Fatec São José dos Campos Prof. Jessen Vidal

10 anos 2006-2016

Centro Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO

Curso Superior de Banco de Dados

Disciplina: Laboratório em Desenvolvimento de Banco de Dados V

Prof. Emanuel Mineda Carneiro

emanuel.mineda@fatec.sp.gov.br

São José dos Campos - SP

Roteiro

- Teste de Integração
- Banco de Dados para teste
- JUnit

Teste de Integração

Teste de Integração

- Testa várias unidades de software trabalhando em conjunto
  - O foco está em verificar a integração entre diferentes unidades de software
    - Cada unidade já se encontra individualmente testada
  - Automatizado
  - Caixa Preta – Se preocupa com a interface, verificando entradas, saídas e formatos

Banco de Dados para teste

Banco de Dados para teste

- Testes de integração fazem de uso de acesso a Banco de Dados e é desejável ter um destinado a esse propósito, com dados pré-cadastrados
  - Para testes costumamos utilizar um SGBD em memória, como o H2. Para utilizá-lo precisamos incluir sua dependência no "pom.xml"

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Banco de Dados para teste

- O próximo passo consiste em copiar o arquivo de configuração "application.properties" para a pasta "src/test/resources" e modificar a configuração de acesso ao Banco de Dados

```
## Logging
# Show sql statement
logging.level.org.hibernate.SQL=debug

# Show sql values
logging.level.org.hibernate.type.descriptor.sql=trace

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1
spring.datasource.username=sa
spring.datasource.password=sa

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = validate
```

Banco de Dados para teste

- Para criar o Banco de Dados, basta colocar um arquivo de script com o nome de "schema.sql" na mesma pasta ("src/test/resources")
  - Esse arquivo é executado antes de cada teste, portanto precisamos destruir ("drop") as tabelas antes de cada execução
  - O dialeto SQL a utilizar deve ser o do H2!

```
drop table if exists usr_usuario;

create table usr_usuario (
  usr_id bigint not null auto_increment,
  usr_nome varchar(20) not null,
  usr_senha varchar(100) not null,
  primary key (usr_id),
  unique (usr_nome)
);
```

- Da mesma forma, podemos povoar o Banco de Dados com um arquivo e nome "data.sql" na mesma pasta ("src/test/resources")

```
insert into usr_usuario (usr_nome, usr_senha)
    values ('admin',
'$2a$10$i3.Z8Yv1Fwl0I5SNjdCGkOTRGQjGvHjh/gMZhdc3e7LIovAklqM6C');
```

JUnit

JUnit

- Testes de integração não usam @MockBean, verificando como as classes se comportam em conjunto
  - Exemplo de teste para um Repository
    - Quando um novo usuário é cadastrado, o atributo "id" é preenchido automaticamente. Ou seja, não pode ser nulo ("assertNotNull")

```
@SpringBootTest
public class UsuarioRepositoryTest {

    @Autowired
    private UsuarioRepository usuarioRepo;

    @Test
    public void novoUsuarioTest() {
        Usuario usuario = new Usuario();
        usuario.setNome("UsuarioTeste");
        usuario.setSenha("123");
        usuario = usuarioRepo.save(usuario);
        assertNotNull(usuario.getId());
    }
}
```

Junit - Controller

- Testes de integração de controller funcionam de forma diferente, pois é necessário subir o sistema inteiro
  - Voltamos a utilizar a anotação de classe @SpringBootTest
  - A anotação @AutoConfigureMockMvc habilita o uso do MockMvc em um teste de integração
  - O uso do MockMvc é idêntico

```
@SpringBootTest
@AutoConfigureMockMvc
public class UsuarioControllerIntegrationTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void novoUsuarioTestOk() throws Exception {
        mvc.perform(post("/usuario")
            .content("{\"nome\":\"TesteMvc\", \"senha\":\"senha\"}")
            .contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").exists());
    }
}
```

Planejando um teste

Planejando um teste

- O planejamento ocorre de forma idêntica ao realizado no teste de unidade, contudo, em vez de criar "mocks", precisamos gerar dados de teste (tanto nos parâmetros de entrada quanto no Banco de Dados) que forcem a exploração de todo o código pelos testes

Dúvidas?