# DataSource数据源

什么是数据源?

JDBC2.0 提供了javax.sql.DataSource接口，它负责建立与数据库的连接，当在应用程序中访问数据库时不必编写连接数据库的代码，直接引用DataSource获取数据库的连接对象即可。用于获取操作数据Connection对象。

数据源与数据库连接池

数据源建立多个数据库连接，这些数据库连接会保存在数据库连接池中，当需要访问数据库时，只需要从数据库连接池中

获取空闲的数据库连接，当程序访问数据库结束时，数据库连接会放回数据库连接池中。

数据源DataSource与JNDI

数据源DataSource是JNDI资源的一种，很简单，就是将“DataSource”字符串名称与真正的DataSource对象绑定起来，方便获取。

数据库连接池的优势

传统的JDBC访问数据库技术，每次访问数据库都需要通过数据库驱动器Driver和数据库名称以及密码等等资源建立数据库连接。

这样的连接存在两大问题：

1. 频繁的建立数据库连接与断开数据库，这样会消耗大量的资源和时间，降低性能。

2. 数据库的连接需要用户名和密码等等，这些需要一定的内存和CPU一定开销。

JDBC

Java Database Connectivity(JDBC)JDBC以一种统一的方式来对各种各样的数据库进行存取。和ODBC一样，JDBC为开发人员隐藏了不同数据库的不同特性。另外，由于JDBC建立在Java的基础上,因此还提供了数据库存取的平台独立性。JDBC定义了4种不同的驱动程序：

JDBC-ODBCBridge、

JDBC-native driverbridge、

JDBC-networkbridge、

PureJavadriver。

在一个多层的企业级应用中，更大的可能是在客户端和一个EJB进行通信，采用EJB建立数据库连接。

为了实现和改进可伸缩性和系统性能， 也可以采用连接缓冲池（connection pool）优化数据库连接。

JNDI

Java Naming and Directory Interface (JNDI)JNDI API被用于执行名字和目录服务。

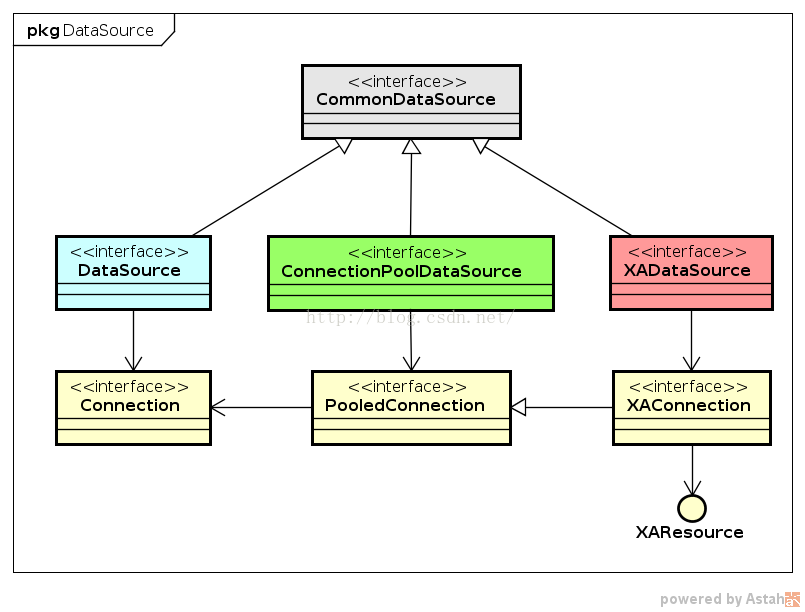
它提供了一致的模型来存取和操作企业级的资源如DNS和LDAP，本地文件系统，后者在应用服务器中的对象。在JNDI中，在目录结构中的每一个结点称为context。每一个JNDI名字都是相对于context的。应用可以通过这个初始化的context经有这个目录树来定位它所需要的资源或对象。

关键

JNDI相对与JDBC来说是他的灵活性,程序员不需要关心“具体的数据库后台是什么？JDBC驱动程序是什么？JDBC URL格式是什么？访问数据库的用户名和口令是什么？”等等这些问题，程序员编写的程序应该没有对 JDBC 驱动程序的引用，没有服务器名称，没有用户名称或口令 —— 甚至没有数据库池或连接管理。而是把这些问题交给J2EE容器来配置和管理，程序员只需要对这些配置和管理进行引用即可。

# Java 数据源 DataSource 架构分析 (jdk7/tomcat7)

Java 数据源类图



简单说明

CommonDataSource：是对数据源概念的顶层抽象，约束了数据源必需实现的方法。  
从架构图中可以看出数据源有三种类型，即CommonDataSource抽象有三种实现方式，分别是：  
 DataSource、

XADataSource、

ConnectionPoolDataSource，

三种类型数据源说明如下：  
DataSource：基础实现，数据库物理连接的工厂，用于创建标准的数据库物理连接Connection，JDBC 2.0中诞生，与早先的DriverManager用途一样，新版本JDBC编程已推荐使用DataSource；

XADataSource：分布式事务实现，为支持分布式事务而诞生，此数据源直接生产出的不是数据库物理连接Connection，而是一个支持XA的XAConnection对象，  
XAConnection对象可以直接生产数据库物理连接Connection，同时生产XAResource用于支持XA事务，通常XAConnection对象生产出的数据库物理连接Connection需要和该XAConnection生产出的XAResource对象配合使用以完成XA事务处理（请参考 XA 和 JTA 规范）；

ConnectionPoolDataSource：连接池实现，此数据源实现并不直接创建数据库物理连接，而是一个逻辑实现，它的作用在于池化数据库物理连接，由于数据库物理连接是一个重量级的对象，频繁的创建销毁很影响性能，将物理连接池化后可降低创建和销毁的频率，复用连接以充分利用连接资源，此数据源通常不为用户应用所知，通常是由中间件服务方来调度，中间件服务方通过它获取一个池化对象PooledConnection，再通过该PooledConnection间接获取到物理连接，获取方式即是调用其getConnection()方法，  
然后***创建出一个规范定义的Connection接口句柄提供给应用使用，应用通过该句柄间接使用物理连接，在应用调用句柄的close方法时，中间件服务方的实现并没有真正调用物理连接的关闭，而是将其归还到连接池中***。

# 黑马程序员\_使用DataSource接口连接数据源

普通的数据库连接往往直接使用**DriverManager**获取，这种方式需要将连接参数，如数据库驱动名、连接字符串、用户名、密码等数据库连接信息写到配置文件或者程序中，其他地方调用连接操作数据库之后，需要释放并关闭数据库连接，并没有重复利用数据库的连接资源。下面的示例代码给出了这种方式获取数据库连接的具体实现：

public static Connection getConnection() throws Exception{

//指定数据库驱动名

String driver = “org.gjt.mm.mysql.Driver” ;

//指定数据库连接url，此处连接本地mysql中的名为dbs的数据库

String url = “jdbc:mysql://localhost/dbs” ;

//指定连接数据库的用户名

String user = “root” ;

//指定连接数据库的密码

String pwd = “root”

//将驱动注册到虚拟机classpath环境

Class.forName(driver) ;

//调用DriverManager的getConnection方法获得一个连接

Connection conn = DriverManager.getConnection(url, user, pwd) ;

return conn ;

}

这段代码实现了Connection的获取功能，使用该方法无法重复利用连接资源，调用该方法的程序需要在用完之后关闭连接。创建一个连接是一个很费时间和资源的操作，需要与远程数据库之间进行“三次握手”。如果能把回收的连接重新利用，将会减少新创建连接的开销，显著地提高系统运行性能，JDBC2.0提供的DataSource为此提供了较好的解决方案。

**一个DataSource对象代表一个真正的数据源**。当一个DataSource对象注册到JNDI（Java Naming and Directory Interface）名字服务中时，应用程序就可以通过JNDI名字服务获得DataSource对象，并用它们来产生一个与DataSource代表的数据源之间的连接。

关于数据源的连接信息，例如数据库服务器的驱动名、连接字符串、用户名、密码等，都包含在DataSource对象的属性中。这样，对应用程序的设计来说就更方便了，程序中不需要再处理这些信息，直接根据JNDI名字就可以获取DataSource，然后根据DataSource来获取连接。如果数据源要移植到另一个数据库驱动中，程序也很容易修改。所需要做的只是更改DataSource的相关属性，而使用DataSource对象的程序不需要做改动。

配置DataSource，包括设定DataSource的属性，然后将其注册到JNDI名字服务中去。在注册DataSource对象的过程中，需要把DataSource对象和一个逻辑名字关联起来，比如名字为“JDBC/BBS”。

本程序使用Tomcat内置的DBCP连接池实现DataSource。数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而再不是重新建立一个；释放空闲时间超过最大空闲时间的数据库连接，来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。

此处给出配置文件中连接池配置参数。

<Resource auth=*“Container”* name=*“jdbc/bbs”* Forum=*“javax.sql.DataSource”* />

<ResourceParams name=*“jdbc/bbs”*>

<parameter>

<name>url</name>

//设置数据库连接url

<value>jdbc:mysql://localhost/bbs</value>

//此处连接本地mysql中的名为bbs的数据库

</parameter>

<parameter>

<name>maxIdle</name>

//可以同时闲置在连接池中的连接的最大数目

<value>5</value>

</parameter>

<parameter>

<name>maxActive</name>

//最大激活连接数

<value>50</value>

</parameter>

<parameter>

<name>maxWait</name>

//请求连接最大等待时间，以秒为单位

<value>3000</value>

</parameter>

<parameter>

<name>driverClassName</name>

//指定数据库驱动名

<value>org.gjt.mm.mysql.Driver</value>

//使用MySQL数据库JDBC驱动

</parameter>

<parameter>

<name>removeAbandoned</name>

//是否回收被遗弃的（一般是忘了释放的）数据库连接到连接池中

<value>true</value>

</parameter>

<parameter>

<name>username</name>

//指定连接数据库的用户名

<value>root</value>

</parameter>

<parameter>

<name>password</name>

//指定连接数据库的密码

<value>root</value>

</parameter>

<parameter>

<name>factory</name>

//使用Tomcat内置的DBCP连接池

<value>org.apache.commons.dbcp.BasicDataSourceFactory</value>

</parameter>

</ResourceParams>

# [JDBC数据库连接池的必要性(转载)](http://junxiangzheng.iteye.com/blog/939749)

**1.JDBC数据库连接池的必要性**

在使用开发基于数据库的web程序时，传统的模式基本是按以下步骤：

在主程序（如servlet、beans）中建立数据库连接。

进行sql操作

断开数据库连接。

这种模式开发，存在的问题:

普通的JDBC数据库连接使用 DriverManager 来获取，每次向数据库建立连接的时候都要将 Connection 加载到内存中，再验证用户名和密码(得花费0.05s～1s的时间)。需要数据库连接的时候，就向数据库要求一个，执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。数据库的连接资源并没有得到很好的重复利用.若同时有几百人甚至几千人在线，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃。

对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库。

这种开发不能控制被创建的连接对象数，系统资源会被毫无顾及的分配出去，如连接过多，也可能导致内存泄漏，服务器崩溃.

**2.数据库连接池（connection pool）**

为解决传统开发中的数据库连接问题，可以采用数据库连接池技术。

数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是重新建立一个。

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由最小数据库连接数来设定的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的最大数据库连接数量限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。

**3.数据库连接池技术的优点**

(1)资源重用：由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。

(2)更快的系统反应速度:数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间

(3)新的资源分配手段对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置实现某一应用最大可用数据库连接数的限制避免某一应用独占所有的数据库资源.

(4)统一的连接管理，避免数据库连接泄露在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露

**4.两种开源的数据库连接池：**

(1)：JDBC 的数据库连接池使用 javax.sql.DataSource 来表示，**DataSource 只是一个接口**，该接口通常由服务器(Weblogic, WebSphere, Tomcat)提供实现。

(2)：DBCP 数据库连接池是 Apache 软件基金组织下的开源连接池实现, 该连接池依赖该组织下的另一个开源系统：Common-pool. 如需使用该连接池实现，应在系统中增加如下两个 jar 文件：Commons-dbcp.jar：连接池的实现 Commons-pool.jar：连接池依赖库

**5.DBCP连接池使用的两种形式**

**(1)：直接设置参数的形式：**

//创建数据源对象

BasicDataSource bds = new BasicDataSource();

//设置连接数据库的驱动

bds.setDriverClassName("com.mysql.jdbc.Driver");

//设置连接数据库的url

bds.setUrl("jdbc:mysql://localhost:3306/test");

//设置连接数据库的用户名

bds.setUsername("root");

//设置连接数据库的密码

bds.setPassword("root");

//设置连接池启动时的初始值

bds.setInitialSize(5);

//设置连接池的最大值

bds.setMaxActive(50);

//最大空闲值.当经过一个高峰时间后，连接池可以慢慢将已经

//用不到的连接慢慢释放一部分，一直减少到maxIdle为止

bds.setMaxIdle(20);

//最小空闲值.当空闲的连接数少于该值时，连接池就会预申请一些连接，

//以避免洪峰来时再申请而造成的性能开销

bds.setMinIdle(5);

得到连接：Connection conn = dbs.getConnection();

……..

(2)：读取配置文件(properties)的方式。

//构建properties对象

Properties properties = new Properties();

  //加载配置文件

properties.load(inputSream);

//BasicDataSourceFactory利用属性文件的信息创建BasicDataSource数据源

dataSource=BasicDataSourceFactory.*createDataSource*(properties);

//获取连接

Connection conn = dataSource.getConnection();

配置文件内容形式如下(注意key严格遵循大小写)：

#连接字符串

url=jdbc:mysql://localhost:3306/test

#用户名

username=root

#密码

password=root

#驱动的类路径

driverClassName=com.mysql.jdbc.Driver

#连接池启动时的初始值

initialSize=1

#连接池的最大值

maxActive=50

#最大空闲数

maxIdle=20

#最小空闲数

minIdle=5

# 【spring框架】关于DataSource(上)

DataSource的概念：  
这是一个接口，可以获取数据库的Connection。是标准化的，取得连接的一种方式。  
  
在hibernate里面是可以提供各种各样的连接池的，spring里面也可以提供数据库连接池，里面有一大堆的数据连接，然后想取的时候getConnection就取出来了。  
  
Spring配置数据库连接池的例子:

1. <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
2. *<!-- results in a setDriverClassName(String) call -->*
3. <property name="driverClassName">
4. <value>com.mysql.jdbc.Driver</value>
5. </property>
6. <property name="url">
7. <value>jdbc:mysql://localhost:3306/mydb</value>
8. </property>
9. <property name="username">
10. <value>root</value>
11. </property>
12. <property name="password">
13. <value>masterkaoli</value>
14. </property>
15. </bean>

写法2：

1. <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
2. *<!-- results in a setDriverClassName(String) call -->*
3. <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
4. <property name="url" value="jdbc:mysql://localhost:3306/mydb"/>
5. <property name="username" value="root"/>
6. <property name="password" value="masterkaoli"/>
7. </bean>

其中org.apache.commons.dbcp.BasicDataSource中的dbcp是"data-base-conntection-pool"数据库连接池的意思。  
  
要引入commons-dbcp.jar包、commons-pool.jar包和Mysql的数据库连接包mysql-connector-java-5.0.3-bin.jar

实验:  
在数据库中创建名为spring的数据库，里面建立一张user表，里面有主键id和name属性。

beans.xml:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
8. http://www.springframework.org/schema/context
9. http://www.springframework.org/schema/context/spring-context-2.5.xsd
10. http://www.springframework.org/schema/aop
11. http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
13. <context:annotation-config/>
14. <context:component-scan base-package="cn.edu.hpu"/>
16. <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
17. *<!-- results in a setDriverClassName(String) call -->*
18. <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
19. <property name="url" value="jdbc:mysql://localhost:3306/spring"/>
20. <property name="username" value="root"/>
21. <property name="password" value="1234"/>
22. </bean>
23. </beans>

UserDaoImpl.java:

1. package cn.edu.hpu.dao.Impl;
3. import java.sql.Connection;
4. import java.sql.SQLException;
6. import javax.annotation.Resource;
7. import javax.sql.DataSource;
9. import org.springframework.stereotype.Component;
11. import cn.edu.hpu.dao.UserDao;
12. import cn.edu.hpu.model.User;
14. @Component("u")
15. public class UserDaoImpl implements UserDao{
17. private DataSource dataSource;
19. public DataSource getDataSource() {
20. return dataSource;
21. }
23. @Resource
24. public void setDataSource(DataSource dataSource) {
25. this.dataSource = dataSource;
26. }

29. public void save(User u) {
30. try {
31. Connection conn=dataSource.getConnection();
32. conn.createStatement().executeUpdate("insert into user values(null,'zhangsan')");
33. conn.close();
34. } catch (SQLException e) {
35. e.printStackTrace();
36. }
37. System.out.println("add success!!");
38. }
39. }

测试：

1. package cn.edu.hpu.service;
2. import org.junit.Test;
3. import org.springframework.beans.factory.BeanFactory;
4. import org.springframework.context.ApplicationContext;
5. import org.springframework.context.support.ClassPathXmlApplicationContext;

8. import cn.edu.hpu.dao.UserDao;
9. import cn.edu.hpu.model.User;

12. public class UserServiceTest {
14. @Test
15. public void testAdd() throws Exception{
16. ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("beans.xml");
18. UserService userService=(UserService)ctx.getBean("userService");
19. userService.add(new User());
20. ctx.destroy();
21. }
22. }

测试结果:  
add success!!  
数据库里多了一条记录:id=1,name='zhangsan'  
  
这就是在spring里面如何设置数据源。

# 【spring框架】关于DataSource(下)

spring连接池的配置，可以有三种连接池的配置(常用):  
i.c3p0  
ii.dbcp  
iii.proxool  
用到哪种去查那种情况的配置即可。  
  
下面我们主要说dbcp。  
除了这么写：

1. <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
2. *<!-- results in a setDriverClassName(String) call -->*
3. <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
4. <property name="url" value="jdbc:mysql://localhost:3306/spring"/>
5. <property name="username" value="root"/>
6. <property name="password" value="1234"/>
7. </bean>

还可以这么写：  
把数据库的信息存进Properties文件  
jdbc.properties文件:  
jdbc.driverClassName=com.mysql.jdbc.Driver  
jdbc.url=jdbc:mysql://localhost:3306/spring  
jdbc.username=root  
jdbc.password=1234

在Spring中可以使用PropertyPlaceHolderConfigure来读取Properties文件的内容

1. <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
2. <property name="locations">
3. <value>classpath:com/foo/jdbc.properties</value>
4. </property>
5. </bean>
7. <bean id="dataSource" destroy-method="close" class="org.apache.commons.dbcp.BasicDataSource">
8. <property name="driverClassName" value="${jdbc.driverClassName}"/>
9. <property name="url" value="${jdbc.url}"/>
10. <property name="username" value="${jdbc.username}"/>
11. <property name="password" value="${jdbc.password}"/>
12. </bean>

测试：

1. package cn.edu.hpu.service;
2. import org.junit.Test;
3. import org.springframework.beans.factory.BeanFactory;
4. import org.springframework.context.ApplicationContext;
5. import org.springframework.context.support.ClassPathXmlApplicationContext;
7. import cn.edu.hpu.dao.UserDao;
8. import cn.edu.hpu.model.User;
10. public class UserServiceTest {
12. @Test
13. public void testAdd() throws Exception{
14. ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("beans.xml");
16. UserService userService=(UserService)ctx.getBean("userService");
17. System.out.println(userService.getClass());
18. User u=new User();
19. u.setUsername("u1");
20. u.setPassword("p1");
21. userService.add(u);
22. ctx.destroy();
23. }
24. }

结果:  
add success!!

存储成功

# [spring 读取properties的两种方法](https://www.cnblogs.com/beyang/p/6074115.html)

一：直接使用context命名空间

<beans xmlns="http://www.springframework.org/schema/beans"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
xmlns:context="http://www.springframework.org/schema/context"  
xmlns:websocket="http://www.springframework.org/schema/websocket"  
xsi:schemaLocation="http://www.springframework.org/schema/beans   
http://www.springframework.org/schema/beans/spring-beans.xsd  
http://www.springframework.org/schema/websocket  
http://www.springframework.org/schema/websocket/spring-websocket.xsd  
http://www.springframework.org/schema/context   
http://www.springframework.org/schema/context/spring-context.xsd">

<context:property-placeholder location="classpath:log4j.properties" />

二：在配置文件中配置PropertyPlaceholderConfigurer

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
 <property name="locations" value="classpath:jdbc.properties" />  
</bean>

实例：

<!-- property-placeholder是一个属性遍历器，定位一个属性文件，属性文件存放的是jdbc一些连接数据 -->

<context:property-placeholder location="classpath:jdbc.properties" />

<!-- 配置数据源 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="${driverClassName}" />

<property name="url" value="${url}" />

<property name="username" value="${username}" />

<property name="password" value="${password}" />

<property name="initialSize" value="${initialSize}" />

<property name="maxActive" value="${maxActive}" />

<property name="maxIdle" value="${maxIdle}" />

<property name="minIdle" value="${minIdle}" />

</bean>

# mybatis,mysql,datasource,dpcb连接池的关系

datasource是配置数据源，顾名思义就是数据的来源，在数据源中可以配置数据是从何而来，可以是mysql，也可以是oracle，所以可以如下配置

<**bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"**>

<**property name="driverClassName"value="${driverClassName}"**/>

<**propertyname="url"value="${jdbcUrl}"**/>

<**propertyname="username"value="${jdbcUsername}"**/>

<**propertyname="password"value="${jdbcPassword}"**/>

<**propertyname="maxActive"value="${maxActive}"**/>

<**propertyname="initialSize"value="${initialSize}"**/>

<**propertyname="maxWait"value="${maxWait}"**/>

<**propertyname="minIdle"value="${minIdle}"**/>

</**bean**>

如上说明数据源是从mysql来的，所以数据源中配置了驱动名称mysql，以及账号密码

如上其实也是配置一种连接池，有dbcp连接池，也有c3p0连接池，如上配置的是dbcp连接池

所以上面也配置了连接池的相关信息，比如最大连接数，初始连接数等

所以如上配置的意思就是：dbcp连接池中的连接就是mysql数据库的连接，mysql数据库就是数据源

**jdbc接口就是从数据源(连接池)中获取连接来访问数据库(mysql,oracle),所以在sqlSessionFactory中需要数据源的信息**，所以可以如下配置：

<**bean id="sqlSessionFactory"class="org.mybatis.spring.SqlSessionFactoryBean"**>

<**propertyname="dataSource"ref="dataSource"**/>

<**propertyname="mapperLocations"value="classpath:mapper/\*.xml"**/>

*<!--<property name="configLocation" value="classpath:mybatis/sqlMapConfig.xml"/>-->*

</**bean**>

如果访问数据库是通过mybatis来访问，那么mybatis底层必然是通过**sqlSessionFactory**来访问的数据库，所以要给mybatis配置**sqlSessionFactory**，所以可以如下配置：

<**beanclass="org.mybatis.spring.mapper.MapperScannerConfigurer"**>

<**propertyname="basePackage"value="com.verify.dao"**/>

<**propertyname="sqlSessionFactoryBeanName"value="sqlSessionFactory"**/>

</**bean**>

mysql的账号驱动等相关信息一般是配置在配置文件中，所以可以通过如下方式来引入配置文件，从而获取配置文件中的参数信息

<**beanclass="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"**>

<**propertyname="location"value="classpath:conf/web-jdbc.properties"**/>

</**bean**>

# SqlSession的作用



# 数据库连接池DataSource的几种整合

## [MyBatis在非Spring环境下第三方DataSource设置-Druid篇](https://www.cnblogs.com/ChenJunHacker/p/6401797.html)

<https://www.cnblogs.com/ChenJunHacker/p/6401797.html>

## [druid+spring配置](https://www.cnblogs.com/shihaiming/p/5981789.html)

<https://www.cnblogs.com/shihaiming/p/5981789.html>

## [spring c3p0的配置](https://www.cnblogs.com/xuguiping/p/6531171.html)

<https://www.cnblogs.com/xuguiping/p/6531171.html>

## [Spring dbcp连接池配置与示例（以及JDBCTemplate的使用）](https://www.cnblogs.com/AnswerTheQuestion/p/6641347.html)

<https://www.cnblogs.com/AnswerTheQuestion/p/6641347.html>

## [Spring使用proxool连接池 管理数据源](https://www.cnblogs.com/liaojie970/p/7241109.html)

<https://www.cnblogs.com/liaojie970/p/7241109.html>

## Spring与HikariCP的结合使用

<https://blog.csdn.net/benben_2015/article/details/78660501>

## Jdbc连接池交个spring管理和事务操作

<https://blog.csdn.net/tianyejun6/article/details/67120639>