使用JDBC在MySQL数据库中如何快速批量插入数据

使用JDBC连接MySQL数据库进行数据插入的时候，特别是大批量数据连续插入（10W+），如何提高效率呢？

在JDBC编程接口中Statement 有两个方法特别值得注意：

void addBatch() throws SQLException

Adds a set of parameters to this PreparedStatement object’s batch of commands.

int[] executeBatch() throws SQLException

Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts. The int elements of the array that is returned are ordered to correspond to the commands in the batch, which are ordered according to the order in which they were added to the batch.

通过使用addBatch（）和executeBatch（）这一对方法可以实现批量处理数据。

不过值得注意的是，首先需要在数据库链接中设置手动提交，connection.setAutoCommit(false)，然后在执行Statement之后执行connection.commit()。

```
package cyl.demo.ipsearcher;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class DbStoreHelper {

private String insert_sql;
private String charset;
private boolean debug;

private String connectStr;
private String username;
private String password;

public DbStoreHelper() {
connectStr = "jdbc:mysql://localhost:3306/db_ip";
// connectStr += "?useServerPrepStmts=false&rewriteBatchedStatements=true";
insert_sql = "INSERT INTO tb_ipinfos (iplong1,iplong2,ipstr1,ipstr2,ipdesc) VALUES (?,?,?,?,?)";
charset = "gbk";
debug = true;
username = "root";
password = "***";
}

public void storeToDb(String srcFile) throws IOException {
BufferedReader bfr = new BufferedReader(new InputStreamReader(new FileInputStream(srcFile), charset));
try {
doStore(bfr);
} catch (Exception e) {
e.printStackTrace();
} finally {
bfr.close();
}
}

private void doStore(BufferedReader bfr) throws ClassNotFoundException, SQLException, IOException {
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(connectStr, username,password);
conn.setAutoCommit(false); // 设置手动提交
int count = 0;
```

```
PreparedStatement psts = conn.prepareStatement(insert_sql);
String line = null;
while (null != (line = bfr.readLine())) {
String[] infos = line.split(";");
if (infos.length < 5) continue;
if (debug) {
System.out.println(line);
}
psts.setLong(1, Long.valueOf(infos[0]));
psts.setLong(2, Long.valueOf(infos[1]));
psts.setString(3, infos[2]);
psts.setString(4, infos[3]);
psts.setString(5, infos[4]);
psts.addBatch(); // 加入批量处理
count++;
}
psts.executeBatch(); // 执行批量处理
conn.commit(); // 提交
System.out.println("All down : " + count);
conn.close();
}

}
```

执行完成以后：

```
All down : 103498
Convert finished.
All spend time/s : 47
```

一共10W+，执行时间一共花费 47 秒.

这个效率仍然不高，似乎没有达到想要的效果，需要进一步改进。

在MySQL JDBC连接字符串中还可以加入参数，

rewriteBatchedStatements=true，mysql默认关闭了batch处理，通过此参数进行打开，这个参数可以重写向数据库提交的SQL语句。

useServerPrepStmts=false，如果不开启(useServerPrepStmts=false)，使用com.mysql.jdbc.PreparedStatement进行本地SQL拼装，最后送到db上就是已经替换了?后的最终SQL.

在此稍加改进，连接字符串中加入下面语句（代码构造方法中去掉注释）：
connectStr += “?useServerPrepStmts=false&rewriteBatchedStatements=true”;

再次执行如下：

```
All down : 103498
Convert finished.
All spend time/s : 10
```

同样的数据量，这次执行只花费了10秒 ，处理效率大大提高.