Escuelita Crombie 3er Edición

Dapper

Fuente

¿Qué es Dapper?

Dapper es una biblioteca de mapeo relacional de objetos (ORM) de código abierto para aplicaciones .NET y .NET Core. La biblioteca permite a los desarrolladores acceder de forma rápida y sencilla a los datos de las bases de datos sin necesidad de escribir código tedioso. Dapper permite ejecutar consultas SQL sin formato , mapear los resultados a objetos y ejecutar procedimientos almacenados , entre otras cosas. Está disponible como un paquete NuGet .

- **Dapper es liviano y rápido, lo que lo convierte en una opción ideal para aplicaciones que requieren baja latencia y alto rendimiento.**
- **Es una herramienta de mapeo de objetos simple pero poderosa para cualquier lenguaje .NET, como C#, que permite a los desarrolladores mapear rápida y fácilmente los resultados de consultas de los lectores de datos ADO.NET a instancias de objetos comerciales.**
- **Tiene un excelente soporte para consultas de bases de datos tanto asincrónicas como sincrónicas y para agrupar múltiples consultas en una sola llamada.**
- **Además, dapper admite consultas parametrizadas para ayudar a protegerse contra ataques de inyección SQL.**

NuGet a usar

```
Install-Package Microsoft.Data.SqlClient

Install-Package Dapper
```

¿Qué hace Dapper?

A continuación se muestra un código C# ADO.NET estándar para recuperar datos de una base de datos y materializarlos como una colección de `Product` objetos:

```
var sql = "SELECT * FROM products";
var products = new List<Product>();
using (var connection = new SqlConnection(connString))
{
    connection.Open();
    using (var command = new SqlCommand(sql, connection))
    {
        using (var reader = command.ExecuteReader())
        {
            var product = new Product
            {
                ProductID = reader.GetInt32(reader.GetOrdinal("ProductID")),
                ProductName = reader.GetString(reader.GetOrdinal("ProductName")),
                SupplierID = reader.GetInt32(reader.GetOrdinal("SupplierID")),
                CategoryID = reader.GetInt32(reader.GetOrdinal("CategoryID")),
                UnitPrice = reader.GetDecimal(reader.GetOrdinal("UnitPrice")),
                UnitsInStock = reader.GetInt32(reader.GetOrdinal("UnitsInStock")),
                UnitsOnOrder = reader.GetInt32(reader.GetOrdinal("UnitsOnOrder")),
                Discontinued = reader.GetBoolean(reader.GetOrdinal("Discontinued")),
                DiscontinuedDate = reader.GetDateTime(reader.GetOrdinal("DiscontinuedDate"))
            };
            products.Add(product);
        }
    }
}
```

En su nivel más básico, Dapper reemplaza el bloque de código de asignación resaltado en el ejemplo anterior con lo siguiente:

lenguaje-csharp

```
products = connection.Query<Product>(sql).ToList();
```

Consulta de datos con Dapper

Consultar una base de datos a través de Dapper le permite acceder rápida y fácilmente a sus datos con consultas simples.

- **Está diseñado para ser rápido, eficiente e intuitivo para que los usuarios puedan obtener rápidamente la información que necesitan de manera organizada.**
- **Los usuarios pueden ahorrar tiempo creando rápidamente consultas personalizadas según sus necesidades específicas.**
- **Las funciones fáciles de usar facilitan aprovechar al máximo su base de datos, lo que le permite concentrarse en otros aspectos de su proyecto.**

Consultar datos con Dapper es bastante sencillo. Solo tienes que proporcionar la consulta de selección y los parámetros de Dapper y, después, Dapper asignará automáticamente las columnas resultantes a sus propiedades correspondientes en tu tipo de modelo.

lenguaje-csharp

```
var sql = "SELECT * FROM Product WHERE CategoryID = @categoryID";
var products = connection.Query<Product>(sql, new { categoryID = 1}).ToList();
```

Ejecutar escalar de Dapper

Por ejemplo, si desea seleccionar la cantidad total de registros en una tabla, puede utilizar una consulta como `SELECT COUNT(*) FROM Products`. Luego, puede utilizar el `ExecuteScalar`método en la consulta para obtener un valor único que represente la cantidad total de registros.

Si desea realizar alguna operación en el valor devuelto, debe convertirlo a su tipo esperado. Como alternativa, utilice la versión de `ExecuteScalar<T>`que toma un parámetro genérico y especifica el tipo de retorno de forma explícita:

```
var sql = "SELECT COUNT(*) FROM Product";
var count = connection.ExecuteScalar<int>(sql);

Console.WriteLine($"Total products: {count}");
```

Consulta de una sola fila con Dapper

Dapper proporciona varios métodos para seleccionar una sola fila de datos, dependiendo de cómo desee trabajar con los datos que recupere.

Método	Descripción	Excepción
`QuerySingle`	Se utiliza cuando se espera que se devuelva solo una fila. Devuelve un `dynamic` tipo	`InvalidOperationException`, cuando la consulta devuelve cero o más de un elemento
`QuerySingle<T>`	Se utiliza cuando se espera que se devuelva solo una fila. Devuelve una instancia del tipo especificado por el `T` parámetro de tipo.	`InvalidOperationException`, cuando la consulta devuelve cero o más de un elemento
`QuerySingleOrDefault`	Se utiliza cuando se espera que se devuelva una o cero filas. Devuelve un `dynamic` tipo o `null`	`InvalidOperationException`, cuando la consulta devuelve más de un elemento
`QuerySingleOrDefault<T>`	Se utiliza cuando se espera que se devuelva una o cero filas. Devuelve una instancia del tipo especificado por el `T` parámetro de tipo o `null`	`InvalidOperationException`, cuando la consulta devuelve más de un elemento
`QueryFirst`	Devuelve la primera fila como un `dynamic` tipo	`InvalidOperationException`, cuando la consulta devuelve cero elementos
`QueryFirst<T>`	Devuelve la primera fila como una instancia del tipo especificado por el `T` parámetro de tipo	`InvalidOperationException`, cuando la consulta devuelve cero elementos
`QueryFirstOrDefault`	Devuelve la primera fila como un `dynamic` tipo o `null` si no se devuelven resultados	
`QueryFirstOrDefault<T>`	Devuelve la primera fila como una instancia del tipo especificado por el `T` parámetro de tipo o `null` si no se	

Dapper Query Single

Dapper permite a los desarrolladores seleccionar fácilmente una sola fila de datos de la base de datos mediante su `QuerySingle` método. Este método toma la consulta SQL y cualquier parámetro asociado como argumentos y devuelve una instancia del tipo especificado.

- **Los métodos `QuerySingle`y `QuerySingle<T>`están diseñados para usarse cuando se espera que solo una fila coincida con los criterios especificados en la consulta.**
- **La diferencia entre `QuerySingle`y `QuerySingle<T>`es que `QuerySingle`devuelve resultados como un `dynamic`tipo, mientras que `QuerySingle<T>`devuelve una instancia del tipo representado por `T`el argumento de tipo.**

lenguaje-csharp

```csharp
var sql = "SELECT * FROM Product WHERE ProductID = @productID";
var product = connection.QuerySingle(sql, new { productID = 1 });

Console.WriteLine($"ProductID: {product.ProductID}; Name: {product.Name}");
```

El siguiente ejemplo es idéntico al primero excepto por la presencia del `<Product>`parámetro de tipo:

```csharp
var sql = "SELECT * FROM Product WHERE ProductID = @productID";
var product = connection.QuerySingle<Product>(sql, new { productID = 1 });

Console.WriteLine($"ProductID: {product.ProductID}; Name: {product.Name}");
```

Dapper QueryFirst

Dapper `QueryFirst`le permite recopilar el primer resultado de una consulta. Esto resulta especialmente útil cuando solo necesita una fila de datos, como cuando realiza una consulta para obtener un valor de ID u otros resultados de una sola fila. QueryFirst puede aceptar cualquier tipo de consulta SQL y devolverá el resultado correspondiente.

```
var sql = "SELECT * FROM Product WHERE ProductID = @productID";
var product = connection.QueryFirst(sql, new { productID = 1 });
// var product = connection.QueryFirst<Product>(sql, new { productID = 1 });

Console.WriteLine($"ProductID: {product.ProductID}; Name: {product.Name}");
```

INSERT DAPPER

Para insertar en dapper, debe utilizar el método Execute con una `INSERT`declaración y proporcionar los valores de los parámetros de consulta.

En este ejemplo:

1. **Crearemos una conexión**
2. **Crearemos una `INSERT` sentencia sql**
3. **Llamar al `Execute` método**

- **3a. La primera vez, pasaremos valores de parámetros con un tipo anónimo.**
- **3b. La segunda vez, pasaremos valores de parámetros proporcionando la entidad del cliente.**

```
public void Insert()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        var sql = "INSERT INTO Productos (Name, Description, Stock, CreateDate) VALUES (@Name, @Description, @Stock, @CreateDate)";

        var anonymousProduct = new
        {
            Name = "Producto A",
            Description = "Descripción del Producto A",
            Stock = "100",
            CreateDate = DateTime.Now
        };
        var rowsAffected1 = connection.Execute(sql, anonymousProduct);

        var producto = new ProductoModel
        {
            Name = "Producto B",
            Description = "Descripción del Producto B",
            Stock = "200",
            CreateDate = DateTime.Now
        };
        var rowsAffected2 = connection.Execute(sql, producto);

        var insertedProductos = connection.Query<ProductoModel>("SELECT * FROM Productos").ToList();

    }
}
```