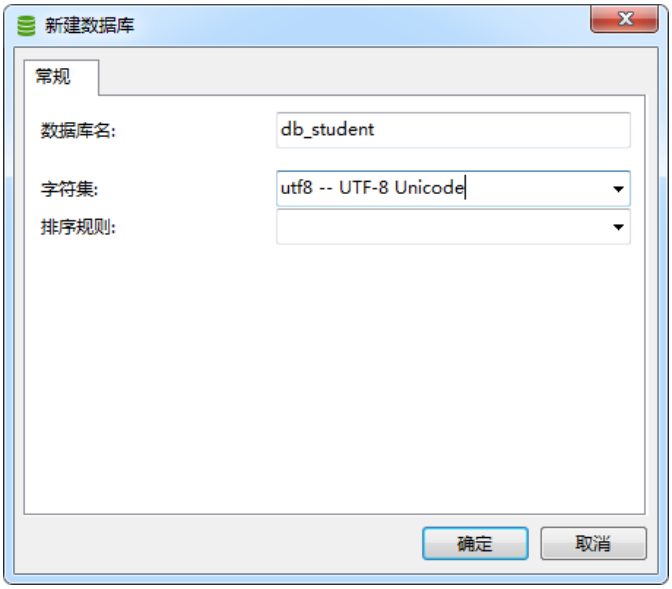
【Java框架型项目从入门到装逼】第九节 - 数据库建表和CRUD操作

1、新建学生表

这节课我们来把和数据库以及jdbc相关的内容完成，首先，进行数据库建表。数据库呢，我们采用MySQL数据库，我们可以通过navcat之类的管理工具来轻松建表。



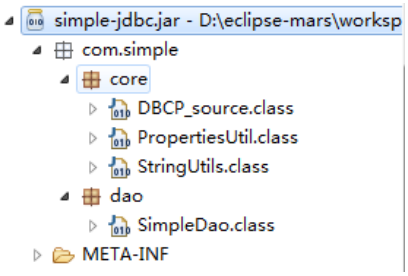
首先，我们得建一个数据库，名字叫db_student。然后，开始建表：

名		类型	长度	小数点	不是 null	
id	学号	int			☑	1
username	用户名	varchar	20		☐	
password	密码	varchar	20		☐	
name	学生姓名	varchar	30		☐	
sex	性别	varchar	10		☐	

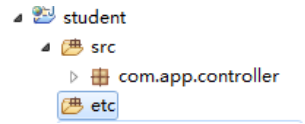
表名为t_student，保存。

学生表建好之后，我们开始测试具体的增删改查操作。

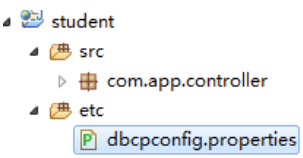
2、自定义jdbc框架 simple-jdbc



如图所示，simple-jdbc是我自己编写的一个jdbc框架，有点类似于spring-jdbc，采用DBCP数据源。目录结构如图所示，我已经把它打成了一个jar包，只需要在项目中引入即可。接下来，我们需要配置一下数据库的链接信息。



我们新建一个源文件夹etc，里面写一个配置文件，名字叫dbcpconfig.properties.注意哦，一定是叫这个名字，因为我在simple-jdbc框架中就设置了默认去读取这个文件。




```
#连接设置
driverClassName=com.mysql.jdbc.Driver
```

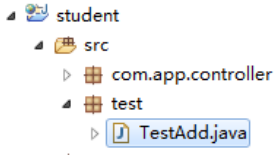
```
url=jdbc:mysql://127.0.0.1:3306/db_student
username=root
password=123

#<!-- 初始化连接 -->
initialSize=100

#最大连接数量
maxActive=50

#<!-- 最大空闲连接 -->
maxIdle=200

#<!-- 最小空闲连接 -->
minIdle=5

#<!-- 超时等待时间以毫秒为单位 6000毫秒/1000等于60秒 -->
maxWait=60000


#JDBC驱动建立连接时附带的连接属性属性的格式必须为这样：[属性名=property;]
#注意："user" 与 "password" 两个属性会被明确地传递，因此这里不需要包含他们。
connectionProperties=useUnicode=true;characterEncoding=UTF8

#指定由连接池所创建的连接的自动提交（auto-commit）状态。
defaultAutoCommit=true

#driver default 指定由连接池所创建的连接的只读（read-only）状态。
#如果没有设置该值，则“setReadOnly”方法将不被调用。（某些驱动并不支持只读模式，如：Informix）
defaultReadOnly=

#driver default 指定由连接池所创建的连接的事务级别（TransactionIsolation）。
#可用值为下列之一：（详情可见javadoc。）NONE,READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE
defaultTransactionIsolation=READ_UNCOMMITTED
```

对应的源码：

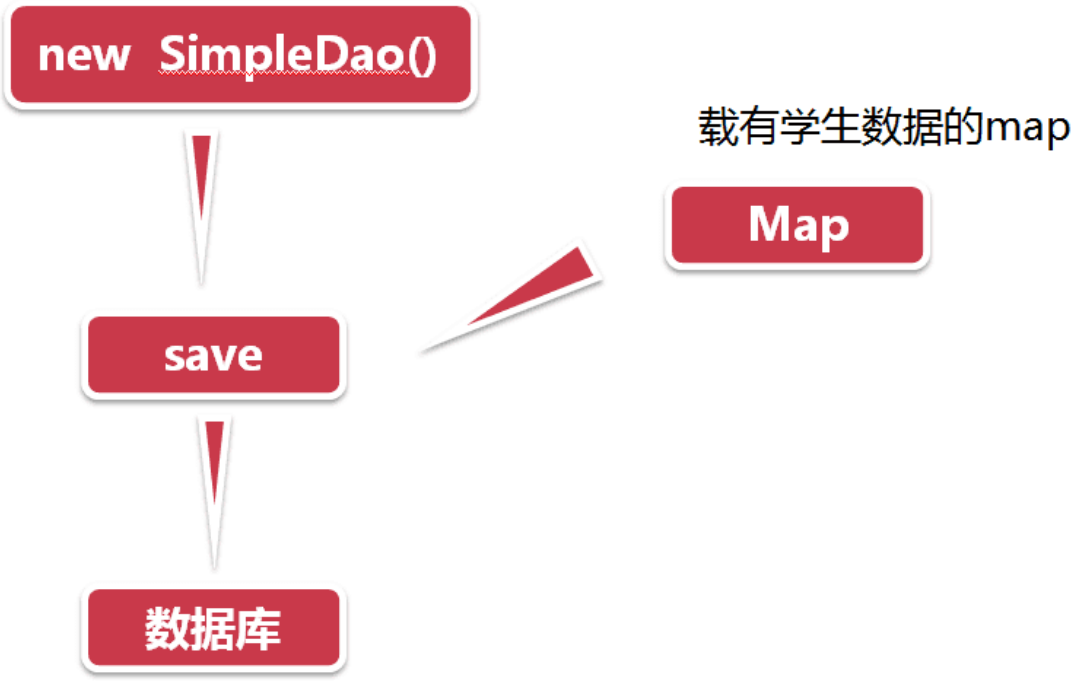
```
static {
        try {
            InputStream e = DBCP_source.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");
            Properties prop = new Properties();
            prop.load(e);
            ds = BasicDataSourceFactory.createDataSource(prop);
        } catch (Exception arg1) {
            throw new ExceptionInInitializerError(arg1);
        }
    }
```

3、新增用户

现在我们来测试新增用户的操作，建一个测试类：



因为我们使用了simple-jdbc，所以我们可以直接调用里面的save方法进行保存而不需要进行繁琐的jdbc操作。



在simple-jdbc中，有一个save方法，支持你传入一个map，然后把对应的数据保存到某个数据库的某一张表中。（MySQL）

案例：

```
public class TestAdd {
    public static void main(String[] args) {
        Map map = new HashMap();
        map.put("id", "2018012101"); //学号
        map.put("username", "zsf"); //账号
        map.put("password", "123"); //密码
        map.put("name", "张三丰"); //姓名
        map.put("sex", "男"); //性别

        SimpleDao dao = new SimpleDao();
        dao.save("db_student", "t_student", map);
        System.out.println("保存成功！");
    }
}
```

让我们看一下这个方法：

```
dao.save("db_student", "t_student", map);
```

第一个参数是数据库名称，第二个参数是表名，第三个参数是载有数据的map。通过这个方法，我们不需要任何繁琐的jdbc语句，就能实现数据新增的操作！是不是很方便？

效果：

id	username	password	name	sex
2018012101	zsf	123	张三丰	男

如果我们再次运行代码，就会报错：

```
com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '2018012101' for key 'PRIMARY'
```

因为id是主键，所以插入的时候是不允许重复的。

让我们多换几个数据测试一下，加几条数据：

id	username	password	name	sex
2018012101	zsf	123	张三丰	男
2018012102	hqg	123	洪七公	男
2018012103	gj	123	郭靖	男

3、删除用户

simple-jdbc没有直接删除数据的方法，但是支持用类似spring-jdbc的方式执行sql语句。比如：

```
public class TestDelete {
    public static void main(String[] args) {

        SimpleDao dao = new SimpleDao();
```

```
        dao.update("delete from t_student where id = ?", 2018012102);
        System.out.println("删除成功！");
    }
}
```

效果：

id	username	password	name	sex
2018012101	zsf	123	张三丰	男
2018012103	gj	123	郭靖	男

4、修改用户

修改数据，simple-jdbc提供了非常好用的方法，先来看一个具体例子：

```
public class TestModify {
    public static void main(String[] args) {
        Map map = new HashMap();
        map.put("id", "2018012103"); //学号
        map.put("username", "gj"); //账号
        map.put("password", "123456"); //密码
        map.put("name", "郭靖"); //姓名
        map.put("sex", "男"); //性别

        SimpleDao dao = new SimpleDao();
        dao.update("db_student", "t_student", map, "id");
        System.out.println("修改成功！");
    }
}
```

如代码所示，比如我要修改郭靖的数据，将其密码改为123456，就调用update方法。第一个参数是数据库名称，第二个参数是表名，第三个参数是载有数据的map，也就是修改后的数据。最后一个参数是主键，也就是说，根据什么字段信息来找到要修改的那一条数据？我们要修改郭靖的数据，id是一样的，都是2018012103，所以主键字段就是id，我们第四个参数就传一个“id”进去。

不需要任何繁琐的jdbc语句，你只需要一个map，搞定一切！

5、查询用户

查询用户我提供了多个接口：
1、queryForJsonObject
2、queryForJsonArray
3、queryForList
4、queryForMap
5、queryForBean
6、queryForString
7、queryForInt
8、queryForLong
9、queryForPage （分页数据）

每个方法看名字就知道是干什么用的了，这里我们就看一下其中最难的分页查询。分页查询在jdbc操作中一向是个难点，但是如果你使用我这个simple-jdbc框架，就是一句话的事情了。

例子：

```
public class TestQueryForPage {
    public static void main(String[] args) {
        String sql = "select * from t_student where 1=1 and sex = ?";

        SimpleDao dao = new SimpleDao();
        Map<String, Object> list = dao.queryForPage(sql, 1, 10, "男");
        System.out.println(list);
    }
}
```

如代码所示，我要查询出学生表中所有的男同胞，就用**queryForPage**方法，第一个参数是查询的**sql**语句，第二个参数是第几页，第三个参数是每页多少行，之后是一个变长数组，对应sql语句中的？。

效果：

```
{
total=2,
rows=[{"id":2018012101,"sex":"男","username":"zsf","name":"张三丰","password":"123"},
{"id":2018012103,"sex":"男","username":"gj","name":"郭靖","password":"123456"}]
}
```

分页的结果集就是具体的列表加上总条数，于是乎，这样就实现了。

[我要下载源码]