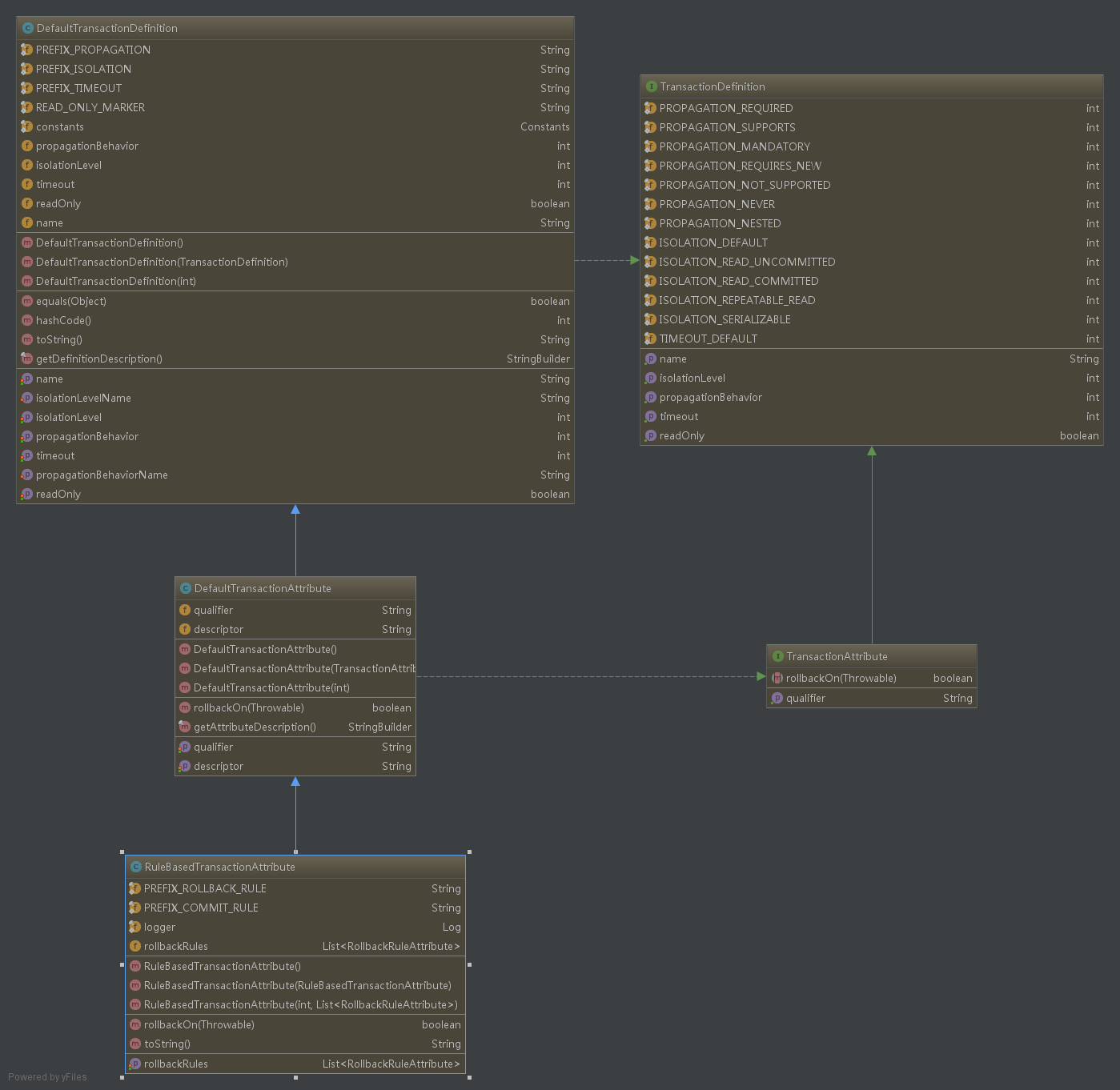
**Spring事务之RuleBasedTransactionAttribute**

**RuleBasedTransactionAttribute类图**



**1. 接口TransactionDefinition**

该接口定义了事务传播规则、事务隔离级别、超时时间、只读事务。

**【事务传播规则】**

int *PROPAGATION\_REQUIRED* = 0;  
int *PROPAGATION\_SUPPORTS* = 1;  
int *PROPAGATION\_MANDATORY* = 2;  
int *PROPAGATION\_REQUIRES\_NEW* = 3;  
int *PROPAGATION\_NOT\_SUPPORTED* = 4;int *PROPAGATION\_NEVER* = 5;int *PROPAGATION\_NESTED* = 6;

**【事务隔离级别】**

int *ISOLATION\_DEFAULT* = -1;int *ISOLATION\_READ\_UNCOMMITTED* = Connection.*TRANSACTION\_READ\_UNCOMMITTED*;int *ISOLATION\_READ\_COMMITTED* = Connection.*TRANSACTION\_READ\_COMMITTED*;int *ISOLATION\_REPEATABLE\_READ* = Connection.*TRANSACTION\_REPEATABLE\_READ*;int *ISOLATION\_SERIALIZABLE* = Connection.*TRANSACTION\_SERIALIZABLE*;

**【超时时间】**

int *TIMEOUT\_DEFAULT* = -1;

**【只读事务】**

boolean isReadOnly();

**【事务名称】**

String getName();

**2. 类DefaultTransactionDefinition**

类DefaultTransactionDefinition是TransactionDefinition的实现，简单的实现了一些get和set方法。下面是构造方法，

*/\*\*  
 \* 构造器  
 \*/*public DefaultTransactionDefinition(TransactionDefinition other) {  
 this.propagationBehavior = other.getPropagationBehavior();  
 this.isolationLevel = other.getIsolationLevel();  
 this.timeout = other.getTimeout();  
 this.readOnly = other.isReadOnly();  
 this.name = other.getName();  
}

**3. 接口TransactionAttribute**

该接口实现了TransactionDefinition，另外定义了下列两个方法。

*/\*\*  
 \* 候选备用，有可能使用 qualifier 来获取事务管理器。  
 \*/*String getQualifier();  
  
*/\*\*  
 \* 判断发生的异常是否需要进行事务回滚。   
 \*/*boolean rollbackOn(Throwable ex);

**4. 类DefaultTransactionAttribute**

改类实现了接口TransactionAttribute，继承了类DefaultTransactionDefinition，主要增加了下面方法。

*/\*\*  
 \* 如果是RuntimeException或者Error及其子类，都会认为当前发生的异常需要回滚。  
 \*/*@Override  
public boolean rollbackOn(Throwable ex) {  
 return (ex instanceof RuntimeException || ex instanceof Error);  
}

**5. 类RuleBasedTransactionAttribute**

类RuleBasedTransactionAttribute集成了类DefaultTransactionAttribute。

增加了私有成员，

private List<RollbackRuleAttribute> rollbackRules;

覆写了方法，

*/\*\*  
 \* 根据发生的异常，判断是否要回滚事务。true：需要回滚；false：不需要回滚。*

*\* 由于rollbackRules是一个list，寻找到最深的那个回滚规则。*

*\*/*@Override  
public boolean rollbackOn(Throwable ex) {  
 if (*logger*.isTraceEnabled()) {  
 *logger*.trace("Applying rules to determine whether transaction should rollback on " + ex);  
 }  
  
 RollbackRuleAttribute winner = null;  
 int deepest = Integer.*MAX\_VALUE*;  
  
 if (this.rollbackRules != null) {  
 for (RollbackRuleAttribute rule : this.rollbackRules) {  
 int depth = rule.getDepth(ex);  
 if (depth >= 0 && depth < deepest) {  
 deepest = depth;  
 winner = rule;  
 }  
 }  
 }  
  
 if (*logger*.isTraceEnabled()) {  
 *logger*.trace("Winning rollback rule is: " + winner);  
 }  
  
 // User superclass behavior (rollback on unchecked) if no rule matches.  
 if (winner == null) {  
 *logger*.trace("No relevant rollback rule found: applying default rules");  
 return super.rollbackOn(ex);  
 }  
  
 return !(winner instanceof NoRollbackRuleAttribute);  
}