Israel Aece

Software Developer

PUBLICADO POR

ISRAEL AECE
POSTADO NO

14/06/2016
PUBLICADO EM

ARQUITETURA, CSD
COMENTÁRIOS

3 COMENTÁRIOS

Eventos de Domínio – Geração

i
3 Votes

Os eventos de domínio nos permite identificar ações importantes que ocorrem em nossa aplicação e que desejamos divulga-la para os interessados. Por interessados, leia-se outras aplicações ou, principalmente, outros contextos que estão interligados e que reagem aos eventos para realizar uma outra atividade relacionada aquela que acabou de acontecer. Para um exemplo simples, considere uma conta corrente que ao atingir o valor negativo, a central de risco do banco deve ser acionada para entender o que está havendo com o cliente e, eventualmente, monitorar as suas atividades financeiras para evitar um prejuízo maior.

O lançamento de débito ou crédito se dá na conta corrente, e se a regra de monitoramento for atendida, temos que passar a monitorar o respectivo cliente. Incorporar eventos à classe correspondente, que neste caso é a classe que representa a conta corrente, ajudará em uma centralização de código, fácil manutenção e, principalmente, agregando à ela a responsabilidade de

notificar que o saldo foi alterado (para cima ou para baixo). Competirá aos consumidores a usar a informação de acordo com a sua necessidade. O monitor de risco talvez não esteja interessado em uma conta que ficou "menos negativa".

```
1  public class ContaCorrente
2  {
3      //Outros membros ocultados
4
5      public void Lancar(Lancamento lancamento)
6      {
7          var saldoAnterior = this.Saldo;
8
9          this.lancamentos.Add(lancamento);
10         this.Saldo += lancamento.Valor;
11     }
12 }
```

Por agora, tudo o que o método acima faz é alteração da propriedade que armazena o saldo e inclui um novo lançamento na coleção interna. Depois do saldo alterado, chega o momento da conta corrente gerar o evento para notificar a alteração no saldo. A implementação padrão de eventos de domínio consiste na criação de uma *marker interface*, que geralmente não possui nenhum membro. Ao contrário do que acontece no .NET, onde os eventos são representados por *delegates*, no domínio utilizamos simples classes que implementam esta *interface*:

```
1  public interface IDomainEvent { }
```

A nomenclatura destas classes são sempre definidas no passado, que indicará que algo já ocorreu, por exemplo: *NovoPedidoAdicionado*, *NotaFiscalEmitida*, e para o nosso exemplo, *SaldoDaContaAlterado*. Vale lembrar que a nomenclatura deve expressar, e muito, exatamente o que ocorreu. E como já era de se esperar, todas as classes que representam eventos devem implementar a *interface IDomainEvent* que será útil para garantirmos a construção e uso dos tipos que envolvem a infraestrutura de eventos:

```
1  public class SaldoDaContaAlterado : IDomainEvent
2  {
3      public SaldoDaContaAlterado(
4          string nomeDoCliente, decimal saldoAnterior, decimal saldoAtual)
5      {
6          this.NomeDoCliente = nomeDoCliente;
7          this.SaldoAnterior = saldoAnterior;
8          this.SaldoAtual = saldoAtual;
9      }
10
11     public string NomeDoCliente { get; private set; }
12
13     public decimal SaldoAnterior { get; private set; }
14
15     public decimal SaldoAtual { get; private set; }
16 }
```

É importante notarmos que a classe que representa o evento possui algumas propriedades para descrever o que ocorreu, mas sempre temos que nos atentar em o que colocar ali, tentando manter a regra do mínimo possível necessário. Seguem algumas considerações importantes que devemos ter em mente durante a construção destas classes:

- **Entidades:** é tentador colocar nestas propriedades a própria entidade que sofreu a alteração (*ContaCorrente*). Devemos ao máximo evitar isso, pois causará uma dependência destas classes para os interessados ao evento. Muitas vezes os eventos serão utilizados para comunicação entre

contextos, e isso evitará a necessidade de referenciar a entidade (física e virtualmente). Tente optar sempre por tipos primitivos.

- **Imutabilidade:** essas classes não devem ter qualquer funcionalidade (métodos), apenas os dados que correspondem ao evento gerado. As propriedades são de somente leitura e são abastecidas no construtor.
- **Ids:** o Id da entidade pode ser definido no evento, mas isso só faz sentido se o consumidor já estiver colaborando com o gerador e tiver condições de recarregar a entidade, ou seja, ter acesso ao mesmo repositório. Caso o acesso não seja possível, teremos que incorporar na classe todas as informações necessárias para reportar a alteração, nem que seja necessário a duplicação de todas as informações da entidade.

Uma vez que os eventos já estão definidos, chega o momento de disparar. Como mencionado acima, o responsável pelo disparo será a própria classe que identifica a mudança, e que neste caso é a *ContaCorrente*, e sem qualquer análise de outra condição, dispara o evento informando que o saldo foi alterado. Note que ela irá recorrer à classe estática chamada de *DomainEvents*.

```
1  public class ContaCorrente
2  {
3      //Outros membros ocultados
4
5      public void Lancar(Lancamento lancamento)
6      {
7          var saldoAnterior = this.Saldo;
8
9          this.lancamentos.Add(lancamento);
10         this.Saldo += lancamento.Valor;
11
12         DomainEvents.Raise(
13             new SaldoDaContaAlterado(this.NomeDoCliente, saldoAnterior, thi
14     }
15 }
```

Agora é de responsabilidade da classe *DomainEvents* encontrar os consumidores deste tipo de evento e notifica-los da alteração. Há diversas técnicas que podemos utilizar na implementação da classe *DomainEvents* bem como em seus consumidores, mas que merece um artigo específico, e será abordado na sequência desta série.

- Código (https://israelaece.com/tag/codigo/)
- Domínio (https://israelaece.com/tag/dominio/)

3 comentários sobre “Eventos de Domínio – Geração”

1. Pingback: Eventos de Domínio – Disparo e Consumo | Israel Aece
2. Pingback: Eventos de Domínio – Outra Opção de Disparo | Israel Aece
3. Fabiano Nalin disse:
   10/09/2016 às 10:59
   Ótimo explicação!
   Abs

   Responder