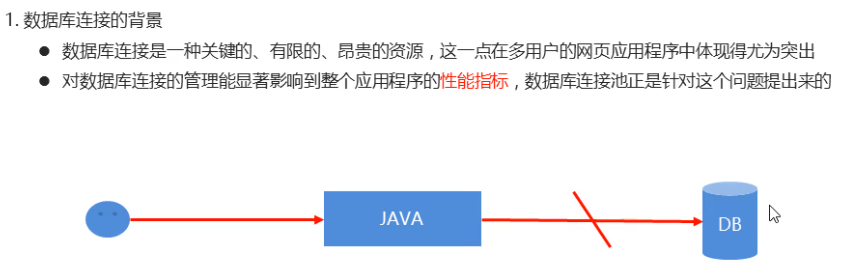
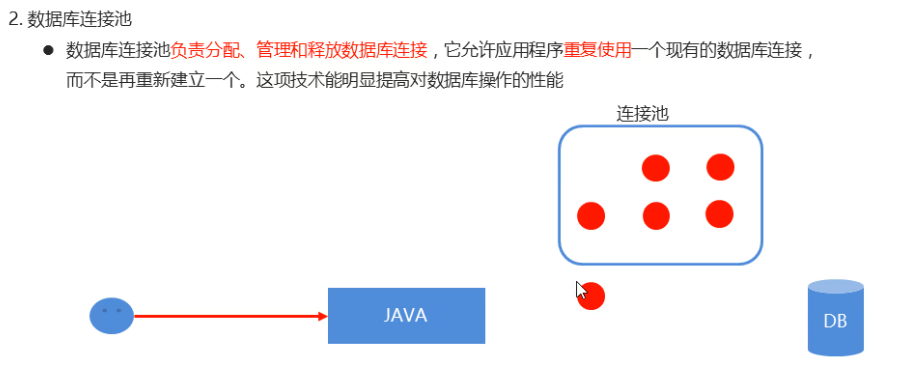
02 jdbc高级

# 1（掌握）数据库连接池的概念

## 1.1（掌握）数据库连接池的概念

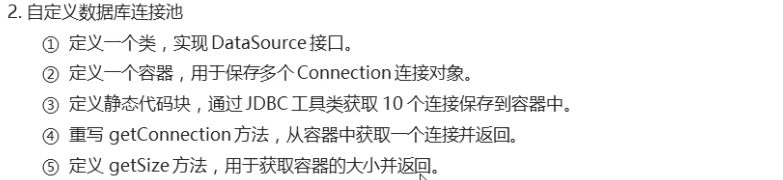




# 2（掌握）自定义数据连接池

## 2.1（掌握）自定义数据连接池





### 源码

|  |
| --- |
| public class MyDataSource implements DataSource {  //1.准备一个容器。用于保存多个数据库连接对象  private static List<Connection> pool = Collections.synchronizedList(new ArrayList<>());  //2.定义静态代码块,获取多个连接对象保存到容器中  static{  for(int i = 1; i <= 10; i++) {  Connection con = JDBCUtils.getConnection();  pool.add(con);  }  }  //4.提供一个获取连接池大小的方法  public int getSize() {  return pool.size();  }  //3.重写getConnection方法，用于返回一个连接对象  @Override  public Connection getConnection() throws SQLException {  if(pool.size() > 0) {  Connection con = pool.remove(0);  return con;  }else {  throw new RuntimeException("连接数量已用尽");  }  }  @Override  public Connection getConnection(String username, String password) throws SQLException {  return null;  }  @Override  public <T> T unwrap(Class<T> iface) throws SQLException {  return null;  }  @Override  public boolean isWrapperFor(Class<?> iface) throws SQLException {  return false;  }  @Override  public PrintWriter getLogWriter() throws SQLException {  return null;  }  @Override  public void setLogWriter(PrintWriter out) throws SQLException {  }  @Override  public void setLoginTimeout(int seconds) throws SQLException {  }  @Override  public int getLoginTimeout() throws SQLException {  return 0;  }  @Override  public Logger getParentLogger() throws SQLFeatureNotSupportedException {  return null;  }  } |



### 源码

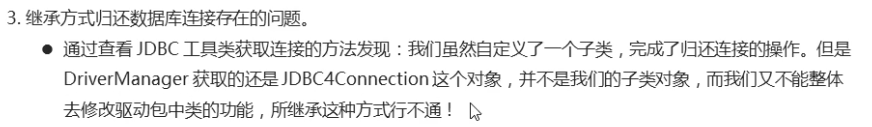
|  |
| --- |
| public class MyDataSourceTest {  public static void main(String[] args) throws Exception{  //1.创建连接池对象  MyDataSource dataSource = new MyDataSource();  System.out.println("使用之前的数量：" + dataSource.getSize());  //2.通过连接池对象获取连接对象  Connection con = dataSource.getConnection();  System.out.println(con.getClass());  //3.查询学生表的全部信息  String sql = "SELECT \* FROM student";  PreparedStatement pst = con.prepareStatement(sql);  //4.执行sql语句，接收结果集  ResultSet rs = pst.executeQuery();  //5.处理结果集  while(rs.next()) {  System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" + rs.getInt("age") + "\t" + rs.getDate("birthday"));  }  //6.释放资源  rs.close();  pst.close();  con.close(); // 用完以后，关闭连接  System.out.println("使用之后的数量：" + dataSource.getSize());  }  } |

## 2.2（了解）归还方式-继承

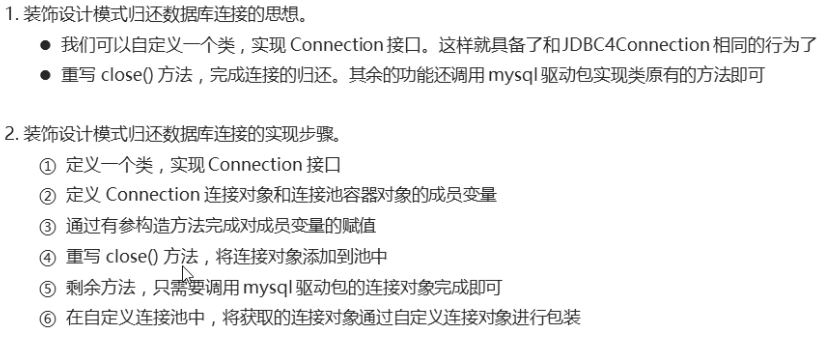


### 源码

|  |
| --- |
| /\*  自定义的连接对象  1.定义一个类，继承JDBC4Connection  2.定义Connection连接对象和容器对象的成员变量  3.通过有参构造方法为成员变量赋值  4.重写close方法，完成归还连接  \*/  public class MyConnection1 extends JDBC4Connection{//1.定义一个类，继承JDBC4Connection  //2.定义Connection连接对象和容器对象的成员变量  private Connection con;  private List<Connection> pool;  //3.通过有参构造方法为成员变量赋值  public MyConnection1(String hostToConnectTo, int portToConnectTo, Properties info, String databaseToConnectTo, String url,Connection con,List<Connection> pool) throws SQLException {  super(hostToConnectTo, portToConnectTo, info, databaseToConnectTo, url);  this.con = con;  this.pool = pool;  }  //4.重写close方法，完成归还连接  @Override  public void close() throws SQLException {  pool.add(con);  }  } |



## 2.3（掌握）归还方式-装饰设计模式

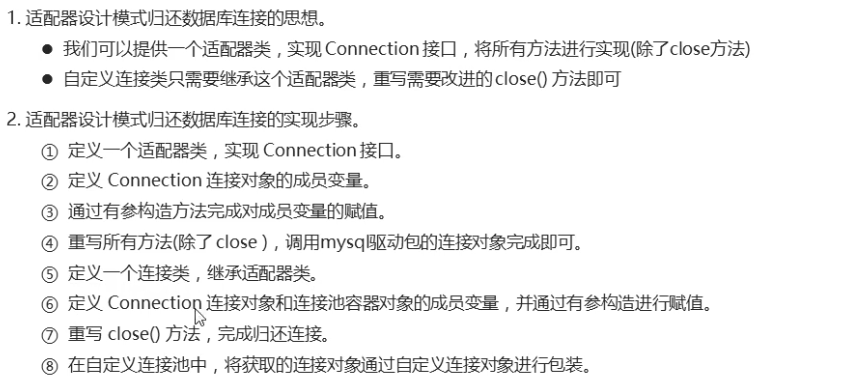


### 源码

|  |
| --- |
| /\*  1.定义一个类，实现Connection接口  2.定义连接对象和连接池容器对象的成员变量  3.通过有参构造方法为成员变量赋值  4.重写close方法，完成归还连接  5.剩余方法，还是调用原有的连接对象中的功能即可  \*/  //1.定义一个类，实现Connection接口  public class MyConnection2 implements Connection{  //2.定义连接对象和连接池容器对象的成员变量  private Connection con;  private List<Connection> pool;  //3.通过有参构造方法为成员变量赋值  public MyConnection2(Connection con,List<Connection> pool) {  this.con = con;  this.pool = pool;  }  //4.重写close方法，完成归还连接  @Override  public void close() throws SQLException {  pool.add(con);  }  //5.剩余方法，还是调用原有的连接对象中的功能即可  @Override  public Statement createStatement() throws SQLException {  return con.createStatement();  }  @Override  public PreparedStatement prepareStatement(String sql) throws SQLException {  return con.prepareStatement(sql);  }  @Override  public CallableStatement prepareCall(String sql) throws SQLException {  return con.prepareCall(sql);  }  @Override  public String nativeSQL(String sql) throws SQLException {  return con.nativeSQL(sql);  }  @Override  public void setAutoCommit(boolean autoCommit) throws SQLException {  con.setAutoCommit(autoCommit);  }  @Override  public boolean getAutoCommit() throws SQLException {  return con.getAutoCommit();  }  @Override  public void commit() throws SQLException {  con.commit();  }  @Override  public void rollback() throws SQLException {  con.rollback();  }  @Override  public boolean isClosed() throws SQLException {  return con.isClosed();  }  @Override  public DatabaseMetaData getMetaData() throws SQLException {  return con.getMetaData();  }  @Override  public void setReadOnly(boolean readOnly) throws SQLException {  con.setReadOnly(readOnly);  }  @Override  public boolean isReadOnly() throws SQLException {  return con.isReadOnly();  }  @Override  public void setCatalog(String catalog) throws SQLException {  con.setCatalog(catalog);  }  @Override  public String getCatalog() throws SQLException {  return con.getCatalog();  }  @Override  public void setTransactionIsolation(int level) throws SQLException {  con.setTransactionIsolation(level);  }  @Override  public int getTransactionIsolation() throws SQLException {  return con.getTransactionIsolation();  }  @Override  public SQLWarning getWarnings() throws SQLException {  return con.getWarnings();  }  @Override  public void clearWarnings() throws SQLException {  con.clearWarnings();  }  @Override  public Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException {  return con.createStatement(resultSetType,resultSetConcurrency);  }  @Override  public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency) throws SQLException {  return con.prepareStatement(sql,resultSetType,resultSetConcurrency);  }  @Override  public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException {  return con.prepareCall(sql,resultSetType,resultSetConcurrency);  }  @Override  public Map<String, Class<?>> getTypeMap() throws SQLException {  return con.getTypeMap();  }  @Override  public void setTypeMap(Map<String, Class<?>> map) throws SQLException {  con.setTypeMap(map);  }  @Override  public void setHoldability(int holdability) throws SQLException {  con.setHoldability(holdability);  }  @Override  public int getHoldability() throws SQLException {  return con.getHoldability();  }  @Override  public Savepoint setSavepoint() throws SQLException {  return con.setSavepoint();  }  @Override  public Savepoint setSavepoint(String name) throws SQLException {  return con.setSavepoint(name);  }  @Override  public void rollback(Savepoint savepoint) throws SQLException {  con.rollback(savepoint);  }  @Override  public void releaseSavepoint(Savepoint savepoint) throws SQLException {  con.releaseSavepoint(savepoint);  }  @Override  public Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.createStatement(resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.prepareStatement(sql,resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.prepareCall(sql,resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public PreparedStatement prepareStatement(String sql, int autoGeneratedKeys) throws SQLException {  return con.prepareStatement(sql,autoGeneratedKeys);  }  @Override  public PreparedStatement prepareStatement(String sql, int[] columnIndexes) throws SQLException {  return con.prepareStatement(sql,columnIndexes);  }  @Override  public PreparedStatement prepareStatement(String sql, String[] columnNames) throws SQLException {  return con.prepareStatement(sql,columnNames);  }  @Override  public Clob createClob() throws SQLException {  return con.createClob();  }  @Override  public Blob createBlob() throws SQLException {  return con.createBlob();  }  @Override  public NClob createNClob() throws SQLException {  return con.createNClob();  }  @Override  public SQLXML createSQLXML() throws SQLException {  return con.createSQLXML();  }  @Override  public boolean isValid(int timeout) throws SQLException {  return con.isValid(timeout);  }  @Override  public void setClientInfo(String name, String value) throws SQLClientInfoException {  con.setClientInfo(name,value);  }  @Override  public void setClientInfo(Properties properties) throws SQLClientInfoException {  con.setClientInfo(properties);  }  @Override  public String getClientInfo(String name) throws SQLException {  return con.getClientInfo(name);  }  @Override  public Properties getClientInfo() throws SQLException {  return con.getClientInfo();  }  @Override  public Array createArrayOf(String typeName, Object[] elements) throws SQLException {  return con.createArrayOf(typeName,elements);  }  @Override  public Struct createStruct(String typeName, Object[] attributes) throws SQLException {  return con.createStruct(typeName,attributes);  }  @Override  public void setSchema(String schema) throws SQLException {  con.setSchema(schema);  }  @Override  public String getSchema() throws SQLException {  return con.getSchema();  }  @Override  public void abort(Executor executor) throws SQLException {  con.abort(executor);  }  @Override  public void setNetworkTimeout(Executor executor, int milliseconds) throws SQLException {  con.setNetworkTimeout(executor,milliseconds);  }  @Override  public int getNetworkTimeout() throws SQLException {  return con.getNetworkTimeout();  }  @Override  public <T> T unwrap(Class<T> iface) throws SQLException {  return con.unwrap(iface);  }  @Override  public boolean isWrapperFor(Class<?> iface) throws SQLException {  return con.isWrapperFor(iface);  }  } |



## 2.4（掌握）归还方式-适配器设计模式



### 源码

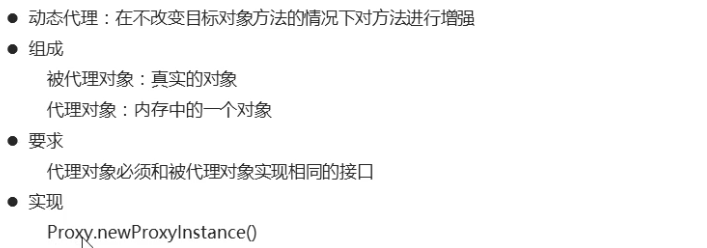
MyAdapter.java

|  |
| --- |
| /\*  1.定义一个适配器类。实现Connection接口  2.定义连接对象的成员变量  3.通过有参构造为变量赋值  4.重写所有的抽象方法(除了close)  \*/  public abstract class MyAdapter implements Connection {  //2.定义连接对象的成员变量  private Connection con;  //3.通过有参构造为变量赋值  public MyAdapter(Connection con) {  this.con = con;  }  //4.重写所有的抽象方法(除了close)  @Override  public Statement createStatement() throws SQLException {  return con.createStatement();  }  @Override  public PreparedStatement prepareStatement(String sql) throws SQLException {  return con.prepareStatement(sql);  }  @Override  public CallableStatement prepareCall(String sql) throws SQLException {  return con.prepareCall(sql);  }  @Override  public String nativeSQL(String sql) throws SQLException {  return con.nativeSQL(sql);  }  @Override  public void setAutoCommit(boolean autoCommit) throws SQLException {  con.setAutoCommit(autoCommit);  }  @Override  public boolean getAutoCommit() throws SQLException {  return con.getAutoCommit();  }  @Override  public void commit() throws SQLException {  con.commit();  }  @Override  public void rollback() throws SQLException {  con.rollback();  }  @Override  public boolean isClosed() throws SQLException {  return con.isClosed();  }  @Override  public DatabaseMetaData getMetaData() throws SQLException {  return con.getMetaData();  }  @Override  public void setReadOnly(boolean readOnly) throws SQLException {  con.setReadOnly(readOnly);  }  @Override  public boolean isReadOnly() throws SQLException {  return con.isReadOnly();  }  @Override  public void setCatalog(String catalog) throws SQLException {  con.setCatalog(catalog);  }  @Override  public String getCatalog() throws SQLException {  return con.getCatalog();  }  @Override  public void setTransactionIsolation(int level) throws SQLException {  con.setTransactionIsolation(level);  }  @Override  public int getTransactionIsolation() throws SQLException {  return con.getTransactionIsolation();  }  @Override  public SQLWarning getWarnings() throws SQLException {  return con.getWarnings();  }  @Override  public void clearWarnings() throws SQLException {  con.clearWarnings();  }  @Override  public Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException {  return con.createStatement(resultSetType,resultSetConcurrency);  }  @Override  public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency) throws SQLException {  return con.prepareStatement(sql,resultSetType,resultSetConcurrency);  }  @Override  public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException {  return con.prepareCall(sql,resultSetType,resultSetConcurrency);  }  @Override  public Map<String, Class<?>> getTypeMap() throws SQLException {  return con.getTypeMap();  }  @Override  public void setTypeMap(Map<String, Class<?>> map) throws SQLException {  con.setTypeMap(map);  }  @Override  public void setHoldability(int holdability) throws SQLException {  con.setHoldability(holdability);  }  @Override  public int getHoldability() throws SQLException {  return con.getHoldability();  }  @Override  public Savepoint setSavepoint() throws SQLException {  return con.setSavepoint();  }  @Override  public Savepoint setSavepoint(String name) throws SQLException {  return con.setSavepoint(name);  }  @Override  public void rollback(Savepoint savepoint) throws SQLException {  con.rollback(savepoint);  }  @Override  public void releaseSavepoint(Savepoint savepoint) throws SQLException {  con.releaseSavepoint(savepoint);  }  @Override  public Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.createStatement(resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.prepareStatement(sql,resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException {  return con.prepareCall(sql,resultSetType,resultSetConcurrency,resultSetHoldability);  }  @Override  public PreparedStatement prepareStatement(String sql, int autoGeneratedKeys) throws SQLException {  return con.prepareStatement(sql,autoGeneratedKeys);  }  @Override  public PreparedStatement prepareStatement(String sql, int[] columnIndexes) throws SQLException {  return con.prepareStatement(sql,columnIndexes);  }  @Override  public PreparedStatement prepareStatement(String sql, String[] columnNames) throws SQLException {  return con.prepareStatement(sql,columnNames);  }  @Override  public Clob createClob() throws SQLException {  return con.createClob();  }  @Override  public Blob createBlob() throws SQLException {  return con.createBlob();  }  @Override  public NClob createNClob() throws SQLException {  return con.createNClob();  }  @Override  public SQLXML createSQLXML() throws SQLException {  return con.createSQLXML();  }  @Override  public boolean isValid(int timeout) throws SQLException {  return con.isValid(timeout);  }  @Override  public void setClientInfo(String name, String value) throws SQLClientInfoException {  con.setClientInfo(name,value);  }  @Override  public void setClientInfo(Properties properties) throws SQLClientInfoException {  con.setClientInfo(properties);  }  @Override  public String getClientInfo(String name) throws SQLException {  return con.getClientInfo(name);  }  @Override  public Properties getClientInfo() throws SQLException {  return con.getClientInfo();  }  @Override  public Array createArrayOf(String typeName, Object[] elements) throws SQLException {  return con.createArrayOf(typeName,elements);  }  @Override  public Struct createStruct(String typeName, Object[] attributes) throws SQLException {  return con.createStruct(typeName,attributes);  }  @Override  public void setSchema(String schema) throws SQLException {  con.setSchema(schema);  }  @Override  public String getSchema() throws SQLException {  return con.getSchema();  }  @Override  public void abort(Executor executor) throws SQLException {  con.abort(executor);  }  @Override  public void setNetworkTimeout(Executor executor, int milliseconds) throws SQLException {  con.setNetworkTimeout(executor,milliseconds);  }  @Override  public int getNetworkTimeout() throws SQLException {  return con.getNetworkTimeout();  }  @Override  public <T> T unwrap(Class<T> iface) throws SQLException {  return con.unwrap(iface);  }  @Override  public boolean isWrapperFor(Class<?> iface) throws SQLException {  return con.isWrapperFor(iface);  }  } |

MyConnection3.java

|  |
| --- |
| /\*  1.定义一个类，继承适配器类  2.定义连接对象和连接池容器对象的成员变量  3.通过有参构造为变量赋值  4.重写close方法，完成归还连接  \*/  //1.定义一个类，继承适配器类  public class MyConnection3 extends MyAdapter {  //2.定义连接对象和连接池容器对象的成员变量  private Connection con;  private List<Connection> pool;  //3.通过有参构造为变量赋值  public MyConnection3(Connection con,List<Connection> pool) {  super(con);  this.con = con;  this.pool = pool;  }  //4.重写close方法，完成归还连接  @Override  public void close() {  pool.add(con);  }  } |

## 2.5（掌握）动态代理



### 源码

StudentInterface.java

|  |
| --- |
| public interface StudentInterface {  void eat(String name);  void study();  } |

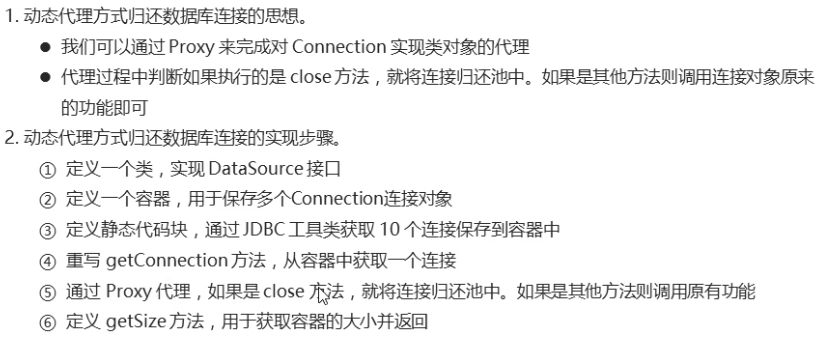
Student.java

|  |
| --- |
| public class Student implements StudentInterface{  public void eat(String name) {  System.out.println("学生吃" + name);  }  public void study() {  System.out.println("在家自学");  }  } |

Test.java

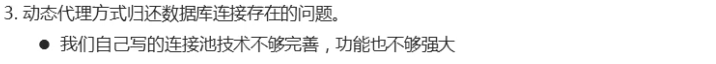
|  |
| --- |
| public class Test {  public static void main(String[] args) {  Student stu = new Student();  /\*stu.eat("米饭");  stu.study();\*/  /\*  要求：在不改动Student类中任何的代码的前提下，通过study方法输出一句话：来黑马学习  类加载器：和被代理对象使用相同的类加载器  接口类型Class数组：和被代理对象使用相同接口  代理规则：完成代理增强的功能  \*/  StudentInterface proxyStu = (StudentInterface) Proxy.newProxyInstance(stu.getClass().getClassLoader(), new Class[]{StudentInterface.class}, new InvocationHandler() {  /\*  执行Student类中所有的方法都会经过invoke方法  对method方法进行判断  如果是study，则对其增强  如果不是，还调用学生对象原有的功能即可  \*/  @Override  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  if(method.getName().equals("study")) {  System.out.println("来黑马学习");  return null;  }else {  return method.invoke(stu,args);  }  }  });  proxyStu.eat("米饭");  proxyStu.study();  }  } |

## 2.6（掌握）归还方式-动态代理



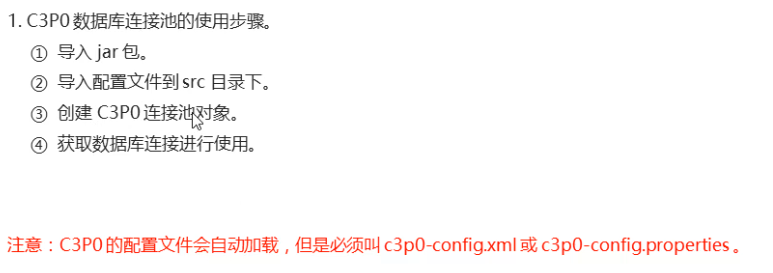
### 源码

|  |
| --- |
| public class MyDataSource implements DataSource {  //1.准备一个容器。用于保存多个数据库连接对象  private static List<Connection> pool = Collections.synchronizedList(new ArrayList<>());  //2.定义静态代码块,获取多个连接对象保存到容器中  static{  for(int i = 1; i <= 10; i++) {  Connection con = JDBCUtils.getConnection();  pool.add(con);  }  }  //4.提供一个获取连接池大小的方法  public int getSize() {  return pool.size();  }  /\*  动态代理方式  \*/  @Override  public Connection getConnection() throws SQLException {  if(pool.size() > 0) {  Connection con = pool.remove(0);  Connection proxyCon = (Connection) Proxy.newProxyInstance(con.getClass().getClassLoader(), new Class[]{Connection.class}, new InvocationHandler() {  /\*  执行Connection实现类连接对象所有的方法都会经过invoke  如果是close方法，归还连接  如果不是，直接执行连接对象原有的功能即可  \*/  @Override  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  if(method.getName().equals("close")) {  //归还连接  pool.add(con);  return null;  }else {  return method.invoke(con,args);  }  }  });  return proxyCon;  }else {  throw new RuntimeException("连接数量已用尽");  }  }  //3.重写getConnection方法，用于返回一个连接对象  /\*@Override  public Connection getConnection() throws SQLException {  if(pool.size() > 0) {  Connection con = pool.remove(0);  //通过自定义的连接对象 对原有的连接对象进行包装  //MyConnection2 myCon = new MyConnection2(con,pool);  MyConnection3 myCon = new MyConnection3(con,pool);  return myCon;  }else {  throw new RuntimeException("连接数量已用尽");  }  }\*/  @Override  public Connection getConnection(String username, String password) throws SQLException {  return null;  }  @Override  public <T> T unwrap(Class<T> iface) throws SQLException {  return null;  }  @Override  public boolean isWrapperFor(Class<?> iface) throws SQLException {  return false;  }  @Override  public PrintWriter getLogWriter() throws SQLException {  return null;  }  @Override  public void setLogWriter(PrintWriter out) throws SQLException {  }  @Override  public void setLoginTimeout(int seconds) throws SQLException {  }  @Override  public int getLoginTimeout() throws SQLException {  return 0;  }  @Override  public Logger getParentLogger() throws SQLFeatureNotSupportedException {  return null;  }  } |



# 3（掌握）开源数据库连接池

## 3.1（掌握）C3P0连接池



### 源码

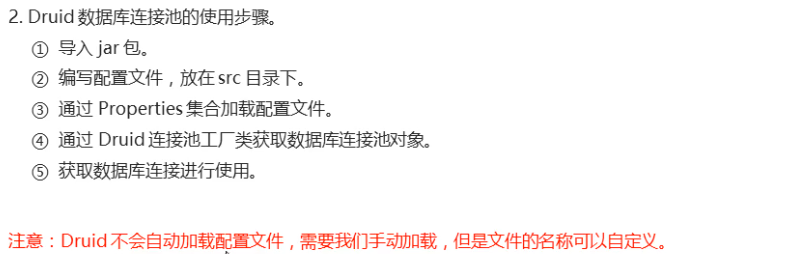
c3p0-config.xml

|  |
| --- |
| <c3p0-config>  <!-- 使用默认的配置读取连接池对象 -->  <default-config>  <!-- 连接参数 -->  <property name="driverClass">com.mysql.jdbc.Driver</property>  <property name="jdbcUrl">jdbc:mysql://192.168.59.129:3306/db14</property>  <property name="user">root</property>  <property name="password">itheima</property>    <!-- 连接池参数 -->  <!--初始化的连接数量-->  <property name="initialPoolSize">5</property>  <!--最大连接数量-->  <property name="maxPoolSize">10</property>  <!--超时时间-->  <property name="checkoutTimeout">3000</property>  </default-config>  <named-config name="otherc3p0">  <!-- 连接参数 -->  <property name="driverClass">com.mysql.jdbc.Driver</property>  <property name="jdbcUrl">jdbc:mysql://localhost:3306/db15</property>  <property name="user">root</property>  <property name="password">itheima</property>    <!-- 连接池参数 -->  <property name="initialPoolSize">5</property>  <property name="maxPoolSize">8</property>  <property name="checkoutTimeout">1000</property>  </named-config>  </c3p0-config> |

C3P0Test1.java

|  |
| --- |
| public class C3P0Test1 {  public static void main(String[] args) throws Exception{  //1.创建c3p0的数据库连接池对象  DataSource dataSource = new ComboPooledDataSource();  //2.通过连接池对象获取数据库连接  Connection con = dataSource.getConnection();  //3.执行操作  String sql = "SELECT \* FROM student";  PreparedStatement pst = con.prepareStatement(sql);  //4.执行sql语句，接收结果集  ResultSet rs = pst.executeQuery();  //5.处理结果集  while(rs.next()) {  System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" + rs.getInt("age") + "\t" + rs.getDate("birthday"));  }  //6.释放资源  rs.close();  pst.close();  con.close();  }  } |

## 3.2（掌握）Druid连接池



### 源码

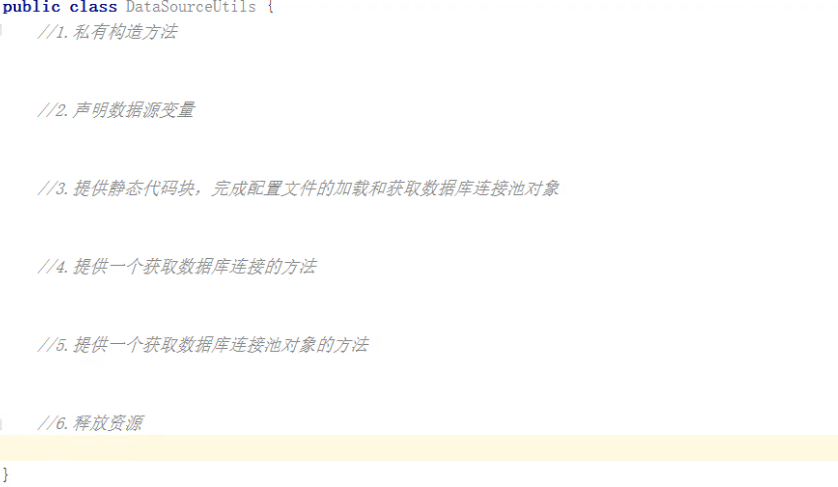
druid.properties

|  |
| --- |
| driverClassName=com.mysql.jdbc.Driver  url=jdbc:mysql://192.168.59.129:3306/db14  username=root  password=itheima  # 初始化连接数量  initialSize=5  # 最大连接数量  maxActive=10  # 超时时间  maxWait=3000 |

DruidTest1.java

|  |
| --- |
| /\*  1.通过Properties集合，加载配置文件  2.通过Druid连接池工厂类获取数据库连接池对象  3.通过连接池对象获取数据库连接进行使用  \*/  public class DruidTest1 {  public static void main(String[] args) throws Exception{  //获取配置文件的流对象  InputStream is = DruidTest1.class.getClassLoader().getResourceAsStream("druid.properties");  //1.通过Properties集合，加载配置文件  Properties prop = new Properties();  prop.load(is);  //2.通过Druid连接池工厂类获取数据库连接池对象  DataSource dataSource = DruidDataSourceFactory.createDataSource(prop);  //3.通过连接池对象获取数据库连接进行使用  Connection con = dataSource.getConnection();  String sql = "SELECT \* FROM student";  PreparedStatement pst = con.prepareStatement(sql);  //4.执行sql语句，接收结果集  ResultSet rs = pst.executeQuery();  //5.处理结果集  while(rs.next()) {  System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" + rs.getInt("age") + "\t" + rs.getDate("birthday"));  }  //6.释放资源  rs.close();  pst.close();  con.close();  }  } |

## 3.3（掌握）连接池工具类



### 源码

DataSourceUtils.java

|  |
| --- |
| /\*  数据库连接池的工具类  \*/  public class DataSourceUtils {  //1.私有构造方法  private DataSourceUtils(){}  //2.声明数据源变量  private static DataSource dataSource;  //3.提供静态代码块，完成配置文件的加载和获取数据库连接池对象  static{  try{  //完成配置文件的加载  InputStream is = DataSourceUtils.class.getClassLoader().getResourceAsStream("druid.properties");  Properties prop = new Properties();  prop.load(is);  //获取数据库连接池对象  dataSource = DruidDataSourceFactory.createDataSource(prop);  } catch (Exception e) {  e.printStackTrace();  }  }  //4.提供一个获取数据库连接的方法  public static Connection getConnection() {  Connection con = null;  try {  con = dataSource.getConnection();  } catch (SQLException e) {  e.printStackTrace();  }  return con;  }  //5.提供一个获取数据库连接池对象的方法  public static DataSource getDataSource() {  return dataSource;  }  //6.释放资源  public static void close(Connection con, Statement stat, ResultSet rs) {  if(con != null) {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if(stat != null) {  try {  stat.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if(rs != null) {  try {  rs.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  }  public static void close(Connection con, Statement stat) {  if(con != null) {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if(stat != null) {  try {  stat.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  }  } |

DruidTest2.java

|  |
| --- |
| public class DruidTest2 {  public static void main(String[] args) throws Exception{  //1.通过连接池工具类获取一个数据库连接  Connection con = DataSourceUtils.getConnection();  String sql = "SELECT \* FROM student";  PreparedStatement pst = con.prepareStatement(sql);  //2.执行sql语句，接收结果集  ResultSet rs = pst.executeQuery();  //3.处理结果集  while(rs.next()) {  System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" + rs.getInt("age") + "\t" + rs.getDate("birthday"));  }  //4.释放资源  DataSourceUtils.close(con,pst,rs);  }  } |