14

# Controle de transações com filtros

Como todo código que faz modificações no banco de dados deve ficar dentro de uma transação, os métodos dos DAOs que adicionam, modificam e removem entidades devem iniciar a transação, executar a modificação e depois fechar a transação:

```
ITransaction transacao = session.BeginTransaction();
// faz a modificação no banco de dados
transacao.Commit();
```

Estamos copiando esse código de controle de transação em todo o projeto, gostaríamos de abrir uma transação antes do processamento da requisição e, depois do processamento feito pelo controller, queremos fechar a transação que foi aberta.

Para executarmos um código antes e depois de uma action, podemos utilizar os action filters do Asp.Net MVC. Queremos criar um filtro global (na pasta Filters) que no método `OnActionExecuting` abre a transação e no `OnActionExecuted` fecha a transação e a sessão atual:

```
public class TransactionFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext contexto)
    {
        // começa a transação
    }
    public override void OnActionExecuted(ActionExecutedContext contexto)
    {
        // fecha a transação e a sessão
    }
}
```

Para que esse filtro consiga realizar seu trabalho, ele precisa da sessão atual do NHibernate, então vamos pedi-la no construtor da classe:

```
public class TransactionFilter : ActionFilterAttribute
{
    private ISession session;
    public TransactionFilter(ISession session)
    {
        this.session = session;
    }
}
```

Para abrirmos a transação, utilizamos o método `BeginTransaction` da session, porém para fecharmos a transação, precisamos da instância de `ITransaction`. Para pegarmos a transação associada com o `ISession`, utilizamos seu atributo `Transaction`:

```
public class TransactionFilter : ActionFilterAttribute
{
    private ISession session;
    public TransactionFilter(ISession session)
```

```
        {
            this.session = session;
        }
        public override void OnActionExecuting(ActionExecutingContext contexto)
        {
            session.BeginTransaction();
        }
        public override void OnActionExecuted(ActionExecutedContext contexto)
        {
            session.Transaction.Commit();
            session.Close();
        }
    }
```

Mas para que a sessão seja injetada no construtor do filtro, precisamos fazer com que ele seja registrado no Ninject, o plugin será responsável por registrar o filtro no Asp.Net MVC.

Vamos voltar novamente para a classe de configuração do Ninject, a `NinjectWebCommon`, que fica dentro da pasta App_Start. Dentro dessa classe registraremos os filtros no método `RegisterServices` já existente no final da classe. Para facilitar o registro dos filtros, utilizaremos um extension method do Ninject definido no namespace `Ninject.Web.Mvc.FilterBindingSyntax`. Vamos, portanto importar esse namespace:

```
using Ninject.Web.Mvc.FilterBindingSyntax;
```

Para registrar um filtro, utilizamos o extension method `BindFilter` do `IKernel` passando a ordem de execução do filtro e o escopo do filtro (escopo global para um filtro é sempre executado, por exemplo):

```
private static void RegisterServices(IKernel kernel)
{
    // registro dos outros componentes
    int ordemExecucao = 1;
    kernel.BindFilter<TransactionFilter>(FilterScope.Global, ordemExecucao);
}
```

Agora temos o controle de transações através de filtros do Asp.Net MVC!

Na pasta Filters do projeto, adicione o `TransactionFilter` implementado acima e remova o controle de transação dos métodos do DAO. Registre esse filtro dentro da classe `NinjectWebCommon`. Não se esqueça de testar a aplicação depois de fazer essa modificação.

**Responda**

| INSERIR CÓDIGO | | FORMATAÇÃO |
|---|---|---|

```
// cole o código do método RegisterServices() e da classe TransactionFilter
```

```
```