1、这里介绍核心配置文件中配置数据库相关属性的意义

核心配置文件的环境配置代码如下

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <!--environments是环境配置，是复数说明下面可以配置多个环境
        所以default属性的值表示这个配置文件用下面的哪一套数据库配置信息
        这里使用development1
    -->
    <environments default="development1">

        <!--第一套数据库连接信息配置，唯一标识id叫development-->
        <environment id="development1">

            <!--transactionManager：表示mybatis提交事务和回滚事务的方式
            type：事务的处理类型：
            1、填JDBC 表示mybatis底层是调用JDBC规范中的Connection对象进行事务提交与回滚
的
            2、填MANAGED 表示mybatis把事务委托给其他容器处理（一个服务器软件，或者是一个框
架（spring））
            -->
            <transactionManager type="JDBC"/>

            <!--dataSource：表示使用的数据源类型，Java中，实现了javax.sql.DataSource接
口
                的都可以当作数据源，数据源就是Connection对象
                type：指定数据源类型
                POOLED：表示mybatis到连接池中获取Connection对象，即使用连接池，创建
PooledDataSource类对                   象
                UNPOOLED：不使用连接池，mybatis每次执行sql语句，创建一个Connection对
象，执行完后将                 Connection对象释放，mybatis创建的是
UnPooledDataSource类对象
                JNDI：是一个java命名和目录（相当于windows注册表，了解就好）
-->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>

    <mappers>
        <!--这里也可以添加多个sql映射文件，因为项目中不止一个表
            这就是映射文件中namespace属性的作用了-->
        <!-- <mapper resource="StudentDao.xml"/>-->
        <mapper resource="com/studymyself/dao/UserDao.xml"/>
    </mappers>
```

```
</configuration>
```

2、mybatis中的数据源和连接池

```
上面的核心配置文件中，数据源的选择类型有三种，其中前两种需要我们掌握
--连接池
    mybatis中实现了javax.sql.DataSource规范，该规范中有许多获取Connection对象的方法，有两个实现类:
    PooledDataSource和UnPooledDataSource
    其中实现类PooledDataSource是可以创建多个Connection对象，以队列的形式存到集合中，先存先被取出。当一个线程需要连接对象使直接从该集合中取出最先存的一个，另一个线程需要时又取出第二存进去的一个，线程结束后将这个连接对象放回集合中，排到后面。这个集合就是线程池，而且是线程安全的，不允许一个Connection对象被两个线程获取。
    --这种数据源的方式是实际开发中使用的，极大的减少了获取连接对象的时间
    实现类UnPooledPooledDataSource，该类中的方法是在需要的时候获取创建Connection对象，完成线程任务后释放掉。一般开发中我们只是在测试阶段使用。
```

实现类PooledDataSource代码片段

```java
public class PooledDataSource implements DataSource {
    private static final Log log = LogFactory.getLog(PooledDataSource.class);
    private final PoolState state = new PoolState(this);
    private final UnpooledDataSource dataSource;
    protected int poolMaximumActiveConnections = 10;
    protected int poolMaximumIdleConnections = 5;
    protected int poolMaximumCheckoutTime = 20000;
    protected int poolTimeToWait = 20000;
    protected int poolMaximumLocalBadConnectionTolerance = 3;
    protected String poolPingQuery = "NO PING QUERY SET";
    protected boolean poolPingEnabled;
    protected int poolPingConnectionsNotUsedFor;
    private int expectedConnectionTypeCode;

    public PooledDataSource() {
        this.dataSource = new UnpooledDataSource();
    }

    public PooledDataSource(UnpooledDataSource dataSource) {
        this.dataSource = dataSource;
    }

    public PooledDataSource(String driver, String url, String username, String
password) {
        this.dataSource = new UnpooledDataSource(driver, url, username,
password);
        this.expectedConnectionTypeCode =
this.assembleConnectionTypeCode(this.dataSource.getUrl(),
this.dataSource.getUsername(), this.dataSource.getPassword());
    }

    public PooledDataSource(String driver, String url, Properties
driverProperties) {
        this.dataSource = new UnpooledDataSource(driver, url, driverProperties);
        this.expectedConnectionTypeCode =
this.assembleConnectionTypeCode(this.dataSource.getUrl(),
this.dataSource.getUsername(), this.dataSource.getPassword());
    }

    public PooledDataSource(ClassLoader driverClassLoader, String driver, String
url, String username, String password) {
        this.dataSource = new UnpooledDataSource(driverClassLoader, driver, url,
username, password);
        this.expectedConnectionTypeCode =
this.assembleConnectionTypeCode(this.dataSource.getUrl(),
this.dataSource.getUsername(), this.dataSource.getPassword());
    }

    public PooledDataSource(ClassLoader driverClassLoader, String driver, String
url, Properties driverProperties) {
        this.dataSource = new UnpooledDataSource(driverClassLoader, driver, url,
driverProperties);
        this.expectedConnectionTypeCode =
this.assembleConnectionTypeCode(this.dataSource.getUrl(),
this.dataSource.getUsername(), this.dataSource.getPassword());
    }

    public Connection getConnection() throws SQLException {
        return this.popConnection(this.dataSource.getUsername(),
this.dataSource.getPassword()).getProxyConnection();
    }

    public Connection getConnection(String username, String password) throws
SQLException {
        return this.popConnection(username, password).getProxyConnection();
    }
    ...
```

实现类UnPooledPooledDataSource代码片段

```java
public class UnpooledDataSource implements DataSource {
    private ClassLoader driverClassLoader;
    private Properties driverProperties;
    private static Map<String, Driver> registeredDrivers = new
ConcurrentHashMap();
    private String driver;
    private String url;
    private String username;
    private String password;
    private Boolean autoCommit;
    private Integer defaultTransactionIsolationLevel;
    private Integer defaultNetworkTimeout;

    public UnpooledDataSource() {
    }

    public UnpooledDataSource(String driver, String url, String username, String
password) {
        this.driver = driver;
        this.url = url;
        this.username = username;
        this.password = password;
    }

    public UnpooledDataSource(String driver, String url, Properties driverProperties) {
        this.driver = driver;
        this.url = url;
        this.driverProperties = driverProperties;
    }

    public UnpooledDataSource(ClassLoader driverClassLoader, String driver, String url, String username, String password) {
        this.driverClassLoader = driverClassLoader;
        this.driver = driver;
        this.url = url;
        this.username = username;
        this.password = password;
    }

    public UnpooledDataSource(ClassLoader driverClassLoader, String driver, String url, Properties driverProperties) {
        this.driverClassLoader = driverClassLoader;
        this.driver = driver;
        this.url = url;
        this.driverProperties = driverProperties;
    }

    public Connection getConnection() throws SQLException {
        return this.doGetConnection(this.username, this.password);
    }

    public Connection getConnection(String username, String password) throws SQLException {
        return this.doGetConnection(username, password);
    }
    ...
```

3、指定多个mapper文件

当我们一个项目中需要的mapper文件很多时，在标签中添加太多的标签来指定各个接口的映射文件非常繁琐，所以我们可以直接指定这些mapper文件的包名，就可直接将包下的mapper文件全部配置到mybatis中了，具体实现和要求如下所示：