Spring ve Entegrasyon Birim Testleri

Entegrasyon Testleri ve Transaction Yönetimi

JAVA Eğitimleri

- **@Transactional** anotasyonu test sınıfı veya metot düzeyinde tanımlanabilir
- Container'da tanımlı bir **PlatformTransactionManager** bean mevcut olmalıdır
- Test metodu başladığında bir **TX başlatılır**
- Test metodu sonlandığında da bu TX **rollback edilir**
- **@Rollback(false)** ile TX **commit** ettirilebilir

Entegrasyon Testleri ve Transaction Yönetimi

JAVA Eğitimleri

```
@ContextConfiguration
@Transactional
public class TransactionalTests {
    @Test
    public void testWithRollback() {
        // ...
    }

    @Rollback(false)
    @Test
    public void testWithoutRollback() {
        // ...
    }
}
```

Entegrasyon Testleri ve Transaction Yönetimi

JAVA Eğitimleri

- **@TransactionConfiguration** ile sınıf düzeyinde TX konfigürasyonu özelleştirilebilir
- **@Before** ve **@After** metotları TX içinde çalıştırılır
- **@BeforeTransaction, @AfterTransaction** ile de TX dışında setup ve destroy işlemleri yapılabilir

Entegrasyon Testleri ve Transaction Yönetimi

JAVA Eğitimleri

```
@ContextConfiguration
@Transactional
@TransactionConfiguration(transactionManager="txMgr",
defaultRollback=false)
public class CustomConfiguredTransactionalTests {

    @Before
    public void setUp() {
    }

    @Rollback(true)
    @Test
    public void testProcessWithoutRollback() {
    }

    @After
    public void tearDown() {
    }
}
```

@Before anotasyonu ile işaretlenmiş setUp metodu TX içerisinde çalıştırılır. TX dışında çalışması için **@BeforeTransaction** kullanılmalıdır

@After anotasyonu ile işaretlenmiş tearDown metodu da TX içerisinde çalıştırılır. TX dışında çalışması için **@AfterTransaction** kullanılmalıdır

ORM Testleri ve False Positive

- Test metotları sonunda **TX rollback yapıldığı için** metot içerisinde gerçekleşen update veya delete gibi Hibernate/JPA işlemleri **DB'ye flush edilmeyecek**tir
- Dolayısı ile DB tarafında çalışan **update/delete SQL ifadesi olmayacağı için** DB düzeyinde varsa **constraint'ler vb kontrol edilmemiş** olacaktır
- Bu durumda da test metodu başarılı sonlanmış olsa bile **üretim ortamında hatalar** ortaya çıkabilecektir

ORM Testleri ve False Positive

```
@Test
@Transactional
public void updateWithSessionFlush() {
    updateEntityInHibernateSession();
    // yukarıdaki metot örneğin bir constraint
violation hatasına yol açsın
    // false positive durumundan sakınmak için
manual flush gerekir
    sessionFactory.getCurrentSession().flush();
}
```

- ORM işlemlerinden sonra, test metodu içerisinde **assertion'lardan hemen önce flush() çalıştırılmalı**dır

Web Uygulamalarının Test Edilmesi

- Spring standalone testler içerisinde **WebApplicationContext**'in oluşturulmasını sağlar
- Böylece **Controller katmanı** da entegrasyon testlerine dahil edilebilir

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration("webapp")
@ContextConfiguration("/appcontext/beans-*.xml")
public class SpringWebAppTests {
    ...
}
```

ContextRoot dizinini belirler
Default **src/main/webapp**'dır

Web Uygulamalarının Test Edilmesi

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration("webapp")
@ContextConfiguration("/appcontext/beans-*.xml")
public class SpringWebAppTests {

    @Autowired
    private WebApplicationContext wac; // cached

    @Autowired
    private MockServletContext servletContext; // cached

    @Autowired
    private MockHttpSession session;

    @Autowired
    private MockHttpServletRequest request;

    @Autowired
    private MockHttpServletResponse response;

}
```

Testler için oluşturulan WebApplicationContext nesnesi

Test metotları içerisinden erişilebilecek built-in mock nesnelerdir

Spring'in Servlet API Mock sınıflarıdır

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com