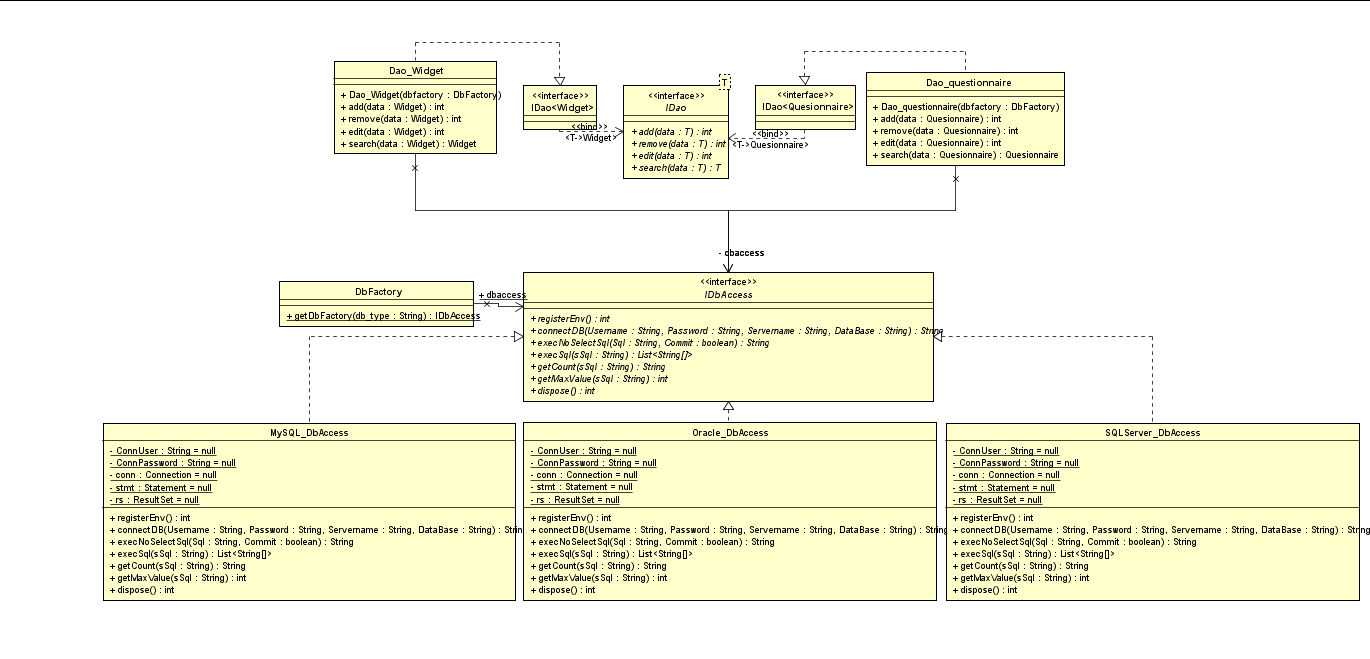
现需要设计一个表单系统。一个表单中的组件有普通的输入框、选择框和文件上传控件。所有组件都拥有控件名、控件类型、控件位置信息等属性。同时，每个控件也有其特殊属性，如输入框的输入校验，选择框的选项以及文件上传控件的后缀名限制等。表单自己也有很多属性。在本题中，请根据提示信息，设计该系统的部分类图。

1. 首先使用工厂模式建立访问多源数据库的类工厂，并对数据库操作实现基本封装，以及对每个类的增删改查行为。以下是执行SQL语句的流程：
   1. 加载数据库驱动
   2. 创建数据库连接
   3. 创建Statement，执行SQL语句
      1. 若是查询语句，执行的是Statement中executeQuery(String sql)，返回ResultSet型
      2. 若是非查询语句，执行的其中的exeCuteUpdate(String sql) 返回int型
   4. 处理返回结果
   5. 关闭连接

请根据以上提示设计数据库基础操作的类图。

1. 上述封装好的对数据库的操作类在服务器上需要保证只有一个实例，考虑到服务器面临多线程问题，请实现线程安全的单例模式，并做简要说明。
2. 试给出各个控件的类图，并实现对表单实现增，删，改，查操作的类图。
3. 该系统需要实现对所有SQL语句执行的log日志记录功能，以防止数据库被意外操作。请设计log类的基本功能，并描述如何整合进上述系统中。

1 

2

public class MySingleton {

volatile private static MySingleton instance = null;

private MySingleton(){}

public static MySingleton getInstance() {

try {

if(instance == null){

synchronized (MySingleton.class) {

if(instance == null){

instance = new MySingleton();

}

}

}

} catch (InterruptedException e) {

e.printStackTrace();

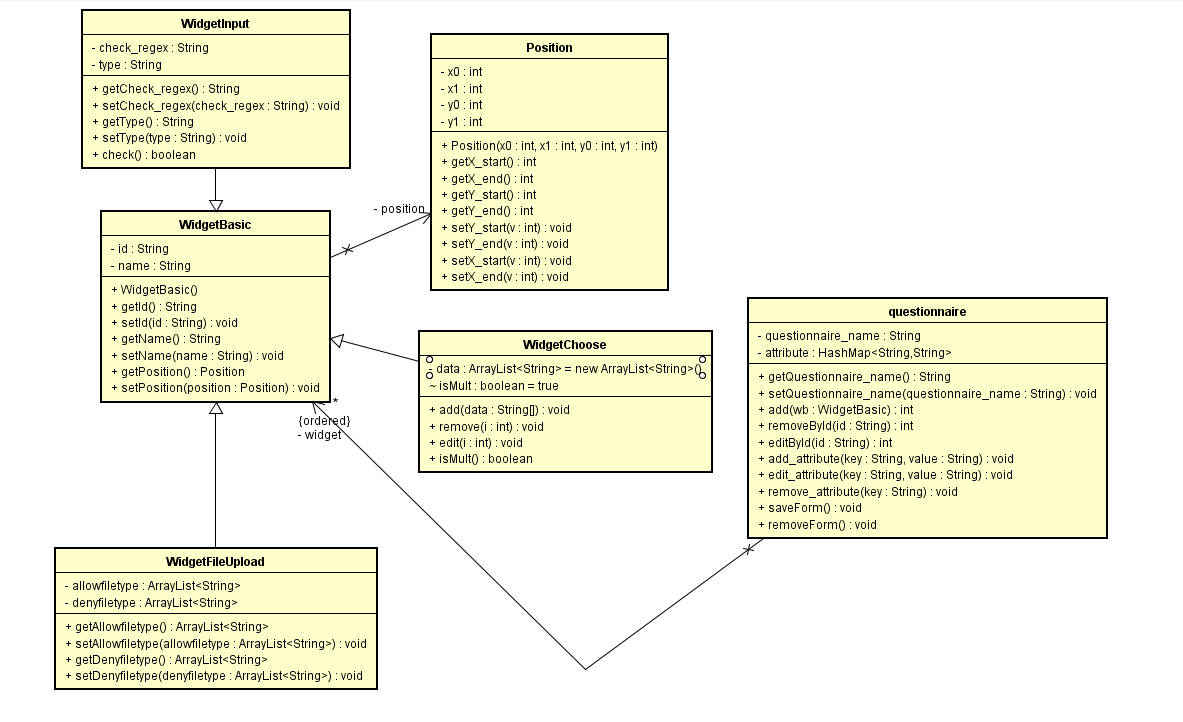
}

return instance;

}

}

3



4

针对上述需求，首先创建一个Log类，在该类中，实现写入log日志的具体方法，如在磁盘的指定文件上实现追加写的功能，以及在文件大小过大时，建立新的log日志文件等操作。然后通过动态代理的机制，代理数据库基本的操作类，将log日志的记录方法嵌入在执行SQL语句的方法中。最后通过工厂方法返回一个Proxy类。然后修改单例模式中的返回工厂即可。

Log.class:

**public** **class** Log {

**public** **static** **void** log(String sql) {

// **TODO** Auto-generated method stub

}

}

ProxyDbAccess.class:

**public** **class** ProxyDbAccess **implements** IDbAccess {

**public** IDbAccess realDbAccess;

**public** ProxyDbAccess(IDbAccess realDbAccess){

**this**.realDbAccess = realDbAccess;

}

@Override

**public** **int** registerEnv() **throws** SQLException {

**return** realDbAccess.registerEnv();

}

@Override

**public** String connectDB(String Username, String Password, String Servername, String DataBase) {

**return** realDbAccess.connectDB(Username, Password, Servername, DataBase);

}

@Override

**public** String execNoSelectSql(String Sql, **boolean** Commit) {

Log.*log*(Sql);

**return** realDbAccess.execNoSelectSql(Sql, Commit);

}

@Override

**public** List<String[]> execSql(String sSql) {

Log.*log*(sSql);

**return** realDbAccess.execSql(sSql);

}

@Override

**public** String getCount(String sSql) {

Log.*log*(sSql);

**return** realDbAccess.getCount(sSql);

}

@Override

**public** **int** getMaxValue(String sSql) {

Log.*log*(sSql);

**return** realDbAccess.getMaxValue(sSql);

}

@Override

**public** **int** dispose() {

**return** realDbAccess.dispose();

}

}