创建mybatis工具类

在创建工具类之前要知道mybatis使用的对象的生命周期

--我们需要知道SqlSessionFactoryBuilder、SqlSessionFactory、SqlSession三个对象的生命周期

--SqlSessionFactoryBuilder对象：
官方文档原话：
--这个类可以被实例化、使用和丢弃，一旦创建了 SqlSessionFactory，就不再需要它了。
因此 SqlSessionFactoryBuilder 实例的最佳作用域是方法作用域（也就是局部方法变量）。 你可以重用 SqlSessionFactoryBuilder 来创建多个 SqlSessionFactory 实例，但最好还是不要一直保留着它，以保证所有的 XML 解析资源可以被释放给更重要的事情。
--这个类是用来创建获取SqlSessionFactory对象的，SqlSessionFactory对象一旦被创建SqlSessionFactoryBuilder就没有用了，可以被垃圾回收器回收销毁了。从官方例子中可以直接看出：
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
从上面官方文档给的代码就知道：
    new出来的SqlSessionFactoryBuilder的对象用完build方法创建SqlSessionFactory对象后就被垃圾回收器回收了
    因为没有引用变量指向这个对象：SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
    而sqlSessionFactory指向的是SqlSessionFactory对象

--SqlSessionFactory对象：
官方文档原话：
--SqlSessionFactory 一旦被创建就应该在应用的运行期间一直存在，没有任何理由丢弃它或重新创建另一个实例。
使用 SqlSessionFactory 的最佳实践是在应用运行期间不要重复创建多次，多次重建 SqlSessionFactory 被视为一种代码“坏习惯”。因此 SqlSessionFactory 的最佳作用域是应用作用域。 有很多方法可以做到，最简单的就是使用单例模式或者静态单例模式。
--很明显该类对象是非常重要的，一个项目只能有一次创建，并且不能随意丢弃和重新创建另一个
如果放在一个工具类里的话，创建SqlSessionFactory对象的语句块需要放到静态语句块中，在项目加载时创建该对象，只执行一次。
SqlSessionFactory是一个接口：--现类是 DefaultSqlSessionFactory
作用：获取SqlSession对象 SqlSession sqlSession =sqlSessionFactory.opensession();
--openSession()方法说明：
    openSession()：方法中无需参数，获取sqlSession的时候会关闭自动提交事务机制
    openSession(boolean)：参数是布尔类型，true表示获取sqlSession的时候会开启自动提交事务机制
                                                false表示获取sqlSession的时候会关闭自动提交事务机制

--SqlSession对象
SqlSessin也是一个接口，该接口中定义了许多操作数据的方法：实现类 DefaultSqlSession
官方文档原话：
--每个线程都应该有它自己的 SqlSession 实例。SqlSession 的实例不是线程安全的，因此是不能被共享的，所以它的最佳的作用域是请求或方法作用域。 绝对不能将 SqlSession 实例的引用放在一个类的静态域，甚至一个类的实例变量也不行。 也绝不能将 SqlSession 实例的引用放在任何类型的托管作用域中，比如 Servlet 框架中的 HttpSession。 如果你现在正在使用一种 Web 框架，考虑将 SqlSession 放在一个和 HTTP 请求相似的作用域中。 换句话说，每次收到 HTTP 请求，就可以打开一个 SqlSession，返回一个响应后，就关闭它。 这个关闭操作很重要，为了确保每次都能执行关闭操作，你应该把这个关闭操作放到 finally 块中。
意思就是：

SqlSession对象不是线程安全的，需要在方法的内部使用，即在一个方法中创建该对象（在执行sql语句前获取SqlSession对象：SqlSession session = sqlSessionFactory.openSession();),使用完毕后关闭sqlsession.close()

每个线程都应该有它自己的 SqlSession 实例，所以一次请求对应一个SqlSession对象，就是说一个请求中不能被创建两个及以上的SqlSession对象，一个线程对应一次事务。不能被共享，就是说该对象不能定义为静态的。

如果在工具类中，确保一次请求只能创建一个SqlSession对象，就是说无论怎么使用openSession方法，创建的都是同一个SqlSession对象，需要把该对象存到ThreadLocal对象local中（这是一个集合类对象），每次调用openSession方法前先判断local中是否有SqlSession对象(local.get()==null)，为null，则执行local.set(sqlSession),然后sqlSession = local.get(),否者直接返回sqlSession。

还需要注意的一点是，在写工具类的关闭SqlSession资源时，除了关闭sqlsession.close()外，还需要将执行local.remove()。理由是执行local.set(sqlSession)方法时，当前这个线程就绑定了local集合，因为Tomcat中存在一个"线程池"，线程结束后是存到该线程池中的，线程结束后如果不移除local集合中的SqlSession对象，下次该线程可能被使用到的时候会继续使用上次的local，也就继续使用上一个SqlSession对象，但是由于上一个SqlSession对象执行了close方法，继续使用就会报错。

一个mybatis工具类：com.studymyself.utils.MyBatisUtil

```
package com.studymyself.utils;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;

public class MyBarisUtil {

    //需要在外边定义，下面的代码好使用
    private static SqlSessionFactory sqlSessionFactory = null;
    //SqlSessionFactory对象只能被创建一次
    static {
        //SqlSessionFactoryBuilder对象只是为了创建SqlSessionFactory对象，所以不需要引用指向
        //SqlSessionFactoryBuilder对象，代码执行完毕该对象就被回收了
        try {
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(Resources.getResourceAsStream("mybatis.xml"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //定义一个集合存储SqlSession对象，确保一次事务（线程、请求）只创建一个
    private static ThreadLocal<SqlSession> local = new ThreadLocal<>();

    /**
     * 获取当前线程的SqlSession对象
     * @return
     */
    public static SqlSession getSqlSession(){
        //先获取local集合中的SqlSession
        SqlSession sqlSession = local.get();
        if (sqlSession == null) {
```

```
            sqlSession = sqlSessionFactory.openSession();
            local.set(sqlSession);
        }
        return sqlSession;
    }

    /**
     * 关闭资源以及移除集合中的sqlSession对象
     */
    public static void close(SqlSession sqlSession){

        if (sqlSession != null) {

            sqlSession.close();

            local.remove();

        }

    }
}
```