext

{

public IEPersistedGrantDbContext(DbContextOptions<PersistedGrantDbContext> options, OperationalStoreOptions storeOptions) : base(options, storeOptions)

{

}

}

// IEConfigurationDbContext

public class IEConfigurationDbContext : ConfigurationDbContext

{

public IEConfigurationDbContext(DbContextOptions<ConfigurationDbContext> options, ConfigurationStoreOptions storeOptions) : base(options, storeOptions)

{

}

}

为了顺利迁移，还需建两个DbContext工厂供迁移工具迁移

// IEPersistedGrantDbContextFactory

public class IEPersistedGrantDbContextFactory : IDesignTimeDbContextFactory<IEPersistedGrantDbContext>

{

public IEPersistedGrantDbContext CreateDbContext(string[] args)

{

var builder = new DbContextOptionsBuilder<PersistedGrantDbContext>();

builder.UseSqlServer(“ConnectionString”);

return new IEPersistedGrantDbContext(builder.Options, new IdentityServer4.EntityFramework.Options.OperationalStoreOptions());

}

}

// IEConfigurationDbContextFactory

public class IEConfigurationDbContextFactory : IDesignTimeDbContextFactory<IEConfigurationDbContext>

{

public IEConfigurationDbContext CreateDbContext(string[] args)

{

var builder = new DbContextOptionsBuilder<ConfigurationDbContext>();

builder.UseSqlServer(“ConnectionString”);

return new IEConfigurationDbContext(builder.Options, new IdentityServer4.EntityFramework.Options.ConfigurationStoreOptions());

}

}

打开包管理控制台，开始迁移

Add-Migration InitialIdentityServerPersistedGrantDbMigration -c IEPersistedGrantDbContext -o Migrations/IdentityServer/PersistedGrantDb

Add-Migration InitialIdentityServerConfigurationDbMigration -c IEConfigurationDbContext -o Migrations/IdentityServer/ConfigurationDb

Update-Database -c IEPersistedGrantDbContext

Update-Database -c IEConfigurationDbContext

# **配置store**

下一步是在Startup.cs中ConfigureServices方法中的AddInMemoryClients，AddInMemoryIdentityResources和AddInMemoryApiResources进行替换。 我们将用这个代码替换它们：

**const** string connectionString = @"Data Source=(LocalDb)\MSSQLLocalDB;database=IdentityServer4.Quickstart.EntityFramework-2.0.0;trusted\_connection=yes;";

*// configure identity server with in-memory stores, keys, clients and scopes*

services.AddIdentityServer()

.AddDeveloperSigningCredential()

.AddTestUsers(Config.GetUsers())

*// this adds the config data from DB (clients, resources)*

.AddConfigurationStore(options =>

{

options.ConfigureDbContext = builder =>

builder.UseSqlServer(connectionString);

})

*// this adds the operational data from DB (codes, tokens, consents)*

.AddOperationalStore(options =>

{

options.ConfigureDbContext = builder =>

builder.UseSqlServer(connectionString);

*// this enables automatic token cleanup. this is optional.*

options.EnableTokenCleanup = **true**;

options.TokenCleanupInterval = 30;

});

# **初始化数据库**

为数据库设定种子。

在Startup.cs中添加此方法以帮助初始化数据库：

**private** **void** InitializeDatabase(IApplicationBuilder app){

**using** (var serviceScope = app.ApplicationServices.GetService<IServiceScopeFactory>().CreateScope())

{

serviceScope.ServiceProvider.GetRequiredService<PersistedGrantDbContext>().Database.Migrate();

var context = serviceScope.ServiceProvider.GetRequiredService<ConfigurationDbContext>();

context.Database.Migrate();

**if** (!context.Clients.Any())

{

**foreach** (var client **in** Config.GetClients())

{

context.Clients.Add(client.ToEntity());

}

context.SaveChanges();

}

**if** (!context.IdentityResources.Any())

{

**foreach** (var resource **in** Config.GetIdentityResources())

{

context.IdentityResources.Add(resource.ToEntity());

}

context.SaveChanges();

}

**if** (!context.ApiResources.Any())

{

**foreach** (var resource **in** Config.GetApiResources())

{

context.ApiResources.Add(resource.ToEntity());

}

context.SaveChanges();

}

}}

然后我们可以从Configure方法中调用它：

**public** **void** Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory){

*// this will do the initial DB population*

InitializeDatabase(app);

*// the rest of the code that was already here*

*// ...*}

# **运行程序**

您现在应该能够运行任何现有的客户端应用程序并登录，获取令牌并调用API - 全部基于数据库配置。