


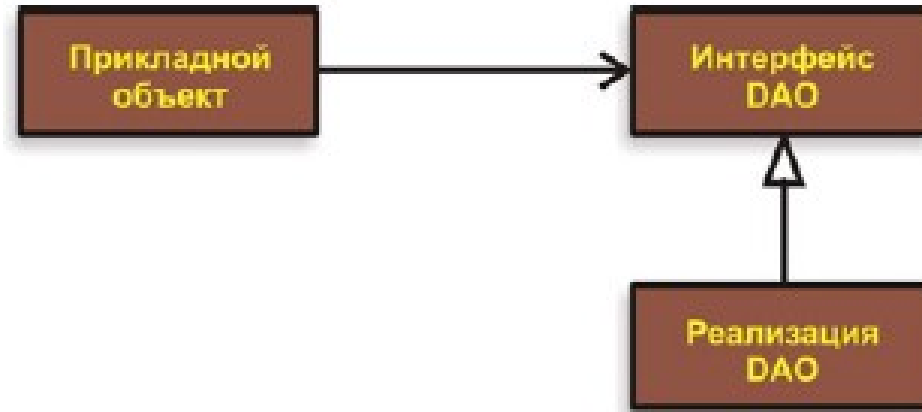
Фреймворк Spring. Работа с БД.

Составил: Черниговский А.С.
Старший преподаватель кафедры “Информатика”
ИКИТ СФУ



Философия доступа к данным Spring

- Data Access Object – объект доступа к данным



JDBC

- Java DataBase Connectivity (JDBC) — платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета `java.sql`, входящего в состав Java SE.

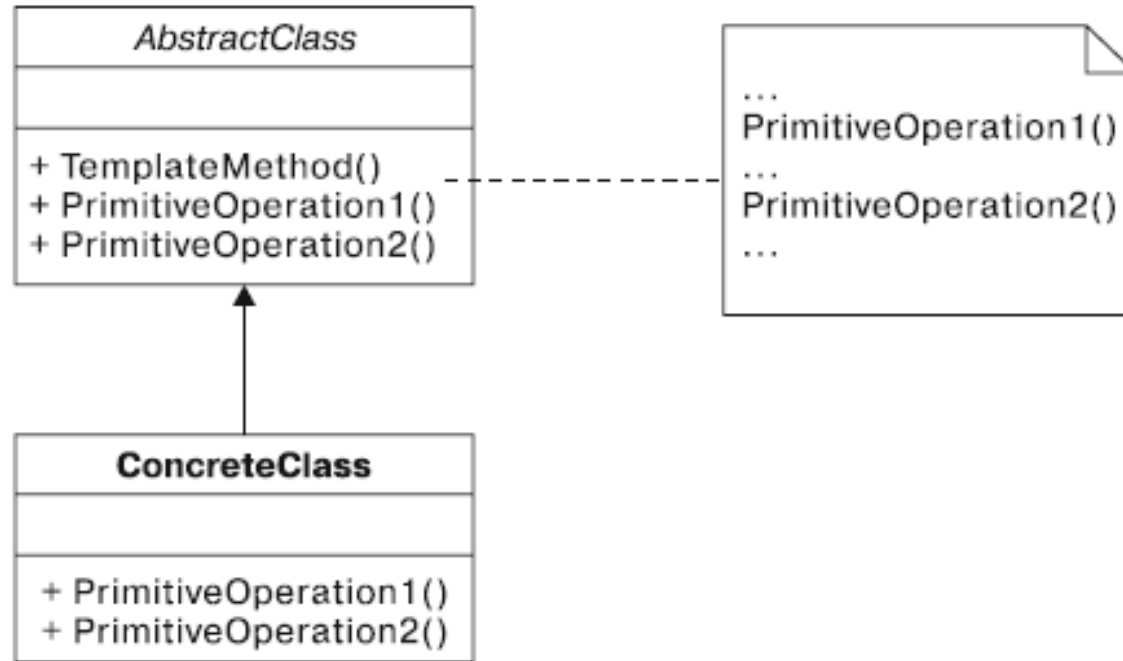
Исключение SQL Exception

- приложение не может подключиться к базе данных;
- выполняемый запрос имеет синтаксические ошибки;
- таблицы и/или столбцы, упомянутые в запросе, не существуют;
- попытка вставить или обновить значения нарушает ограничения базы данных.

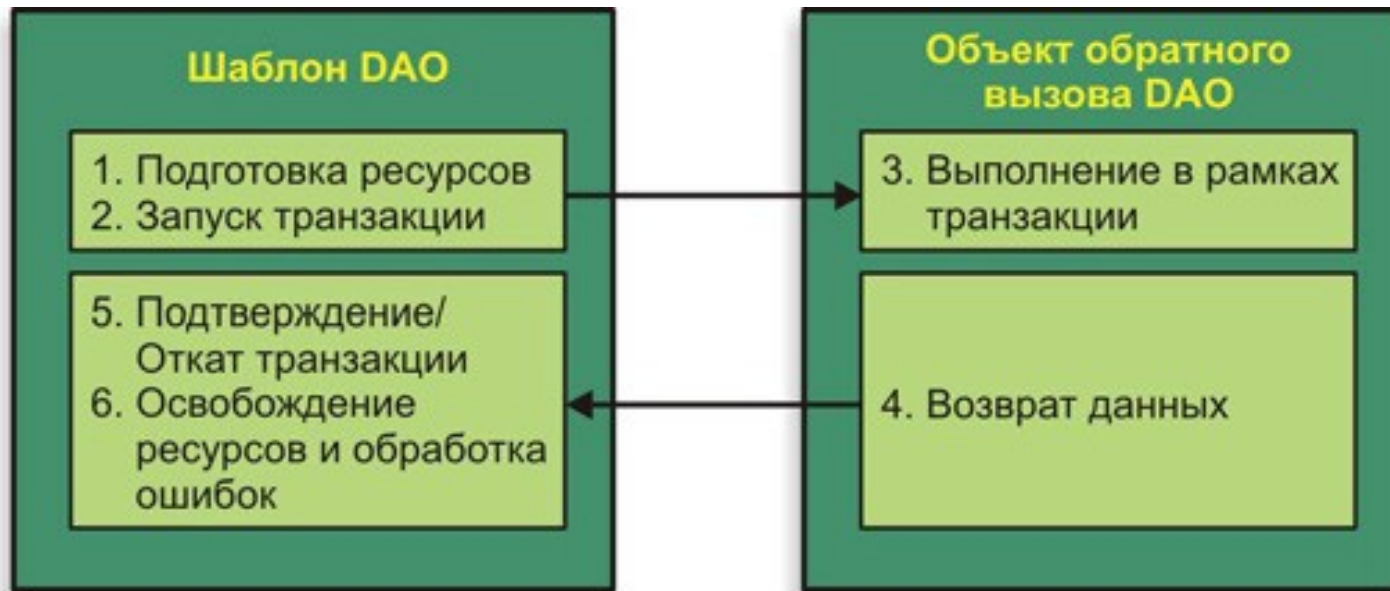
Универсальная иерархия исключений в Spring

Исключения JDBC	Исключения доступа к данным в Spring
BatchUpdateException DataTruncation SQLException SQLWarning	CannotAcquireLockException CannotSerializeTransactionException CleanupFailureDataAccessException ConcurrencyFailureException DataAccessException DataAccessResourceFailureException DataIntegrityViolationException DataRetrievalFailureException DeadlockLoserDataAccessException EmptyResultDataAccessException IncorrectResultSizeDataAccessException IncorrectUpdateSemanticsDataAccessException InvalidDataAccessApiUsageException InvalidDataAccessResourceUsageException OptimisticLockingFailureException PermissionDeniedDataAccessException PessimisticLockingFailureException TypeMismatchDataAccessException UncategorizedDataAccessException

Паттерн шаблонный метод



Шаблоны доступа к данным



Использование классов поддержки DAO



Источники данных JDBC

- DriverManagerDataSource – каждый раз, когда запрашивается соединение, возвращает новое соединение. В отличие от BasicDataSource в DBCP, соединения, предоставляемые DriverManagerDataSource, не объединяются в пул.
- SingleConnectionDataSource – каждый раз, когда запрашивается соединение, возвращает одно и то же соединение. Хотя SingleConnectionDataSource и не является пулом в полном смысле этого слова, тем не менее его можно воспринимать как источник данных с пулом, содержащим единственное соединение.

Java-конфигурация DataSource

```
@Autowired
private Environment env;

@Bean
DataSource dataSource() {
    DriverManagerDataSource dataSource = new
    DriverManagerDataSource();
    dataSource.setDriverClassName(env.getProperty("dataSource.driver
    ClassName"));
    dataSource.setUrl(env.getProperty("dataSource.url"));
    dataSource.setUsername(env.getProperty("dataSource.username"));
    dataSource.setPassword(env.getProperty("dataSource.password"));
    return dataSource;
}
```

Использование JDBC совместно со Spring

```
private static final String SQL_INSERT_STUDENT =  
"insert into student (id, first_name, last_name, patronymic, avg_mark) values  
(?, ?, ?, ?, ?)";
```

```
private DataSource dataSource;  
public void addStudent(Student student) {  
    Connection conn = null;  
    PreparedStatement stmt = null;  
    try  
    {  
        conn = dataSource.getConnection(); // Получить соединение  
        stmt = conn.prepareStatement(SQL_INSERT_STUDENT); // Создать запрос  
        stmt.setInt(1, student.getId()); // Связать параметры  
        // здесь были параметры  
        stmt.execute(); // Выполнить запрос  
    } catch (SQLException e) { // Обработать исключение (как-нибудь)  
        // выполнить что-нибудь... хотя... не уверен, что тут можно сделать  
    } finally {  
        try {  
            if (stmt != null) { // Освободить ресурсы  
                stmt.close();  
            }  
            if (conn != null) {  
                conn.close();  
            }  
        }  
    } catch (SQLException e) {  
        // Я еще менее уверен, что тут можно сделать  
    }  
}
```

Работа с шаблонами JDBC

- JdbcTemplate – самый основной шаблон JDBC, этот класс предоставляет простой доступ к базе данных через JDBC и простые запросы с индексированными параметрами;

JdbcTemplate

```
@Autowired  
public void setDataSource(DataSource dataSource){  
    this.jdbcTemplate = new JdbcTemplate(dataSource);  
}
```

Внедрение в DAO

```
@Component
public class StudentJdbcDao {

    JdbcTemplate jdbcTemplate;

    @Autowired
    public void setDataSource(DataSource dataSource){
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
}
```

addStudent В DAO

```
public int insert(Student student){  
    return jdbcTemplate.update("insert into student " + "(id,  
        first_name, last_name, patronymic, avg_mark) "  
        + "values (?, ?, ?, ?, ?)",  
        new Object[] {student.getId(),  
            student.getFirstName(),  
            student.getLastName(), student.getPatronymic(),  
            student.getAvgMark()});  
}
```

Интеграция Hibernate и Spring

- JDBC – это велосипед в мире технологий хранения данных. Он отлично подходит, для чего придуман, и в некоторых ситуациях просто великолепен. Но поскольку наши приложения становятся все более сложными, попробуем сформулировать наши новые требования к хранению данных.
- Фреймворк Spring включает поддержку некоторых фреймворков, реализующих хранение данных, включая Hibernate, iBATIS, Java Data Objects (JDO) и Java Persistence API (JPA). Как и в случае с поддержкой JDBC, Spring обеспечивает поддержку фреймворков ORM, предоставляя точки интеграции для фреймворков, а также некоторые дополнительные услуги, такие как:
 - интегрированная поддержка декларативных транзакций;
 - прозрачная обработка исключений;
 - легковесные классы шаблонов с поддержкой выполнения в многопоточной среде;
 - классы поддержки DAO;
 - управление ресурсами

Hibernate

- `HibernateTemplate`
- `org.hibernate.Session`

SessionFactory

```
@Bean
LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean localSessionFactoryBean = new
    LocalSessionFactoryBean();
    localSessionFactoryBean.setDataSource(dataSource());
    localSessionFactoryBean.setPackagesToScan("entity");
    Properties properties = new Properties();
    properties.setProperty("hibernate.dialect",
    env.getProperty("hibernate.dialect"));
    localSessionFactoryBean.setHibernateProperties(properties);
    return localSessionFactoryBean;
}
```

Создание классов для работы с Hibernate

```
@Repository
@Transactional
public class HibernateStudentDao {
    private SessionFactory sessionFactory;

    @Autowired
    public HibernateStudentDao(SessionFactory sessionFactory){
        this.sessionFactory = sessionFactory;
    }
    private Session currentSession() {                // Извлекает текущий
        return sessionFactory.getCurrentSession();    // сеанс из фабрики
                                                    // SessionFactory
    }
    public void addStudent(Student student) {
        currentSession().save(student);                // Использует текущий сеанс
    }
    public Student getStudentById(long id) {            // Использует текущий сеанс
        return (Student) currentSession().get(Student.class, id);
    }
    public void saveStudent(Student student) {
        currentSession().update(student);              // Использует текущий сеанс
    }
}
```

Spring и Java Persistence API

- С самого начала спецификация EJB включала понятие компонентов-сущностей (entity beans). В терминах EJB *компонент-сущность* представляет собой тип EJB, описывающий прикладные объекты, хранимые в реляционной базе данных. Компоненты-сущности претерпели несколько этапов развития на протяжении последних лет, включая появление компонентов-сущностей, которые *сами управляют своим сохранением* (bean-managed persistence, BMP), и компонентов-сущностей, *сохранением которых управляет контейнер* (container-managed persistence, CMP).

Настройка фабрики диспетчера сущностей

- Если говорить в двух словах, приложения на основе JPA используют реализацию EntityManagerFactory для получения экземпляра EntityManager. Спецификация JPA определяет два вида диспетчеров сущностей (entity managers).
- Управляемые приложением – эти диспетчеры сущностей создаются, когда приложение непосредственно запрашивает у фабрики диспетчеров сущностей. За создание и уничтожение диспетчеров сущностей, управляемых приложением, а также за их использование в транзакциях отвечает само приложение. Этот тип диспетчеров сущностей в большей степени подходит для использования в автономных приложениях, выполняющихся вне контейнера Java EE.
- Управляемые контейнером – эти диспетчеры сущностей создаются и управляются контейнером Java EE. Приложение никак не взаимодействует с фабрикой диспетчеров сущностей. Вместо этого диспетчеры сущностей приобретаются приложением посредством внедрения или из JNDI. За настройку фабрик диспетчеров сущностей отвечает контейнер.

Настройка JPA, управляемого контейнером

@Bean

```
public EntityManagerFactory entityManagerFactory() {  
    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
    vendorAdapter.setGenerateDdl(true);
```

```
    LocalContainerEntityManagerFactoryBean factory = new  
    LocalContainerEntityManagerFactoryBean();  
    factory.setJpaVendorAdapter(vendorAdapter);  
    factory.setPackagesToScan("ru.jdbctest");  
    factory.setDataSource(dataSource());  
    factory.afterPropertiesSet();
```

```
    return factory.getObject();  
}
```

@Bean

```
public PlatformTransactionManager transactionManager() {
```

```
    JpaTransactionManager txManager = new JpaTransactionManager();  
    txManager.setEntityManagerFactory(entityManagerFactory());  
    return txManager;
```

```
    }  
}
```

Репозиторий Spring Data

```
@Entity
public class Student {

    @Id
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    private String patronymic;

    @Column(name = "avg_mark")
    private double avgMark;
```

Репозиторий Spring Data

```
@Repository
public interface StudentRepository extends JpaRepository<Student,
Integer> {
    @Query("select s from Student s where s.id = ?1")
    Student findById(int id);
}
```

Query methods

```
List<Student> findByAvgMarkGreaterThan(double avgMark);
```

Спасибо за внимание!