Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
2021

A biblioteca *FireMapper* disponibiliza uma abstração sobre uma **colecção** de uma base de dados documental FireStore. Em modo resumido, o FireStore é uma base de dados NoSQL que armazena **colecções** de **documentos** JSON.

Uma base de dados FireStore é gerida a partir de um projecto FireBase.

O objectivo da biblioteca *FireMapper* é facilitar o acesso aos **documentos** de uma **colecção** por via de um `IDataMapper`. Esta interface especifica os métodos de acesso à **coleção** e que correspondem às operações CRUD.

```
public interface IDataMapper
{
    IEnumerable GetAll();
    object GetById(object keyValue);
    void Add(object obj);
    void Update(object obj);
    void Delete(object keyValue);
}
```

Por cada colecção deve existir uma **classe de domínio** com propriedades correspondentes às propriedades de um **documento**. Essas classes podem ter informação complementar dada na forma de anotações, por via dos seguintes *custom attributes*:

- `FireCollection` - aplicado a uma classe para identificar o nome da coleção Firestore.
- `FireKey` - identifica a propriedade que é chave única na pesquisa de um documento através do método `GetById`
- `FireIgnore` - propriedade a ignorar no mapeamento com um documento.

Exemplo:

```
[FireCollection("Students")]
public record Student(
    [property:FireKey] string Number,
    string Name,
    [property:FireIgnore] string Classroom)
{}
```

Students collection:

```
{
    “Name”: “Zanda Cantanda”,
    “Number”: “72538”,
    “Classroom”: “TLI41D”
}
…
```

A classe `FireDataMapper` implementa a interface `IDataMapper` com um comportamento dependente da classe de domínio (e.g. `Student`), cujo `Type` é fornecido na sua instanciação.

```
IDataMapper studentsMapper = new FireDataMapper(typeof(Student), ...);
```

A implementação de `FileDataMapper` é feita com o suporte da classe `FireDataSource` da biblioteca *FireSource* disponibilizado no respectivo projecto que integra a solução.

Enquanto a classe `FireDataSource` lida com dados fracamente tipificados na forma de `Dictionary<string, object>`, a classe `FireDataMapper` trata objectos de domínio, e.g. instâncias de `Student`.

1. C*ustom attributes* `FireCollection`, `FireKey` e `FireIgnore`.
2. A classe `FireDataMapper` faz o mapeamento entre objectos de domínio e dados na forma de `Dictionary<string, object>` manipulados por uma instância de `IDataSource`.
3. A classe `WeakDataSource` mantém os dados apenas em memória (defina a estrutura de dados ao seu critério).
4. A classe `FireDataMapper` pode funcionar com qualquer implementação de `IDataSource` especificada por parâmetro do construtor.

`FireDataMapper` suporta classes de domínio com propriedades de tipo definido por outras classes de domínio. Neste caso cria uma outra instância de `FireDataMapper` auxiliar para o respectivo tipo da propriedade que permite aceder à respectiva colecção.

Exemplo a classe `ClassroomInfo` correspondente ao tipo da propriedade `Classroom`:

```
[FireCollection("Students")]
public record Student( [property:FireKey] string Number, string Name, ClassroomInfo
Classroom)  {}

[FireCollection("Classrooms")]
public record ClassroomInfo([property:FireKey] string Token, string Teacher) {}
```

A classe `DynamicDataMapper` implementa `IDataMapper`, mas ao contrário de `FireDataMapper` **NÃO usa reflexão no acesso (leitura ou escrita) das propriedades das classes de domínio**. Note, que **continua a ser usada reflexão na consulta** da *metadata*, deixando apenas de ser usada reflexão em operações como `<property>.SetValue(…)` ou `<property>.GetValue(…)`.

O acesso a propriedades é realizado directamente com base em código IL emitido em tempo de execução através da API de `System.Reflection.Emit`.

Para tal, `DynamicDataMapper` gera em tempo de execução implementações, em que **cada tipo** implementa o acesso a uma determinada propriedade de uma classe de domínio.

`FireDataMapper` mantem uma estrutura de dados com um tipo de elemento (interface) que define a API de acesso a uma propriedade de uma classe de domínio.
Esta interface deve ter implementações diferentes, consoante represente o acesso a uma propriedade "simples" (*string* ou primitivo) ou "complexa" (do tipo de outra classe de domínio).

`DynamicFireMapper` que gera dinamicamente implementações, para cada propriedade de cada classe de domínio acedida.