主 题：日报

汇报人：左超

日 期：2018年7月25日 星期3

内 容：

* 今日学习了的内容：

MaBatis 缓存机制：

mybatis的缓存机制

mybatis的查询缓存分为一级缓存和二级缓存，一级缓存是SqlSession级别的缓存，二级缓存时mapper级别的缓存，二级缓存是多个SqlSession共享的。

一级缓存：

mybatis的一级缓存是SQLSession级别的缓存，在操作数据库时需要构造SqlSession对象，在对象中有一个HashMap用于存储缓存数据，不同的SqlSession之间缓存数据区域（HashMap）是互相不影响的。

一级缓存的作用域是SqlSession范围的，当在同一个SqlSession中执行两次相同的sql语句时，第一次执行完毕会将数据库中查询的数据写到缓存（内存）中，第二次查询时会从缓存中获取数据，不再去底层进行数据库查询，从而提高了查询效率。需要注意的是：如果SqlSession执行了DML操作（insert、update、delete），并执行commit（）操作，mybatis则会清空SqlSession中的一级缓存，这样做的目的是为了保证缓存数据中存储的是最新的信息，避免出现脏读现象。

当一个SqlSession结束后该SqlSession中的一级缓存也就不存在了，Mybatis默认开启一级缓存，不需要进行任何配置。

注意：Mybatis的缓存机制是基于id进行缓存，也就是说Mybatis在使用HashMap缓存数据时，是使用对象的id作为key，而对象作为value保存

二级缓存：

二级缓存是mapper级别的缓存，使用二级缓存时，多个SqlSession使用同一个Mapper的sql语句去操作数据库，得到的数据会存在二级缓存区域，它同样是使用HashMapper进行数据存储，相比一级缓存SqlSession，二级缓存的范围更大，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。

二级缓存是多个SqlSession共享的，其作用域是mapper的同一个namespace，不同的SqlSession两次执行相同的namespace下的sql语句，且向sql中传递的参数也相同，即最终执行相同的sql语句，则第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次查询时会从缓存中获取数据，不再去底层数据库查询，从而提高查询效率。

Mybatis默认没有开启二级缓存，需要在setting全局参数中配置开启二级缓存。

mybatis框架内部的执行器体系。

复习了class类和反射机制

设计模式：

动态代理---MyBatis（反射机制）

**package** Review.Se.DynamicProxy;

**public** **interface** Person {

//上交班费

**void** giveMoney();

}

**package** Review.Se.DynamicProxy;

**public** **class** Student **implements** Person{

**private** String name;

**public** Student(String name) {

**this**.name = name;

}

**public** **void** giveMoney() {

System.***out***.println(name + "交班费50元");

}

}

**package** Review.Se.DynamicProxy;

**import** java.lang.reflect.InvocationHandler;

**import** java.lang.reflect.Method;

**public** **class** StuInvocationHander<T> **implements** InvocationHandler {

T target ;

**public** StuInvocationHander(T target) {

**this**.target = target;

}

@Override

**public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {

System.***out***.println("代理执行"+method.getName()+"方法");

Object result = method.invoke(target, args);

System.***out***.println("after calling");

**return** result;

}

}

**package** Review.Se.DynamicProxy;

**import** java.lang.reflect.InvocationHandler;

**import** java.lang.reflect.Proxy;

**public** **class** ProxyTest {

**public** **static** **void** main(String[] args) {

Person zhangsan = **new** Student("zhangsan");

InvocationHandler stuhandler = **new** StuInvocationHander<Person>(zhangsan);

Person stuProxy = (Person) Proxy.*newProxyInstance*(Person.**class**.getClassLoader(),

**new** Class<?>[] {Person.**class**},stuhandler);

stuProxy.giveMoney();

System.***out***.println(**int**.**class**);

}

}

遗留的问题：动态代理源代码看不懂--InvocationHandler接口和 le Proxy.*newProxyInstance()* 只会用。

【学习内容真实可靠，给自己的评分和完成度客观真实，不谦虚，不夸大】