06

Criando o DAO para o contato

Transcrição

Conseguimos adaptar o código do banco de dados para utilizar `async` `await`. Mas existe um outro detalhe bem importante que diz respeito à maneira como organizamos nosso programa e algumas **técnicas de refatoração**.

Até o momento, tudo está funcionando como esperado, e nossa solução atual está toda dentro de um único arquivo com poucos comportamentos e poucas configurações. Mas conforme o projeto evolui, é natural que adicionemos outras ações como salvar e buscar novas entidades por exemplo, fazendo com que o código cresça tornando sua manutenção cada vez mais complexa com lógica que não faz sentido ser compartilhada, e assim por diante.

Portanto, veremos como **separar responsabilidades** para **camadas mais específicas** nesta etapa, principalmente em relação ao código do banco de dados.

Um padrão comum em diversos projetos como Flutter é usar ***Data Access Object*** mais conhecido como **DAO**, ou **objeto de acesso aos dados**. Sua proposta é ser esta **camada** que mantém os comportamentos de uma entidade, como salvar, buscar e atualizar.

Então, criamos um DAO para nosso contato manter todas essas atividades, pois se no futuro começarmos a trabalhar com transferências por exemplo, este recurso será mantido pelo DAO também, evitando que cresça infinitamente.

Para começar a migração no IntelliJ IDEA, separamos o código em um pacote específico de um novo diretório chamado "dao" dentro da pasta "database" que contém todos os DAOs gerados neste momento e futuramente. Neste novo local, criamos um documento Dart chamado `contact_dao.dart`.

O primeiro passo para implementar este padrão no novo arquivo é criar uma nova classe `ContactDao` que recebe todos os comportamentos de `app_database.dart` relacionados ao contato, sendo `save()` e `findAll()`. Usamos a metodologia de minimizar os blocos de código para recortar e colar na nova classe, expandindo-os novamente.

Em seguida, importamos as bibliotecas `contact.dart` e `sqflite.dart` nas ocorrências em evidência e escrevemos manualmente no topo do texto o `import` de `app_database.dart`.

Como estamos fazendo a refatoração do código, podemos excluir os blocos comentados no passo anterior, limpando o texto. Mas é possível deixá-los por questões de estudo antes do projeto final.

Com este ajuste, temos alguns arquivos e referências que fazem uso de `save()` e `findAll()` a partir do arquivo único, e precisamos ajustá-los para que utilizem nosso DAO. Começando por `contact_form.dart`, adicionamos a dependência para cada uma das páginas através de um novo atributo `final` chamado de `ContactDao` na classe `_ContactFormState`. Em seguida, colocamos como `_dao` pois é o único presente nesta tela e a instância `ContactDao()`.

```
class _ContactFormState extends State<ContactForm> {
    final TextEditingController _nameController = TextEditingController();
    final TextEditingController _accountNumberController =
        TextEditingController();
    final Contact _dao = ContactDao();
```

Depois, adicionamos a referência `_dao` na linha onde chamamos o `save()` para compilar o código.

O próximo passo é aplicar a mesma metodologia da dependência em `contacts_list.dart` com `_dao` privado e `ContactDao()` como instância.

```
class ContactsList extends StatelessWidget {

    final Contact _dao = ContactDao();
```

Com o atributo inserido, chamamos o `findAll()` por meio de `_dao.` da mesma forma. Assim, nosso código volta a compilar novamente.

Feita a migração, a boa prática recomendada para evitar problemas futuros é testar no emulador com adição de novo item e buscando contatos.

Constatando o bom funcionamento da aplicação sem efeitos colaterais, podemos dar continuidade. Outro ponto importante diz respeito ao `app_database.dart`, onde retiramos os comentários para analisar com mais precisão o que devemos melhorar.

Na `String` que gera a tabela de contatos, percebemos o tipo de responsabilidade que o banco de dados não precisa ter, pois o próprio DAO pode fornecer este *script* para esta configuração, o que acontece naturalmente caso criemos novos bancos no futuro.

O próximo passo é recortar este script e colar no DAO, fornecendo por meio de uma constante estática para evitar criação de instâncias neste caso. Na classe `ContactDao`, inserimos `static final` que é uma `String` e indicamos o SQL da tabela com `tableSql` que consiste justamente no script recortado.

Adicionada a constante com `final`, existe uma outra maneira de declarar contantes através da chave **`const`**, a qual possui uma característica chamada **constante por meio do tempo de compilação**; é feita para valores que não mudam seu estado, como o caso de `String` e inteiros, por exemplo.

```
class ContactDao {

    static const String tableSql = 'CREATE TABLE contacts('
        'id INTEGER PRUMARY KEY, '
        'name, TEXT, '
        'account_number INTEGER)';
```

Não nos alongaremos em suas vantagens, mas ao longo dos exercícios do curso, artigos e documentação, é possível atestar os benefícios deste tipo de abordagem. No geral, se estamos trabalhando com `String`, inteiros e outros tipos primitivos, utilizamos `const` para ter uma melhor performance em relação ao `final`.

Com a tabela em DAO, a chamamos dentro de `db.execute()` no arquivo `app_database.dart` por meio de `ContactDao.tableSql`.

```
Future<Database> getDatabase() async {
    final String path = join(await getDatabasesPath(), 'bytebank.db');
    return openDatabase(
        path,
        onCreate (db, version) {
            db.execute(ContactDao.tableSql);
        },
```

```
        version: 1,
    );
}
```

Desta maneira, nosso banco de dados não possui nenhuma referência de contato, logo não faz mais sentido manter o `import` do pacote `contact.dart` no topo. A única coisa que precisamos saber é quem são os DAOs e quais queremos trabalhar, pedindo somente o script de criação da tabela.

Feita a refatoração, precisamos fazer melhorias dentro do próprio DAO. Uma delas é fazer extrações de código no caso da geração de mapa e geração de uma lista de contatos, por exemplo. Começamos pegando todo o código que cria o mapa e extraímos pelo atalho "Ctrl + Alt + M" para um novo método `_toMap()`, indicando o sentido de contatos para mapa.

```
Future<int> save(Contact contact) async {
    final Database db = await getDatabase();
    Map<String, dynamic> contactMap = _toMap(contact);
    return db.insert('contacts', contactMap);
}
```

Já no trecho onde fazemos a lista, podemos aplicar a extração para `_toList()`. Pegamos todo o bloco onde aparece a lista dentro de `findAll()` e usamos o atalho "Ctrl + Alt + M" para converter o código.

```
Future<List<Contact>> findAll() async {
    final Database db = await getDatabase();
    final List<Map<String, dynamic>> result = await db.query('contacts');
    List<Contact> contacts = _toList(result);
    return contacts;
}
```

Outro ponto a considerar diz respeito às `String` soltas que reutilizamos, como o nome da tabela e campos do formulário, fazendo sentido aplicar uma extração para evitar problemas de digitação para o caso de futuramente quisermos adicionar novos comportamentos que precisam de tabelas ou campos.

No arquivo `contact_dao.dart`, trabalhamos com **constantes privadas** pois não são interessantes de serem mantidas públicas, visto que o próprio DAO lida diretamente com elas. Então, adicionamos `static const String` à classe, começando pelo nome da tabela `tableName` que recebe `'contacts'`.

Também, podemos reutilizar o script para não errarmos aqui também, o que é muito importante. Para isso, usamos o `$` para `_tableName` no lugar de `contacts`.

```
class ContactDao {

    static const String tableSql = 'CREATE TABLE $_tableName('
        'id INTEGER PRIMARY KEY, '
        'name, TEXT, '
        'account_number INTEGER)';
    static const String _tableName = 'contacts';
```

Com isso, podemos fazer as substituições de `contacts` a seguir por `_tableName` tanto em `.query()` quanto em `.insert()`. Para finalizar, adotamos a mesma metodologia para cada um dos campos `id`, `name` e `account_number`.

Uma sugestão é copiar e colar os nomes dos campos nos valores para não errar a digitação. Em seguida, nos campos da tabela aplicamos `$` da mesma forma para a conversão.

```
class ContactDao {

    static const String tableSql = 'CREATE TABLE $_tableName('
        '$_id INTEGER PRUMARY KEY, '
        '$_name, TEXT, '
        '$_account_number INTEGER)';
    static const String _tableName = 'contacts';
    static const String _id = 'id';
    static const String _name = 'name';
    static const String _ accountNumber = 'account_number';
```

Em seguida, partimos para as funções utilitárias de conversão de mapa e lista em `_toMap()`.

```
Map<String, dynamic> _toMap(Contact contact) {
    final Map<String, dynamic> contactMap = Map();
    contactMap[_name] = contact.name;
    contactMap[_accountNumber] = contact.accountNumber;
    return contactMap;
}
```

Por fim, aplicamos a mesma alteração em `_toList()`.

```
List<Contact> _toList(List<Map<String, dynamic>> result) {
    final List<Contact> contacts = List();
    for (Map<String, dynamic> row in result) {
        final Contact contact = Contact(
            row[_id];
            row[_name];
            row[_accountNumber],
        );
        contacts.add(contact);
    }
    return contacts;
}
```

Desta forma, realizamos a conversão necessária fornecendo os comportamentos esperados e deixamos claros os elementos públicos com "Ctrl + Shift + seta para cima".

Agora, usamos o Dart Analysis para verificar alguns pontos que podemos limpar, como no caso de alguns `imports` com "Ctrl + Alt + O". Aproveitamos estes últimos momentos para também limpar comentários desnecessários e realizar extrações necessárias.

Por fim, falta-nos testar o código novamente no emulador e, se tudo estiver funcionando corretamente, nosso trabalho do curso está pronto.

O benefício de fazer a refatoração consiste na manutenção dos comportamentos e na separação de responsabilidades de DAO em relação à configuração do banco de dados, o que nos permite adicionar novas ações sem risco de efeitos colaterais e facilita a manutenção através do **padrão DAO**.