Java的动态代理技术（InvocationHandler和Proxy）

1. **接口InvocationHandler**：

**invoke** **引用；invocation** **引用**

**handler** 操作者，处理者，处理器

**只有一个方法：invoke方法**。

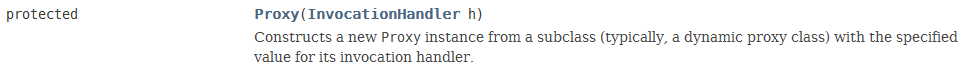


1. **接口Proxy**：

**proxy 代理权，代理人，替代值，受托人**

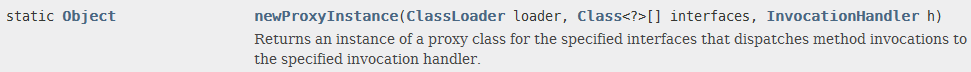
**存在于java.lang.reflec.\*包中，直接继承于java.lang.Object。**

**构造方法：**



**成员方法：四个特有的静态方法。**

1. **newProxyInstance方法，返回一个Object对象。**



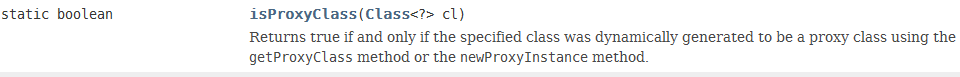
1. **getInvocationHandler返回一个InvocationHandler对象。**



1. **getProxyClass,返回Class对象。**



1. **isProxyClass：判断是否是一个ProxyClass对象。**



1. 方法：创建一个**新的类MyConnectionHandler**，实现接口InvocationHandler，并实现其中的**invoke方法**。创建一个新的方法bind，返回对**真正的连接对象**包装好后的**包装连接对象**，对于方法的调用，可以通过invoke方法进行限制，这里只是修改close方法，所以只需要修改一下close即可，而对于其他方法，只需要调用realConnection的相应方法即可。
2. 示例：

MyConnectionHandler：

**public** **class** MyConnectionHandler **implements** InvocationHandler {

**private** Connection realConn;

**private** Connection wrapedConnection;

**private** ConnectionPool connpool;

**private** **int** currentUseCount = 0;

**private** **int** maxUseCount = 5;

**public** MyConnectionHandler(ConnectionPool connectionPool) {

**this**.connpool = connectionPool ;

}

@Override

**public** Object invoke(Object proxy, Method method, Object[] args)

**throws** Throwable {

**if**("close".equals(method.getName())){

**this**.currentUseCount++;

**if** (**this**.currentUseCount<**this**.maxUseCount) {

**this**.connpool.connPool.addLast(**this**.wrapedConnection);

**return** **null**;

}

**else** {

**this**.connpool.currentCount--;

**return** method.invoke(**this**.realConn,args);

}

}

**else** {

**return** method.invoke(**this**.realConn,args);

}

}

**public** Connection bind(Connection realConnection) {

**this**.realConn = realConnection;

this.wrapedConnection = (Connection) Proxy.*newProxyInstance*(this

.getClass().getClassLoader(), new Class[] { Connection.class },this);

**return** wrapedConnection; }

ConnectionPool：

**public** **class** ConnectionPool {

**private** **int** maxCount = 8;

**public** **int** currentCount = 0;

**public** LinkedList<Connection> connPool = **new** LinkedList<Connection>();

**public** ConnectionPool() {

**for** (**int** i = 0; i < 5; i++) {

connPool.addLast(**this**.creatNewConn());

**this**.currentCount++;

}

}

**public** Connection getConnection() **throws** SQLException {

**synchronized** (connPool) {

**if** (connPool.size() > 0)

**return** connPool.removeFirst();

**else** **if** (**this**.currentCount < **this**.maxCount) {

**this**.currentCount++;

**return** **this**.creatNewConn();

}

**else**

**throw** **new** SQLException("已径达到了最大的限度。");

}

}

**public** **void** free(Connection conn) {

connPool.addLast(conn);

}

**public** Connection creatNewConn() {

Connection realConn = DriverUtil.*getConnection*();

// MyConnection myConn = new MyConnection(realConn, this);

// return myConn;

MyConnectionHandler mch = **new** MyConnectionHandler(**this**);

Connection wrapedConnection = mch.bind(realConn);

**return** wrapedConnection;

} }

Test2：

**public** **class** Tetst2 {

**public** **static** **void** main(String[] args) **throws** SQLException {

ConnectionPool connpool = **new** ConnectionPool();

**for** (**int** i = 0; i < 29; i++) {

Connection connection = connpool.getConnection();

System.***out***.println(connection);

connection.close();

}

} }

**操作成功。**