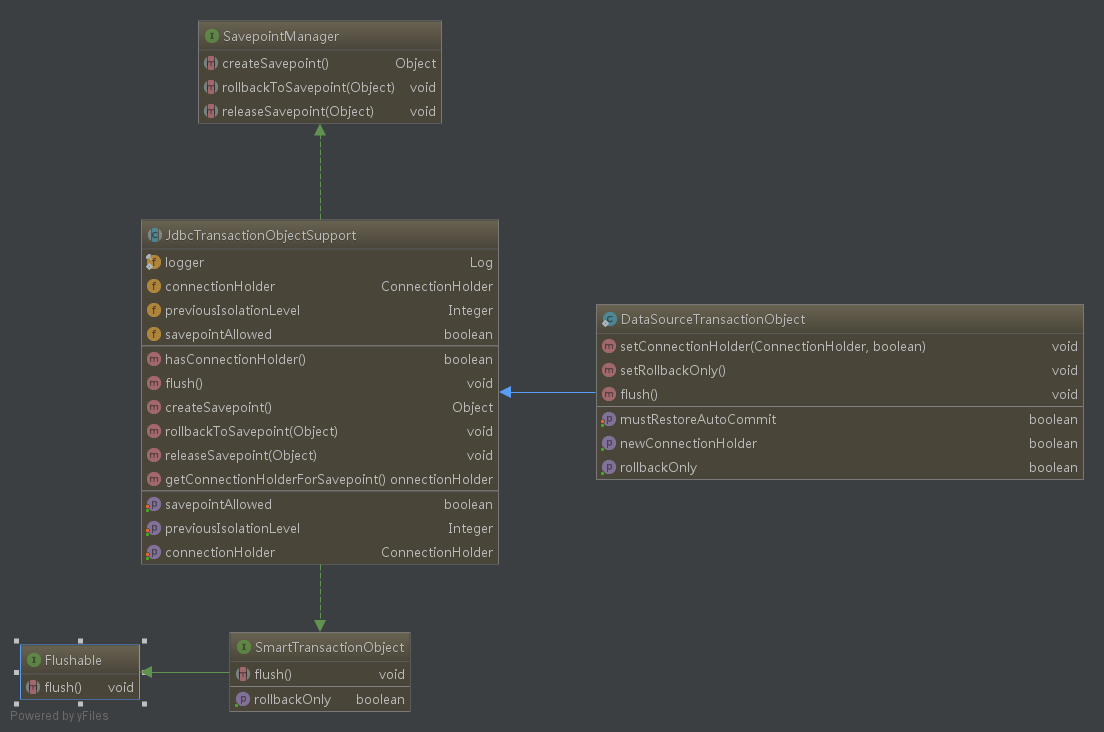
**Spring事务之DataSourceTransactionObject**

**类图DataSourceTransactionObject**



**1. 接口SavepointManager**

*/\*\*  
 \* 创建一个新的保存点  
 \*/*Object createSavepoint() throws TransactionException;  
  
*/\*\*  
 \* 回滚到某一个保存点  
 \*/*void rollbackToSavepoint(Object savepoint) throws TransactionException;  
  
*/\*\*  
 \* 释放保存点  
 \*/*void releaseSavepoint(Object savepoint) throws TransactionException;

**2. 抽象类JdbcTransactionObjectSupport**

定义了几个类成员

private ConnectionHolder connectionHolder;  
private Integer previousIsolationLevel;  
private boolean savepointAllowed = false;

实现了接口SavepointManager三个方法，

@Override  
public Object createSavepoint() throws TransactionException {  
 ConnectionHolder conHolder = getConnectionHolderForSavepoint();  
 try {  
 if (!conHolder.supportsSavepoints()) {  
 throw new NestedTransactionNotSupportedException(  
 "Cannot create a nested transaction because savepoints are not supported by your JDBC driver");  
 }  
 if (conHolder.isRollbackOnly()) {  
 throw new CannotCreateTransactionException(  
 "Cannot create savepoint for transaction which is already marked as rollback-only");  
 }  
 return conHolder.createSavepoint();  
 }  
 catch (SQLException ex) {  
 throw new CannotCreateTransactionException("Could not create JDBC savepoint", ex);  
 }  
}  
@Override  
public void rollbackToSavepoint(Object savepoint) throws TransactionException {  
 ConnectionHolder conHolder = getConnectionHolderForSavepoint();  
 try {  
 conHolder.getConnection().rollback((Savepoint) savepoint);  
 conHolder.resetRollbackOnly();  
 }  
 catch (Throwable ex) {  
 throw new TransactionSystemException("Could not roll back to JDBC savepoint", ex);  
 }  
}  
@Override  
public void releaseSavepoint(Object savepoint) throws TransactionException {  
 ConnectionHolder conHolder = getConnectionHolderForSavepoint();  
 try {  
 conHolder.getConnection().releaseSavepoint((Savepoint) savepoint);  
 }  
 catch (Throwable ex) {  
 *logger*.debug("Could not explicitly release JDBC savepoint", ex);  
 }  
}

**2. 类DataSourceTransactionObject**

定义了两个私有成员

private boolean newConnectionHolder;  
private boolean mustRestoreAutoCommit;

覆写了两个方法成员，

//仅回滚状态的事务

@Override  
public boolean isRollbackOnly() {  
 return getConnectionHolder().isRollbackOnly();  
}

//执行TransactionSynchronization中flush方法  
@Override  
public void flush() {  
 if (TransactionSynchronizationManager.*isSynchronizationActive*()) {  
 TransactionSynchronizationUtils.*triggerFlush*();  
 }  
}