MyBatis入门—Getting Started

# MyBatis的jar包只有一个

**mybatis-x.x.x.jar包**。不过，mybatis还提供了一些依赖包在lib文件夹中，比如**日志包**。

此外，由于用什么数据库管理系统不限定，所以使用时，必须**导入相应的数据库驱动包**，如MySQL的驱动包：。

如果使用**Maven管理**的话，需要在pom.xml文件中添加依赖：

在<dependencies>中添加：

<dependency>

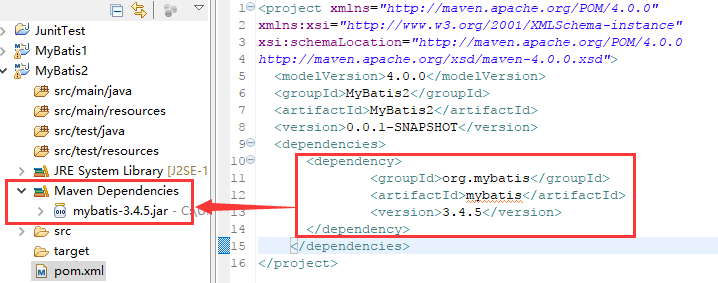
<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>x.x.x</version>

</dependency>

示例：在**pom.xml**添加依赖后,**Maven工程**就会自动下载依赖包。



# 创建SqlSessionFactory的两种方法

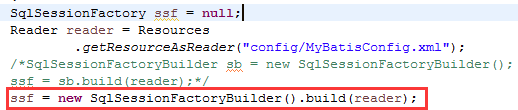
## Building SqlSessionFactory from XML（重点）

## Building SqlSessionFactory without XML

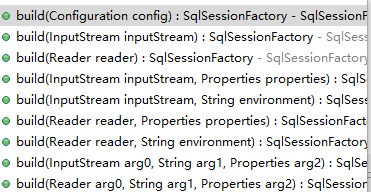
# MyBatis开发的核心：获取SqlSession对象

无论传统的DAO开发方式还是新的Mapper代理开发方式，都是需要获取**SqlSession对象**。前者直接**利用SqlSession对象的方法**进行**增删改查（CRUD）**，而后者通过getMapper方法获取Mapper接口的实现对象，利用**接口中的方法**进行增删改查。

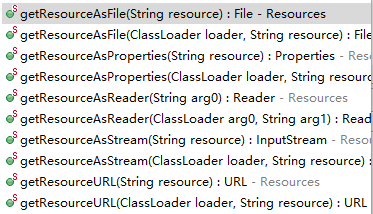
**SqlSession**创建方法：首先创建一个**SqlSessionFactoryBuilder**对象，然后调用**build方法**，与**全局配合文件**关联，创建**SqlSessionFactory**。调用**SqlSessionFactory**的**openSession**方法就可以创建**SqlSession**对象。



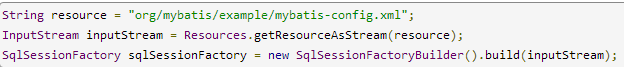
**SqlSessionFactoryBuilder**的build方法。



**MyBatis**中提供了一个类**Resources**，有很多**静态方法**，加载资源。



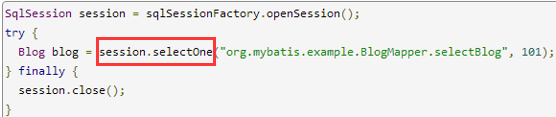
示例：



# 获取SqlSession对象：Acquiring a SqlSession from SqlSessionFactory

## 对于之前的DAO开发方式

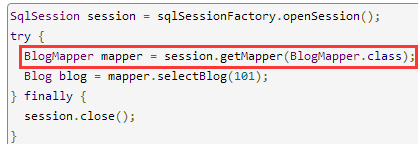
直接利用**SqlSession**对象中的selectOne、selectList、update、delete、insert等方法操作。



## 对于新的Mapper代理开发方式

利用SqlSession对象中的getMapper方法获取Mapper接口的实现对象，通过获取的Mapper代理对象，进行操作。





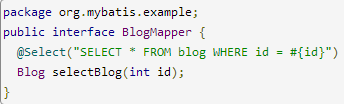
# Building SqlSessionFactory without XML（使用很少）

不写**全局配置文件**，仅仅通过Java代码也可以获取**SqlSessionFactory**对象。

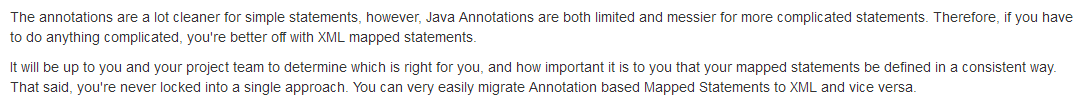
需要使用**Configuration类**。利用SqlSessionFactoryBuilder.buile(Configuration config)加载这个类。还会用到**Environment**、**DataSource**等类。



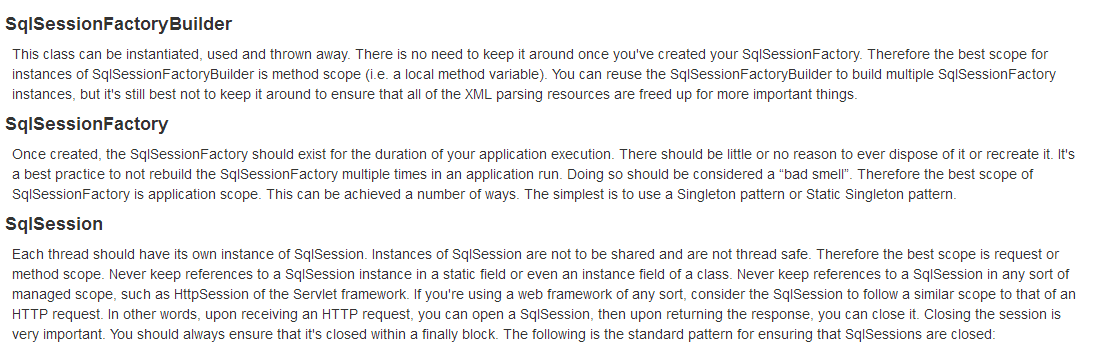
# 利用注解取代mapper.xml文件



这种方法是有限的，一般比较复杂的语句还是通过**mapper.xml文件配置**。



# Scope and Lifecycle： SqlSessionFactoryBuilder、SqlSessionFactory与SqlSession



在MyBatis教程入门，最下端讲述了如何使用这些类。

正确使用**SqlSessionFactoryBuilder**、**SqlSessionFactory**、**SqlSession**、**Mapper**对象十分重要，否则会出现严重的并发问题。

## 对于SqlSessionFactoryBuilder类

一旦创建了SqlSessionFactory，就可以把SqlSessionFactoryBuilder回收掉。尽量不要一直维护着他的存在。

## 对于SqlSessionFactory类，

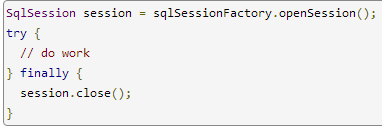
**最好使用单例模式。**在项目工程中，一旦创建了SqlSessionFactory，就应该让其一直生存着。销毁并多次创建SqlSessionFactory是非常坏的。



建议利用单例模式生成**SqlSessionFactory对象**，其单例实现见后面。

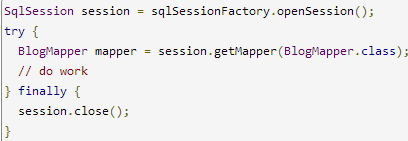
## SqlSession：sql会话



使用完毕一定要关闭，在finally中关闭。  


## 对于Mapper对象（Mapper代理开发方式）

使用完毕后需要关闭sqlSession。



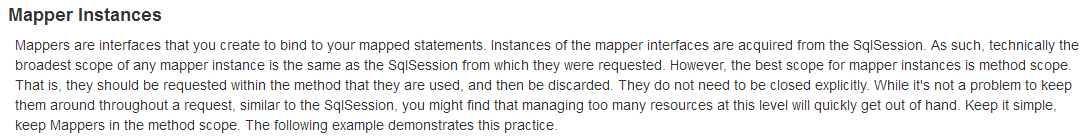
一定要再**finally**中关闭，这样可以做到万无一失。

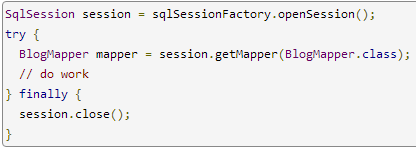
# 生成SqlSession的方法：关键

三种：1. mapper代理，2. 接口实现（DAO），3. 利用注解（Java Annotation）

## Mapper代理开发方式：

利用生成的mapper对象作为**SqlSession**的代理进行操作。





# SqlSessionFactory的单例实现

SqlSessionFactory的单例实现，创建工具类提供方法分别get**SqlSession**和closeSqlSession。

示例：

package com.unisk.utils;

import java.io.IOException;

import java.io.Reader;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class SqlSessionFactoryUtils {

private static ThreadLocal<SqlSession> tl = new ThreadLocal<SqlSession>();

private static SqlSessionFactory ssf = null;

/\*\*

\* 禁止外界通过new的方式获取该对象(构造方法私有化)

\*/

private **SqlSessionFactoryUtils**() { }

/\*\*

\* 静态代码块：加载全局配置文件，并生成SqlSessionFactory

\*/

static {

try {

Reader reader = Resources

.getResourceAsReader("config/MyBatisConfig.xml");

ssf = new SqlSessionFactoryBuilder().build(reader);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 获取SqlSession

\* @return

\*/

public static SqlSession **getSqlSession**() {

SqlSession ss = tl.get();

if (ss == null) {

ss = ssf.openSession();

tl.set(ss);

}

return ss;

}

/\*\*

\* 关闭SqlSession，并与当前线程分离

\*/

public static void **closeSqlSession**() {

//从当前线程中获取SqlSession

SqlSession ss = tl.get();

if (ss != null) {

//关闭sqlSession

ss.close();

//关闭当前线程与SqlSession的关系，目的是让GC尽早回收

tl.remove();

}

}

}