Spring Transaction Propagation

caolei

Exported on 01/10/2020

Table of Contents

关于事务，想必大家都已经很了解了，经常提到的事务的特征(Atomicity, Consistency, Isolation, Durability)及隔离级别(Read_Uncommitted, Read_Committed, Repeatable_Read, Serializable)在这里就不赘述了，感兴趣的可以查阅数据库系统概念[1] 和 数据库系统概论[2].

Spring是Java语言最重要的轮子之一，所以对其掌握还是有些必要的，Spring中的事务还是有些许的复杂，今天我们只关注其一点-事务传播(Transaction Propagation)，通过代码演示来说明其工作原理。

1 http://product.dangdang.com/22632572.html
2 http://product.dangdang.com/25205830.html

1 Propagation源码

Propagation

```java
/**
 * Enumeration that represents transaction propagation behaviors for use
 * with the {@link Transactional} annotation, corresponding to the
 * {@link TransactionDefinition} interface.
 *
 * @author Colin Sampaleanu
 * @author Juergen Hoeller
 * @since 1.2
 */
public enum Propagation {

   /**
    * Support a current transaction, create a new one if none exists.
    * Analogous to EJB transaction attribute of the same name.
    * <p>This is the default setting of a transaction annotation.
    */
   REQUIRED(TransactionDefinition.PROPAGATION_REQUIRED),

   /**
    * Support a current transaction, execute non-transactionally if none exists.
    * Analogous to EJB transaction attribute of the same name.
    * <p>Note: For transaction managers with transaction synchronization,
    * {@code SUPPORTS} is slightly different from no transaction at all,
    * as it defines a transaction scope that synchronization will apply for.
    * As a consequence, the same resources (JDBC Connection, Hibernate Session, etc)
    * will be shared for the entire specified scope. Note that this depends on
    * the actual synchronization configuration of the transaction manager.
    * @see
org.springframework.transaction.support.AbstractPlatformTransactionManager#setTransactionSynchronization
    */
   SUPPORTS(TransactionDefinition.PROPAGATION_SUPPORTS),

   /**
    * Support a current transaction, throw an exception if none exists.
    * Analogous to EJB transaction attribute of the same name.
    */
   MANDATORY(TransactionDefinition.PROPAGATION_MANDATORY),

   /**
    * Create a new transaction, and suspend the current transaction if one exists.
    * Analogous to the EJB transaction attribute of the same name.
    * <p><b>NOTE:</b> Actual transaction suspension will not work out-of-the-box
    * on all transaction managers. This in particular applies to
    * {@link org.springframework.transaction.jta.JtaTransactionManager},
    * which requires the {@code javax.transaction.TransactionManager} to be
    * made available to it (which is server-specific in standard Java EE).
    * @see org.springframework.transaction.jta.JtaTransactionManager#setTransactionManager
    */
   REQUIRES_NEW(TransactionDefinition.PROPAGATION_REQUIRES_NEW),

   /**
    * Execute non-transactionally, suspend the current transaction if one exists.
    * Analogous to EJB transaction attribute of the same name.
    * <p><b>NOTE:</b> Actual transaction suspension will not work out-of-the-box
    * on all transaction managers. This in particular applies to
    * {@link org.springframework.transaction.jta.JtaTransactionManager},
    * which requires the {@code javax.transaction.TransactionManager} to be
    * made available to it (which is server-specific in standard Java EE).
    * @see org.springframework.transaction.jta.JtaTransactionManager#setTransactionManager
    */
   NOT_SUPPORTED(TransactionDefinition.PROPAGATION_NOT_SUPPORTED),

   /**
    * Execute non-transactionally, throw an exception if a transaction exists.
    * Analogous to EJB transaction attribute of the same name.
    */
   NEVER(TransactionDefinition.PROPAGATION_NEVER),

   /**
    * Execute within a nested transaction if a current transaction exists,
    * behave like {@code REQUIRED} otherwise. There is no analogous feature in EJB.
    * <p>Note: Actual creation of a nested transaction will only work on specific
    * transaction managers. Out of the box, this only applies to the JDBC
    * DataSourceTransactionManager. Some JTA providers might support nested
    * transactions as well.
    * @see org.springframework.jdbc.datasource.DataSourceTransactionManager
    */
   NESTED(TransactionDefinition.PROPAGATION_NESTED);


   private final int value;


   Propagation(int value) {
      this.value = value;
   }

   public int value() {
      return this.value;
   }

}
```

2 Propagation类型

通过源码我们可知，Spring中事务的传播机制有以下7种

传播机制	描述	备注
REQUIRED	支持当前事务，如果当前没有事务，就新建一个事务	默认机制
SUPPORTS	支持当前事务，如果当前没有事务，就以非事务方式执行	随意
MANDATORY	支持当前事务，如果当前没有事务，就抛出异常	相信当前事务
REQUIRES_NEW	新建事务，如果当前存在事务，把当前事务挂起	不相信当前事务
NOT_SUPPORTED	以非事务方式执行操作，如果当前存在事务，就把当前事务挂起	不愿意加入任何事务
NEVER	以非事务方式执行，如果当前存在事务，则抛出异常	不允许加入任何事务
NESTED	支持当前事务，如果当前事务存在，则执行一个嵌套事务，如果当前没有事务，就新建一个事务	事务嵌套

3 数据准备

3.1 User Entity(DataObject & DDL)

UserEntity

```java
@Data
@Entity
@Builder
@NoArgsConstructor(access = AccessLevel.PUBLIC)
@AllArgsConstructor(access = AccessLevel.PUBLIC)
@Table(schema = "practice", name = "users")
public class UserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", columnDefinition = "BIGINT", nullable = false)
    private Long id;

    @JsonFormat(timezone = "GMT+8", pattern = "yyyy-MM-dd HH:mm:ss")
    @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    @Column(name = "gmt_create", columnDefinition = "timestamp without time zone", nullable = false)
    private Date gmtCreate;

    @JsonFormat(timezone = "GMT+8", pattern = "yyyy-MM-dd HH:mm:ss")
    @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    @Column(name = "gmt_modify", columnDefinition = "TIMESTAMP WITHOUT TIME ZONE", nullable = false)
    private Date gmtModify;

    @Column(name = "user_name", columnDefinition = "CHARACTER VARYING(128)", nullable = false)
    private String userName;

    @Column(name = "user_email", columnDefinition = "CHARACTER VARYING(128)", nullable = false)
    private String userEmail;

    @Column(name = "user_age", columnDefinition = "SMALLINT", nullable = false)
    private Integer userAge;

    @Column(name = "user_gender", columnDefinition = "CHARACTER VARYING(16)", nullable = false)
    @Enumerated(EnumType.STRING)
    private GenderEnum userGender;

}
```

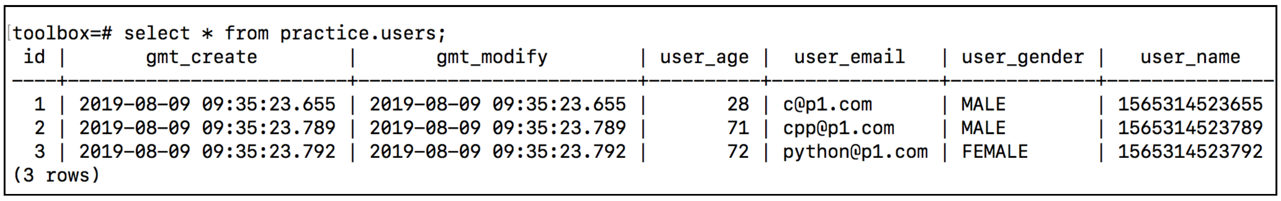
3.2 UserServiceImpl(CRUD on UserEntity)

UserServiceImpl

```java
/**
 * <p> describe in detail the functions of your class
 *
 * @version 1.0
 * @author: caolei
 */
@Slf4j
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    UserRepository userRepository;

    /**
     *
     * @author caolei
     * @Date 09:30 09/08/2019
     * @param
     * @return java.util.List<com.tantanapp.toolbox.entity.UserEntity>
     */
    @Override
    public List<UserEntity> findAll() {
        return userRepository.findAll();
    }

    /**
     *
     * @author caolei
     * @Date 09:30 09/08/2019
     * @param userEntity
     * @return com.tantanapp.toolbox.entity.UserEntity
     */
    @Override
    public UserEntity saveA(UserEntity userEntity) {
        return userRepository.save(userEntity);
    }

    /**
     *
     * @author caolei
     * @Date 09:30 09/08/2019
     * @param userEntity
     * @return com.tantanapp.toolbox.entity.UserEntity
     */
    @Override
    public UserEntity saveB(UserEntity userEntity) {
        return userRepository.save(userEntity);
    }

    /**
     *
     * @author caolei
     * @Date 09:31 09/08/2019
     * @param id
     * @return java.util.Optional<com.tantanapp.toolbox.entity.UserEntity>
     */
    @Override
    public Optional<UserEntity> findById(Long id) {
        return userRepository.findById(id);
    }
}
```

data preparation

```java
@Override
public void run(String... args) {
    UserEntity userEntity = buildUserEntity();
    UserEntity savedUserEntity = userService.saveA(userEntity);

    userEntity = buildUserEntity();
    savedUserEntity = userService.saveA(userEntity);

    userEntity = buildUserEntity();
    savedUserEntity = userService.saveA(userEntity);
}


/**
* 构建user实体对象
* @author caolei
* @Date 09:38 09/08/2019
* @param
* @return com.tantanapp.toolbox.entity.UserEntity
*/
private UserEntity buildUserEntity(){
    return UserEntity.builder()
            .gmtCreate(new Date())
            .gmtModify(new Date())
            .userName(String.valueOf(System.currentTimeMillis()))
            .userEmail(emails.get((new Random().nextInt(4))))
            .userAge((new Random().nextInt(100)))
            .userGender(System.currentTimeMillis() % 2 == 0 ? GenderEnum.FEMALE : GenderEnum.MALE)
            .build();
}
```



```
toolbox=# select * from practice.users;
 id |       gmt_create        |       gmt_modify        | user_age |  user_email   | user_gender |   user_name
----+-------------------------+-------------------------+----------+---------------+-------------+---------------
  1 | 2019-08-09 09:35:23.655 | 2019-08-09 09:35:23.655 |       28 | c@p1.com      | MALE        | 1565314523655
  2 | 2019-08-09 09:35:23.789 | 2019-08-09 09:35:23.789 |       71 | cpp@p1.com    | MALE        | 1565314523789
  3 | 2019-08-09 09:35:23.792 | 2019-08-09 09:35:23.792 |       72 | python@p1.com | FEMALE      | 1565314523792
(3 rows)
```

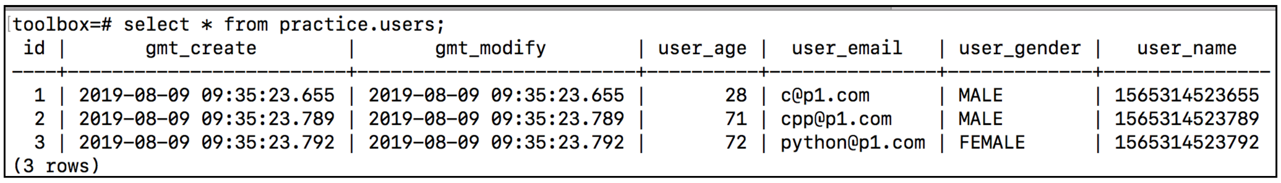
4 Propagation示例代码

接下来具体演示几个常用的事务传播机制

4.1 REQUIRED(TransactionDefinition.PROPAGATION_REQUIRED)

data preparation

```java
@Override
@Transactional(propagation = Propagation.REQUIRED)
public void run(String... args) {
    updateA();
    updateB();
}

/**
* update user with id equal 1
* @author caolei
* @Date 09:43 09/08/2019
* @param
* @return void
*/
public void updateA(){
    UserEntity userEntity = userService.findById(1L).get();
    userEntity.setUserName("UserA_No_Transactional");
    UserEntity savedUserEntity = userService.saveA(userEntity);
    log.info(savedUserEntity.getId().toString());
}

/**
* update user with id equal 2
* @author caolei
* @Date 09:43 09/08/2019
* @param
* @return void
*/
public void updateB(){
    UserEntity userEntity = userService.findById(2L).get();
    userEntity.setUserName("UserB_No_Transactional");
    UserEntity savedUserEntity = userService.saveB(userEntity);
    log.info(savedUserEntity.getId().toString());
    // throw RuntimeException，如果当前存在事务，事务就会回滚
    throw new RuntimeException("运行时异常，有事务吗？有的话就该回滚了");
}



/**
* 构建user实体对象
* @author caolei
* @Date 09:38 09/08/2019
* @param
* @return com.tantanapp.toolbox.entity.UserEntity
*/
private UserEntity buildUserEntity(){
        return UserEntity.builder()
                .gmtCreate(new Date())
                .gmtModify(new Date())
                .userName(String.valueOf(System.currentTimeMillis()))
                .userEmail(emails.get((new Random().nextInt(4))))
                .userAge((new Random().nextInt(100)))
                .userGender(System.currentTimeMillis() % 2 == 0 ? GenderEnum.FEMALE : GenderEnum.MALE)
                .build();
}
```



```
toolbox=# select * from practice.users;
 id |        gmt_create          |        gmt_modify          | user_age |  user_email   | user_gender |   user_name
----+----------------------------+----------------------------+----------+---------------+-------------+---------------
  1 | 2019-08-09 09:35:23.655 | 2019-08-09 09:35:23.655 |       28 | c@p1.com      | MALE        | 1565314523655
  2 | 2019-08-09 09:35:23.789 | 2019-08-09 09:35:23.789 |       71 | cpp@p1.com    | MALE        | 1565314523789
  3 | 2019-08-09 09:35:23.792 | 2019-08-09 09:35:23.792 |       72 | python@p1.com | FEMALE      | 1565314523792
(3 rows)
```

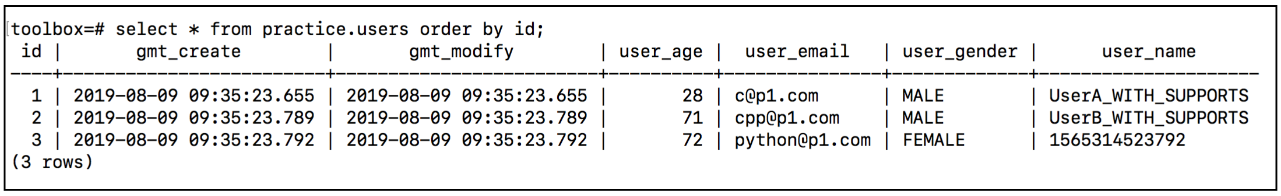
数据库中的数据未发生变更，原因是方法updateA和updateB虽然没有自己的事务，但是处于外层方法run的事务中，根据 REQUIRED的语义(支持当前事务，如有则加入，如果当前没有事务，就新建一个事务)，updateA和updateB都加入到run的事务中来，所以updateB方法抛出了RuntimeException，事务需要回滚，所以updateA也就跟随事务一起回滚了。

4.2 SUPPORTS(TransactionDefinition.PROPAGATION_SUPPORTS)

data preparation

```
@Override
public void run(String... args) {
    updateA();
    updateB();
}

/**
* update user with id equal 1
* @author caolei
* @Date 09:43 09/08/2019
* @param
* @return void
*/
@Transactional(propagation = Propagation.SUPPORTS)
public void updateA(){
    UserEntity userEntity = userService.findById(1L).get();
    userEntity.setUserName("UserA_WITH_SUPPORTS");
    UserEntity savedUserEntity = userService.saveA(userEntity);
    log.info(savedUserEntity.getId().toString());
}

/**
* update user with id equal 2
* @author caolei
* @Date 09:43 09/08/2019
* @param
* @return void
*/
@Transactional(propagation = Propagation.SUPPORTS)
public void updateB(){
    UserEntity userEntity = userService.findById(2L).get();
    userEntity.setUserName("UserB_WITH_SUPPORTS");
    UserEntity savedUserEntity = userService.saveB(userEntity);
    log.info(savedUserEntity.getId().toString());
    // throw RuntimeException，如果当前存在事务，事务就会回滚
    throw new RuntimeException("运行时异常，有事务吗？有的话就该回滚了");
}




/**
* 构建user实体对象
* @author caolei
* @Date 09:38 09/08/2019
* @param
* @return com.tantanapp.toolbox.entity.UserEntity
*/
```

```java
private UserEntity buildUserEntity(){
    return UserEntity.builder()
            .gmtCreate(new Date())
            .gmtModify(new Date())
            .userName(String.valueOf(System.currentTimeMillis()))
            .userEmail(emails.get((new Random().nextInt(4))))
            .userAge((new Random().nextInt(100)))
            .userGender(System.currentTimeMillis() % 2 == 0 ? GenderEnum.FEMALE : GenderEnum.MALE)
            .build();
}
```



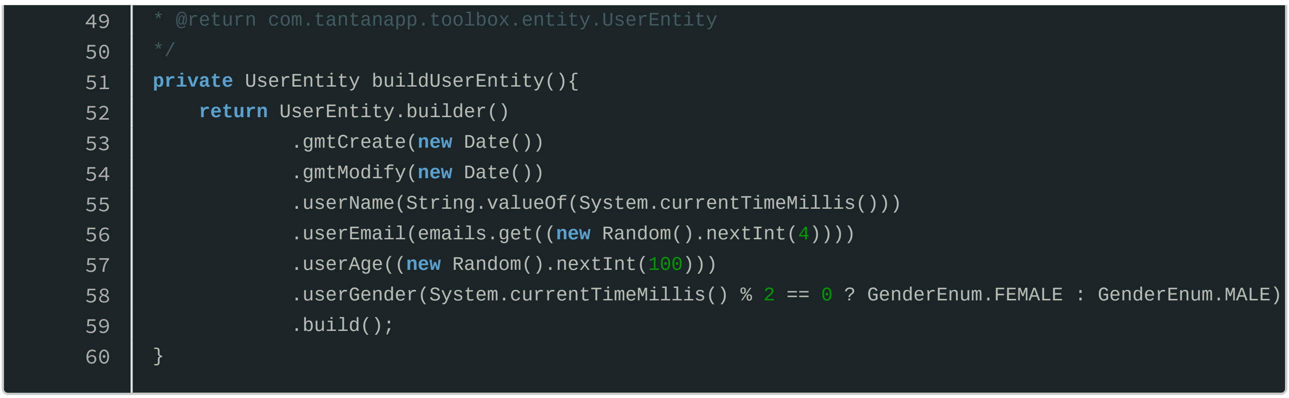
```
toolbox=# select * from practice.users order by id;
 id |         gmt_create         |         gmt_modify         | user_age |  user_email    | user_gender |      user_name
----+----------------------------+----------------------------+----------+----------------+-------------+---------------------
  1 | 2019-08-09 09:35:23.655 | 2019-08-09 09:35:23.655 |       28 | c@p1.com       | MALE        | UserA_WITH_SUPPORTS
  2 | 2019-08-09 09:35:23.789 | 2019-08-09 09:35:23.789 |       71 | cpp@p1.com     | MALE        | UserB_WITH_SUPPORTS
  3 | 2019-08-09 09:35:23.792 | 2019-08-09 09:35:23.792 |       72 | python@p1.com | FEMALE      | 1565314523792
(3 rows)
```

数据库中的id为1和2的数据发生变更，根据 SUPPORTS的语义(当前有事务则加入，没有则不添加事务)，run方法没有事务，updateA和updateB标注了SUPPORTS传播语义，所以最终updateA和updateB是没有事务的，集市updateB方法抛出了RuntimeException，因为没有事务，所以不需要回滚，最终数据中id为1和2的user_name被更新了。

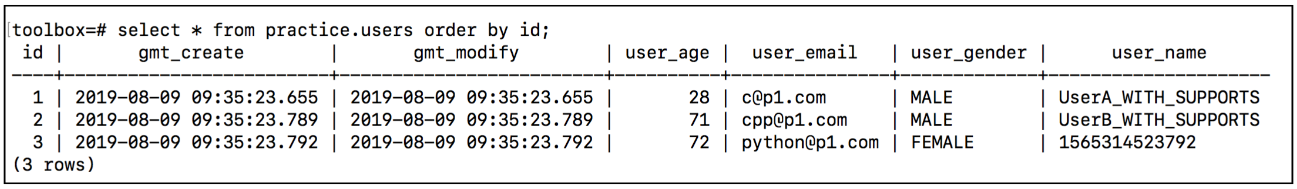
4.3 NEVER(TransactionDefinition.PROPAGATION_NEVER)

data preparation

```
@Transactional(propagation = Propagation.REQUIRED)
public void getUsers(HttpServletRequest request) {
    updateA();
    updateB();
}

/**
 * update user with id equal 1
 * @author caolei
 * @Date 09:43 09/08/2019
 * @param
 * @return void
 */
@Transactional(propagation = Propagation.NEVER)
public void updateA(){
    UserEntity userEntity = userService.findById(1L).get();
    userEntity.setUserName("UserA_WITH_NEVER");
    UserEntity savedUserEntity = userService.saveA(userEntity);
    log.info(savedUserEntity.getId().toString());
}

/**
 * update user with id equal 2
 * @author caolei
 * @Date 09:43 09/08/2019
 * @param
 * @return void
 */
@Transactional(propagation = Propagation.NEVER)
public void updateB(){
    UserEntity userEntity = userService.findById(2L).get();
    userEntity.setUserName("UserB_WITH_NEVER");
    UserEntity savedUserEntity = userService.saveB(userEntity);
    log.info(savedUserEntity.getId().toString());
    // throw RuntimeException，如果当前存在事务，事务就会回滚
    throw new RuntimeException("运行时异常，有事务吗？有的话就该回滚了");
}






/**
* 构建user实体对象
* @author caolei
* @Date 09:38 09/08/2019
* @param
```



```java
     * @return com.tantanapp.toolbox.entity.UserEntity
     */
    private UserEntity buildUserEntity(){
        return UserEntity.builder()
                .gmtCreate(new Date())
                .gmtModify(new Date())
                .userName(String.valueOf(System.currentTimeMillis()))
                .userEmail(emails.get((new Random().nextInt(4))))
                .userAge((new Random().nextInt(100)))
                .userGender(System.currentTimeMillis() % 2 == 0 ? GenderEnum.FEMALE : GenderEnum.MALE)
                .build();
    }
```



```
toolbox=# select * from practice.users order by id;
 id |         gmt_create         |         gmt_modify         | user_age |  user_email    | user_gender |      user_name
----+-------------------------+-------------------------+----------+----------------+-------------+---------------------
  1 | 2019-08-09 09:35:23.655 | 2019-08-09 09:35:23.655 |       28 | c@p1.com       | MALE        | UserA_WITH_SUPPORTS
  2 | 2019-08-09 09:35:23.789 | 2019-08-09 09:35:23.789 |       71 | cpp@p1.com     | MALE        | UserB_WITH_SUPPORTS
  3 | 2019-08-09 09:35:23.792 | 2019-08-09 09:35:23.792 |       72 | python@p1.com  | FEMALE      | 1565314523792
(3 rows)
```

Propagation.NEVER 的意思是以非事务的方式去运行，如果有事务，则抛出异常。上述代码中，getUsers是有事务的，但是 updateB 和 updateC 添加了 NEVER 的事务，意思是别给我加事务，我拒绝，你要是给我加了事务我就给你抛异常。