

# 第46期-上下文DbContext的生命周期

2019年7月10日 09:51

## 数据库连接管理

CanConnect、CanConnectAsync、GetDbConnection、OpenConnection、OpenConnectionAsync 和 CloseConnection方法。

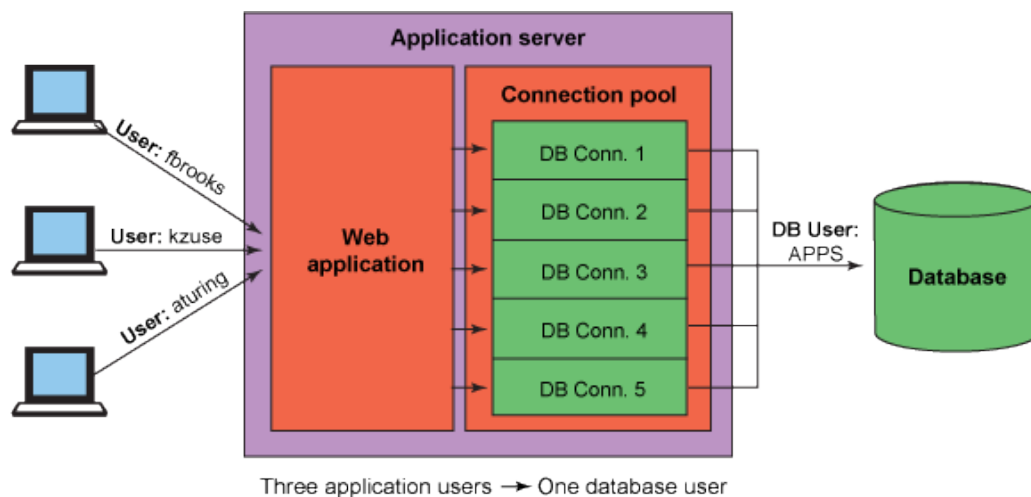
首先担心数据库连接没有释放肯定是多余的，因为DbContext在SaveChanges完成后会释放掉打开的数据库连接。

```
Console.WriteLine(_context.Database.GetDbConnection().State);
_context.Add(new Blog { Url = "www.xcode.me" });
_context.SaveChanges();
Console.WriteLine(_context.Database.GetDbConnection().State);
```

```
SELECT * FROM SYSPROCESSES WHERE DBID = DB_ID(' DatabaseName')
```

## 数据库连接池

连接到数据库可能需要很长时间，需要建立TCP七层模型建立物理连接，打开连接的成本较高，为了降低打开连接的成本，ADO.NET 使用名为连接池的技术，这样就可以将反复打开或关闭连接的成本降至最低。 .NET Framework 数据提供程序处理连接池的方式有所不同。



### SQL Server 连接池 (ADO.NET)

提供的连接池的概述和介绍连接池在SQL Server 中的工作方式。

### OLE DB、ODBC 和 Oracle 连接池

说明适用于OLE DB 的 .NET Framework 数据提供程序、适用于ODBC 的 .NET Framework 数据提供程序和适用于Oracle 的 .NET Framework 数据提供程序的连接池。

## 释放上下文相关的资源

DbContext.cs

using statement 和 IDisposable Interface

不要随意使用 Dispose 显式释放资源，这会导致延迟加载的不可用。

EF Core 中 DbContext 可以被 Dispose 多次，幂等操作，效果相同，调用 Dispose 方法后，就不能再用 DbContext 去操作数据了，除非重新 new 一个新的 DbContext 上下文，

**结论，您可以调用 Dispose 方法显式释放资源，但在大多数常见场景中并不需要。**

*Do I always have to call Dispose() on my DbContext objects? Nope*

来自 <<https://blog.jongallant.com/2012/10/do-i-have-to-call-dispose-on-dbcontext/>>

## 依赖注入生命周期

```
services.AddDbContext<BloggingContext>(options => options.UseSqlServer("connectionString"))
```

```
public enum ServiceLifetime { Singleton = 0, Scoped = 1, Transient = 2 }
```

Singleton：整个应用程序生命周期以内只创建一个实例。

Scoped：在同一个Scope内只初始化一个实例，可以理解为（每一个 request 级别只创建一个实例）

Transient：每一次 GetService 都会创建一个新的实例。

EntityFrameworkServiceCollectionExtensions.cs

## DbContext 不是线程安全的

如果在同一个 DbContext 上启动两个并行的操作，则会抛出异常，这是因为 DbContext 本身不能保证是线程安全，因此，同一时刻，一个上下文只能执行一个异步方法。