# 浪花的博客(最淡的墨水，也胜过最强的记忆)



## Hibernate的flush()和evict()

**博客分类：**
- hibernate

HibernateSQLOracleMySQLBlog
隔离级别  脏读 不可重复读 幻读

ReadUncommitted  Y Y Y
ReadCommitted N Y Y
RepeatableRead N N Y
Serializable N N N

ReadCommited是oracle的默认隔离级别。可以通过悲观锁，消除不可重复读。
RepeatableRead是Mysql的默认级别。

session flush方法主要做了两件事：
* 清理缓存
* 执行sql

session在什么情况下执行flush
* 默认在事务提交时
* 显示的调用flush
* 在执行查询前，如：iterate

hibernate按照save(insert),update、delete顺序提交相关操作
Java代码

```
1. /**
```

```
     * 测试uuid主键生成策略
     */
    public void testSave1() {
        Session session = null;
        Transaction tx = null;
        try {
            session = HibernateUtils.getSession();
            tx = session.beginTransaction();

            User1 user = new User1();
            user.setName("李四");
            user.setPassword("123");
            user.setCreateTime(new Date());
            user.setExpireTime(new Date());

            //因为user的主键生成侧路采用的是uuid，所以调用完成save后，只是将user纳入到了session的管理
            //不会发出insert语句，但是id已经生成，session中existsInDatebase状态为false
            session.save(user);

            //调用flush，hibernate会清理缓存，执行sql
            //如果数据库的隔离级别设置为为提交读，那么我们可以看到flush过的数据
            //并且session中existsInDatebase状态为true
            session.flush();

            //提交事务
            //默认情况下commit操作会先执行flush清理缓存，所以不用显示的调用flush
            //commit后数据是无法回滚的
            tx.commit();
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();
        }finally {
            HibernateUtils.closeSession(session);
        }
    }

    /**
     * 测试native主键生成策略
     */
    public void testSave2() {
        Session session = null;
        Transaction tx = null;
        try {
            session = HibernateUtils.getSession();
            tx = session.beginTransaction();

            User2 user = new User2();
            user.setName("张三1");
            user.setPassword("123");
```

```
            user.setCreateTime(new Date());
            user.setExpireTime(new Date());

            //因为user的主键生成策略为native,所以调用session.save后，将执行insert语句，返回有数据库生成的id
            //纳入了session的管理，修改了session中existsInDatebase状态为true
            //如果数据库的隔离级别设置为为提交读，那么我们可以看到save过的数据
            session.save(user);
            tx.commit();
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();
        }finally {
            HibernateUtils.closeSession(session);
        }
    }


    /**
     * 测试uuid主键生成策略
     */
    public void testSave3() {
        Session session = null;
        Transaction tx = null;
        try {
            session = HibernateUtils.getSession();
            tx = session.beginTransaction();

            User1 user = new User1();
            user.setName("王五");
            user.setPassword("123");
            user.setCreateTime(new Date());
            user.setExpireTime(new Date());

            //因为user的主键生成侧路采用的是uuid，所以调用完成save后，只是将user纳入到了session的管理
            //不会发出insert语句，但是id已经生成，session中existsInDatebase状态为false
            session.save(user);

            //将user对象从session中逐出，即session的EntityEntries属性中逐出
            session.evict(user);

            //无法成功提交，因为hibernate在清理缓存时，在session的insertions集合中取出user对象进行insert操作后
            //需要更新entityEntries属性中的existsInDatabase为true，而我们采用evict已经将user从session的entityEntries
            //中逐出了，所以找不到相关数据，无法更新，抛出异常
            tx.commit();
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();
        }finally {
            HibernateUtils.closeSession(session);
```

```
100.        }
101.    }
102.
103.    /**
104.     * 测试uuid主键生成策略
105.     */
106.    public void testSave4() {
107.        Session session = null;
108.        Transaction tx = null;
109.        try {
110.            session = HibernateUtils.getSession();
111.            tx = session.beginTransaction();
112.
113.            User1 user = new User1();
114.            user.setName("王五");
115.            user.setPassword("123");
116.            user.setCreateTime(new Date());
117.            user.setExpireTime(new Date());
118.
119.            //因为user的主键生成侧路采用的是uuid，所以调用完成save后，只是将user纳入到了session的管理
120.            //不会发出insert语句，但是id已经生成，session中existsInDatebase状态为false
121.            session.save(user);
122.
123.            //flush后hibernate会清理缓存，会将user对象保存到数据库中，将session中的insertions中的user对象
124.            //清除，并且设置session中existsInDatebase的状态为true
125.            session.flush();
126.
127.            //将user对象从session中逐出，即session的EntityEntries属性中逐出
128.            session.evict(user);
129.
130.            //可以成功提交，因为hibernate在清理缓存时，在session的insertions集合中无法找到user对象
131.            //所以就不会发出insert语句，也不会更新session中的existsInDatabase的状态
132.            tx.commit();
133.        }catch(Exception e) {
134.            e.printStackTrace();
135.            tx.rollback();
136.        }finally {
137.            HibernateUtils.closeSession(session);
138.        }
139.    }
140.
141.    /**
142.     * 测试native主键生成策略
143.     */
144.    public void testSave5() {
145.        Session session = null;
146.        Transaction tx = null;
147.        try {
148.            session = HibernateUtils.getSession();
```

```
            tx = session.beginTransaction();

            User2 user = new User2();
            user.setName("张三11");
            user.setPassword("123");
            user.setCreateTime(new Date());
            user.setExpireTime(new Date());

            //因为user的主键生成策略为native,所以调用session.save后，将执行insert语句，返回有数据库生成的id
            //纳入了session的管理，修改了session中existsInDatebase状态为true
            //如果数据库的隔离级别设置为为提交读，那么我们可以看到save过的数据
            session.save(user);

            //将user对象从session中逐出，即session的EntityEntries属性中逐出
            session.evict(user);

            //可以成功提交，因为hibernate在清理缓存时，在session的insertions集合中无法找到user对象
            //所以就不会发出insert语句，也不会更新session中的existsInDatabase的状态
            tx.commit();
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();
        }finally {
            HibernateUtils.closeSession(session);
        }
    }

    /**
     * 测试assigned主键生成策略
     *
     */
    public void testSave6() {
        Session session = null;
        Transaction tx = null;
        try {
            session = HibernateUtils.getSession();
            tx = session.beginTransaction();

            User3 user = new User3();
            user.setId("001");
            user.setName("张三");

            session.save(user);

            user.setName("王五");
            session.update(user);

            User3 user3 = new User3();
            user3.setId("002");
```

```
            user3.setName("李四");
            session.save(user3);

            //Hibernate: insert into t_user3 (name, password, create_time, expire_time, user_id) values (?, ?, ?, ?, ?)
            //Hibernate: insert into t_user3 (name, password, create_time, expire_time, user_id) values (?, ?, ?, ?, ?)
            //Hibernate: update t_user3 set name=?, password=?, create_time=?, expire_time=? where user_id=?
            //hibernate按照save(insert),update、delete顺序提交相关操作
            tx.commit();
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();
        }finally {
            HibernateUtils.closeSession(session);
        }
    }

    /**
     * 测试assigned主键生成策略
     *
     */
    public void testSave7() {
        Session session = null;
        Transaction tx = null;
        try {
            session = HibernateUtils.getSession();
            tx = session.beginTransaction();

            User3 user = new User3();
            user.setId("003");
            user.setName("张三");

            session.save(user);

            user.setName("王五");
            session.update(user);

            session.flush();

            User3 user3 = new User3();
            user3.setId("004");
            user3.setName("李四");
            session.save(user3);

            //Hibernate: insert into t_user3 (name, password, create_time, expire_time, user_id) values (?, ?, ?, ?, ?)
            //Hibernate: update t_user3 set name=?, password=?, create_time=?, expire_time=? where user_id=?
            //Hibernate: insert into t_user3 (name, password, create_time, expire_time, user_id) values (?, ?, ?, ?, ?)
            //因为我们在session.udpate(user)后执行了flush，所以在清理缓存时执行flush前的sql不会生成
            //sql会按照我们的意愿执行
            tx.commit();
```

```
247.         }catch(Exception e) {
248.             e.printStackTrace();
249.             tx.rollback();
250.         }finally {
251.             HibernateUtils.closeSession(session);
252.         }
253.     }
```

Hibernate的flush机制:http://sind.iteye.com/blog/255429



分享到：


hibernate配置文件的一些属性 | hibernate中get方法和load方法的区别


- 2009-03-05 01:49
- 浏览 7312
- 评论(0)
- 查看更多

**相关资源推荐**


hibernate session中clear、evict、flush方法的区别
Hibernate中get和load方法的区别以及close()、clear()、evict()
Hibernate学习--- Session.evict()方法
Hibernate 中的session 的flush、reflush 和clear 方法 ，及数据库的隔离级别
Hibernate一级缓存管理-evict和clear的用法
hibernate中的session.flush()和commit()的区别
Hibernate的clear(),flush(),evict()方法详解
hibernate的事务处理机制以及flush方法的作用
hibernate中的悲观锁_乐观锁解决事物并发的问题
springMVC整合hibernate的时候数据插入需要flush问题
关于flush和evict
org.hibernate.Session.evict(Object object)方法的使用
Hibernate clear(), flush(), evict()区别
hibernate中flush()、refresh()、clear()缓存操作
Hibernate Clear 与 Flush 方法
Hibernate 的clear()、flush()、evict(obj)使用
Hibernate--clear(),flush(),evict()
Hibernate的evict()方法和clear()方法、flush()方法
Hibernate中Session的flush方法介绍
spring管理的hibernate事务不会自动flush的问题-今天真遇到这问题了


**评论**

**发表评论**