# 简介

侧重点：SQL

优点：规避JDBC代码、参数设置、获取结果集

Mybatis与Hibernate的区别以及优缺点？

Mybatis：

半自动化

侧重点：SQL语句

优点：SQL和Java编程分割，功能边界清晰，一个专注业务，一个专注数据。

缺点：

Hibernate：

全自动

侧重点：持久化对象

优点：自动生成SQL语句。对持久化对象的操作自动解析成SQL语句

缺点：

# HelloWorld

# configuration全局配置文件

## settings 设置

<settings>

<setting name=*"cacheEnabled"* value=*"true"* />

<setting name=*"lazyLoadingEnabled"* value=*"true"* />

<setting name=*"multipleResultSetsEnabled"* value=*"true"* />

<setting name=*"useColumnLabel"* value=*"true"* />

<setting name=*"useGeneratedKeys"* value=*"false"* />

<setting name=*"autoMappingBehavior"* value=*"PARTIAL"* />

<setting name=*"autoMappingUnknownColumnBehavior"*

value=*"WARNING"* />

<setting name=*"defaultExecutorType"* value=*"SIMPLE"* />

<setting name=*"defaultStatementTimeout"* value=*"25"* />

<setting name=*"defaultFetchSize"* value=*"100"* />

<setting name=*"safeRowBoundsEnabled"* value=*"false"* />

<setting name=*"mapUnderscoreToCamelCase"* value=*"false"* />

<setting name=*"localCacheScope"* value=*"SESSION"* />

<setting name=*"jdbcTypeForNull"* value=*"OTHER"* />

<setting name=*"lazyLoadTriggerMethods"*

value=*"equals,clone,hashCode,toString"* />

</settings>

## typeAliases 类型别名

具体类

<typeAliases>

<typeAlias type=*"xxx.bean.xxx"* alias=*"emp"*/>

</typeAliases>

包

<typeAliases>

<package name=*"xxx.xxx.bean"* />

</typeAliases>

别名==类名小写

注解

@Alias("xxx")

Class Xxx

常见的 Java 类型的别名

|  |  |
| --- | --- |
| 别名 | 映射类型 |
| \_x | byte、long、short、int、double、float、boolean |
| 首字母小写 | String、Byte、Long、Short、Integer、Double、Float、Boolean、Date、BigDecimal、Object、Map、HashMap、List、ArrayList、Collection、Iterator |
| decimal | BigDecimal |
| int | Integer |

## typeHandlers 类型处理器

## objectFactory 对象工厂

## plugins 插件

## properties 属性文件

<properties resource=*"dbconfig.properties"* />

## environments 环境

### environment 环境变量

### transactionManager 事务管理器

### dataSource 数据源

<environments default=*"dev\_mysql"*>

<environment id=*"dev\_mysql"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${mysql.driver}"*/>

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

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

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

</dataSource>

</environment>

<environment id=*"dev\_oracle"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${oracle.driver}"*/>

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

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

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

</dataSource>

</environment>

</environments>

## databaseIdProvider 数据库厂商标识

<databaseIdProvider type=*"DB\_VENDOR"*>

<property name=*"MySQL"* value=*"mysql"* />

<property name=*"Oracle"* value=*"oracle"* />

<property name=*"SQL Server"* value=*"sqlServer"* />

</databaseIdProvider>

## mappers 映射器

<mappers>

<!-- 相对config.xml的路径 -->

<mapper resource=*"org/mybatis/builder/AuthorMapper.xml"* />

<!-- 绝对路径 -->

<mapper url=*"file:///var/mappers/AuthorMapper.xml"*/>

<!-- 指定接口 -->

<mapper class=*"org.mybatis.builder.AuthorMapper"*/>

类名==映射文件名

<!-- 包下接口 -->

<package name=*"org.mybatis.builder"*/>

类名==映射文件名

</mappers>

# 映射文件

mapper.xml对应接口

mapper文件名==接口名

namespace==接口全类名

语句==方法

语句id==方法名

语句parameterType==方法参数

语句resultType==方法返回值

## namespace

接口全类名

## insert、delete、update

id=*""*

parameterType=*""*

databaseId=*""*

*mybatis-config.xml配置databaseIdProvider*

timeout=*""*

*在抛出异常之前，驱动程序等待数据库返回请求结果的秒数*

*unset*

flushCache=*"true"*

*语句调用后清空本地缓存和二级缓存，默认值：true*

statementType

*STATEMENT【语句】*

*CALLABLE【存储过程、函数】*

*PREPARED【预处理语句 】【默认】*

*<!--insert，update独有-->*

useGeneratedKeys=*"true"*

*使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键*

*false【默认】*

keyProperty=*"id"*

*设置键对应的属性*

keyColumn=*""*

*设置键对应的列*

自增主键【适用于mysql】

<insert id=*""* parameterType=*""*

useGeneratedKeys=*"true"* keyProperty=*"id"* databaseId=*"mysql"*>

insert into tbl\_employee(last\_name,email,gender)

values(#{lastName},#{email},#{gender})

</insert>

非自增主键【适用于Oracle】

<insert id=*"addEmp"* databaseId=*"oracle"*>

<selectKey

keyProperty=*"id"*

order=*"BEFORE" 【】*

*BEFORE 先selectKey元素，然后执行插入语句*

*AFTER 先执行插入语句，然后selectKey元素*

resultType=*"Integer"*

>

select EMPLOYEES\_SEQ.nextval from dual

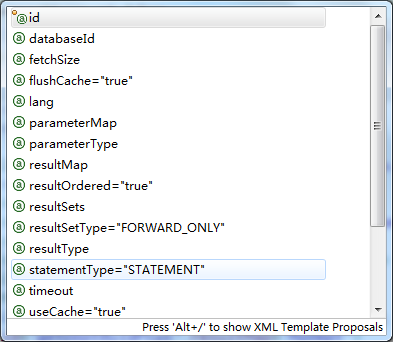
</selectKey>

insert into employees(EMPLOYEE\_ID,LAST\_NAME,EMAIL)

values(#{id},#{lastName},#{email})

</insert>

## selsect



fetchSize

unset【默认】依赖驱动

批量返回的结果行数

userCache

true【默认】：结果将被二级缓存

resultOrdered

true

false

resultSets

resultSetType

FORWARD\_ONLY

SCROLL\_SENSITIVE

SCROLL\_INSENSITIVE

## ResultMap

autoMappingBehavior

PARTIAL

mapUnderscoreToCamelCase=true

<resultMap type=*""* id=*""* autoMapping=*"true"* extend=*""*>

<id/>

<result/>

<constructor></constructor> 【集合】

<collection property=*""*></collection> 【关联】

<association property=*""*></association>

<discriminator javaType=*""*></discriminator> 【鉴别器：根据结果值决定使用哪个ResultMap】

column*【指定列】*

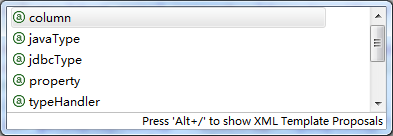
property *【指定属性】*

javaType

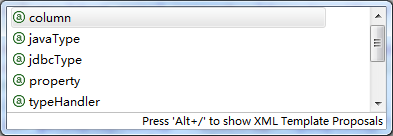
jdbcType

typeHande

<id />



<result />



关联

<association>

Property

column

javaType

jdbcType

typeHander

autoMapping

自动映射

columnPrefix

同一resultMap添加前缀进行映射

foreignColumn

外键列

notNullColumn

resultMap

resultSet

select

*设置查询*

fetchType=*"lazy"*

*lazy【懒加载】*

*eager【立即加载】*

集合

<collection property=*""* ></collection>

Property

column

javaType

jdbcType

typeHander

autoMapping

ofType

columnPrefix

foreignColumn

notNullColumn

resultMap

resultSet

select

fetchType=*"lazy"*

<constructor >

<arg />

typeHander

column

javaType

jdbcType

name

resultMap

select

<idArg />

typeHander

column

javaType

jdbcType

name

resultMap

select

</constructor>

<discriminator>

column

javaType

jdbcType

typeHander

<case>

value

resultMap

resultType

cache

cache-ref

resultMap

sql

select

## 内置参数

### \_parameter

单个：表示该参数

\_parameter

public List<Employee> getEmpsTestInnerParameter(String lastName);

<if test=*"\_parameter!=null"*>

where last\_name like #{\_parameter}

</if>

多个：参数会被封装为一个map；

\_parameter代表这个map

\_parameter.xxx

public List<Employee> getEmpsTestInnerParameter(Employee employee);

<if test=*"\_parameter!=null"*>

where last\_name like #{\_parameter.lastName}

</if>

### \_databaseId

配置的databaseIdProvider值

<databaseIdProvider type=*"DB\_VENDOR"*>

<property name=*"MySQL"* value=*"mysql"*/>

<property name=*"Oracle"* value=*"oracle"*/>

<property name=*"SQL Server"* value=*"sqlserver"*/>

</databaseIdProvider>

## 批量插入

### MySQL

方案一

insert into table() values(),()....

<insert id=*""*>

insert into tbl\_employee(

<include refid=*"insertColumn"*></include>

)

values

<foreach collection=*"emps"* item=*"emp"* separator=*","*>

(#{emp.lastName},#{emp.email},#{emp.gender},#{emp.dept.id})

</foreach>

</insert>

方案二

insert into table() values();

insert into table() values();

...

数据库连接属性：allowMultiQueries=true;

<insert id=*""*>

<foreach collection=*"emps"* item=*"emp"* separator=*";"*>

insert into tbl\_employee(

<include refid=*"insertColumn"*></include>

)

values

(#{emp.lastName},#{emp.email},#{emp.gender},#{emp.dept.id})

</foreach>

</insert>

### Oracle

方案一：多个insert放在begin-end

begin

insert into table() values();

insert into table() values();

...

end

<foreach collection=*"emps"* item=*"emp"* open="begin" close="end;">

insert into table() values();

</foreach>

方案二：利用中间表

insert into table()

select table\_seq.nextval, xxx... from(

select 'test\_a\_01'.xxx... from dual

union

select 'test\_a\_02'.xxx from dual

...

)

insert into table(...)

<foreach collection=*"emps"* item=*"emp"* separator=*"union"*

open=*"select table\_seq.nextval,... from("* close=*")"*>

select #{emp.lastName} lastName,#{emp.email} email from dual

</foreach>

## 批量保存

## SQL片段

<sql id=*"sql\_id"*>*sql\_val,*$*{pro}*</sql>

不能使用#{}方式

<include refid=*"sql\_id"*>

<property name=*"pro"* value=*"pro\_val"*/>

</include>

结果：sql\_id,*pro\_val*

# 缓存机制

## 一级缓存【Session本地缓存】

Session级别

session.flush()或session.close()操作清空缓存

### 一级缓存失效的四种情况

不同session

同一session，不同查询

同一session，相同查询，期间执行增删改或手动清空缓存其他操作

## 二级缓存【namespace全局缓存】

namespace级别

session关闭后数据保存在二级缓存

### 使用步骤

（1）config配置

<setting name="cacheEnabled" value="true"/>

<setting name="localCacheScope" value="SESSION"/>

（2）mapping配置

<cache></cache>

（3）POJO实现序列化接口

## <cache>详解

### type

设置自定义缓存全类名【实现Cache接口即可】

例如：第三方缓存EhCache，*org.mybatis.caches.ehcache.EhcacheCache*

### eviction

回收策略

LRU【默认】

least recently used【最近最少使用的】

移除不常使用的对象

FIFO

first in，first out【先进先出】

按对象进入缓存的顺序来移除

SOFT

软引用

基于垃圾回收器状态和软引用规则

WEAK

弱引用

基于垃圾收集器状态和弱引用规则

### flushInterval

刷新缓存的间隔

默认不清空

单位：秒

### readOnly

true

只读

缓存数据只可读，不可修改

优缺点:

不安全，速度快

false

非只读

缓存数据可以修改

利用序列化和反序列化技术克隆一份新的数据

优缺点:

安全，速度慢

### size

缓存存放多少元素

## 第三方缓存整合【EhCache】

### （1）导入EhCache的jar包

ehcache-core-2.6.8.jar

mybatis-ehcache-1.0.3.jar

slf4j-api-1.6.1.jar

slf4j-log4j12-1.6.2.jar

### （2）编写EhCache.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:noNamespaceSchemaLocation=*"../config/ehcache.xsd"*>

<!-- 磁盘保存路径 -->

<diskStore path=*"D:\44\ehcache"* />

<defaultCache

<!-- 必须 -->

maxElementsInMemory=*"10000"*

*在内存中缓存的元素的最大数目*

maxElementsOnDisk=*"10000000"*

*在磁盘上缓存的元素的最大数目*

*0表示无穷大*

eternal=*"false"*

*是否永远不过期*

overflowToDisk=*"true"*

*内存溢出是否将过期的元素缓存到磁盘上*

<!-- 可选 -->

timeToIdleSeconds=*"120"*

*允许最大空闲时间，前后访问时间间隔超过改值，数据就会被删除*

*0表示无限*

timeToLiveSeconds=*"120"*

*有效生命期*

*0表示无限*

diskExpiryThreadIntervalSeconds=*"120"*

*磁盘缓存的清理线程运行间隔，默认是120秒。*

memoryStoreEvictionPolicy=*"LRU"*

*回收策略*

*LRU【最近最少使用的】*

*FIFO【先进先出】*

*SOFT【软引用】*

*WEAK【弱引用】*

diskSpoolBufferSizeMB

*设置DiskStore(磁盘缓存)的缓存区大小*

*默认是30MB*

diskPersistent

*在VM重启的时候是否启用磁盘保存EhCache中的数据，默认是false*

>

</defaultCache>

</ehcache>

### （3）配置cache标签

方案一：添加*EhcacheCache*类型

<cache type=*"org.mybatis.caches.ehcache.EhcacheCache"*></cache>

方案二：参照其他命名空间的缓存

<cache-ref namespace=*"namespace"*/>

缓存的作用域

localCacheScope

session【默认】

statement

## 缓存相关设置

### （1）全局setting的cacheEnable

二级缓存关闭【false】开启【true】

### 全局setting的localCacheScope

SESSION【默认】

一级缓存

STATEMENT

禁用一级缓存

### 标签的useCache

false

一级 可用

二级 不可用

### 标签的flushCache

true

一级清空

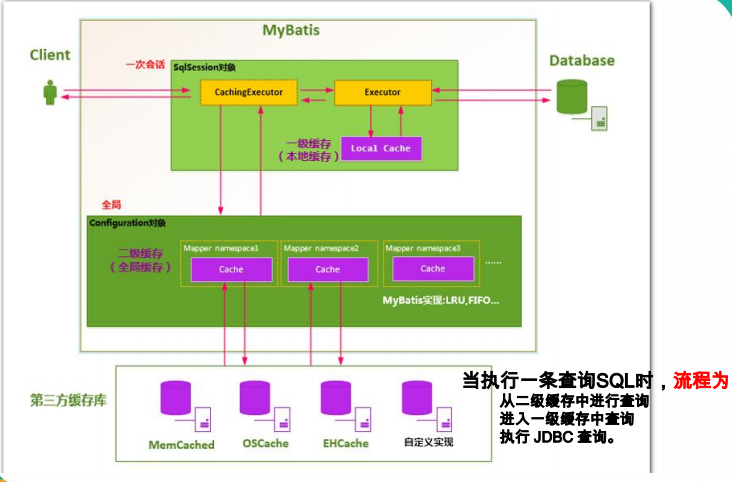
二级 清空

### sqlSession.clearCache()

一级 清空

二级 不清空

## 缓存机制图



# Mybatis-Spring整合

## 目的

1、spring管理所有组件，包括mapper的实现类。

service==>Dao

@Autowired:自动注入mapper

2、spring用来管理事务，spring声明式事务

## Jar包

mybatis

spring

mybatis-spring

c3p0

mysql-connector

## mybatis-config.xml

全局配置

<settings>

<setting>

数据库厂家ID

<databaseIdProvider type=*"DB\_VENDOR"*>

## applicationContext.xml

### 数据库配置文件

<context:property-placeholder

location=*"classpath:dbconfig.properties"* />

### c3p0数据源

<bean id=*"dataSource"*

class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*

>

<property name=*"jdbcUrl"* value=*"${jdbc.url}"*></property>

<property name=*"driverClass"* value=*"${jdbc.driver}"*></property>

<property name=*"user"* value=*"${jdbc.username}"*></property>

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

</bean>

### 事务管理

<bean id=*"dataSourceTransactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

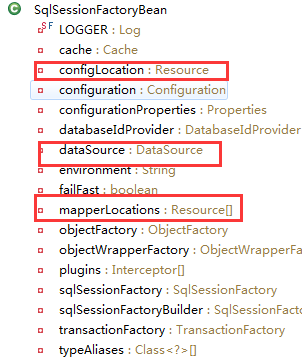
</bean>

<!-- 开启基于注解的事务 -->

<tx:annotation-driven

transaction-manager=*"dataSourceTransactionManager"*/>

### SqlSessionFactory



<bean id=*"sqlSessionFactoryBean"*

class=*"org.mybatis.spring.SqlSessionFactoryBean"*

>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- 全局配置文件的位置 -->

<property name=*"configLocation"*

value=*"classpath:mybatis-config.xml"*></property>

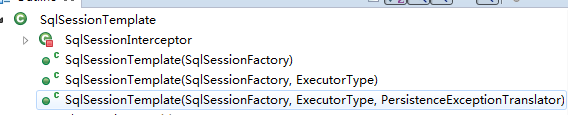
<!-- mapper文件的位置-->

<property name=*"mapperLocations"*

value=*"classpath:mybatis/mapper/\*.xml"*></property>

</bean>

### SqlSession



<bean id=*"sqlSession"* class=*"org.mybatis.spring.SqlSessionTemplate"*>

<constructor-arg name=*"sqlSessionFactory"*

ref=*"sqlSessionFactoryBean"*></constructor-arg>

<constructor-arg name=*"executorType"*

value=*"BATCH"*></constructor-arg>

</bean>

### mapper映射

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"basePackage"*

value=*"com.atguigu.mybatis.dao"*></property>

</bean>

<mybatis-spring:scan base-package=*"com.atguigu.mybatis.dao"*/>

## 测试

方式一：

@Autowired

**private** SqlSession sqlSession;

EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);

方式二：

@Autowired

**private** EmployeeMapper employeeMapper;

employeeMapper.getEmps();

# 逆向工程

## xml配置映射

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration>

<context id=*""* targetRuntime=*"MyBatis3"*>

...

</context>

</generatorConfiguration>

1. <jdbcConnection>【数据库连接】

<jdbcConnection

driverClass=*""*

connectionURL=*""*

password=*""*

userId=*""*

></jdbcConnection>

1. <javaTypeResolver>【Java类型解析器】

<javaTypeResolver

type=*""【解析器类org.mybatis.generator.api.JavaTypeResolver】*

>

<property name=*"forceBigDecimals"* value=*"false"* />

</javaTypeResolver>

*forceBigDecimals false|true*

false

如果比例大于零，或者长度大于18，则将使用java.math.BigDecimal类型

如果比例为零，长度为10到18，那么Java类型解析器将替代java.lang.Long。

如果比例为零，长度为5到9，那么Java类型解析器将替代java.lang.Integer。

如果比例为零，并且长度小于5，那么Java类型解析器将替代java.lang.Short。

true

类型为DECIMAL或NUMERIC

Java类型解析器将始终使用java.math.BigDecimal

1. <javaModelGenerator>【JavaBean生成策略】

<javaModelGenerator

targetPackage=*"" 【目标包名】x.x.x*

targetProject=*"" 【目标工程】*

*.\src\x*

*\projectName\src\x:*

>

<property name=*"enableSubPackages"* value=*"true"* />

<property name=*"trimStrings"* value=*"true"* />

</javaModelGenerator>

*enableSubPackages true|false*

*trimStrings true|false*

*constructorBased*

*immutable*

*rootClass*

1. <sqlMapGenerator>【SQL生成策略】

<sqlMapGenerator

targetPackage=*""*

targetProject=*""*

>

<property name=*"enableSubPackages"* value=*"true"* />

</sqlMapGenerator>

*enableSubPackages*

1. <javaClientGenerator>【map接口生成位置】

<javaClientGenerator

targetPackage=*""*

type=*""*

targetProject=*""*

>

<property name=*"enableSubPackages"* value=*"true"* />

</javaClientGenerator>

1. <table>【映射数据表】

<table tableName=*""* domainObjectName=*""* >

<property>

<generatedKey>

<domainObjectRenamingRule>

<columnRenamingRule>

<columnOverride>

<ignoreColumn>

</table>

## 运行

### cmd命令

java

-jar mybatis-generator-core-x.x.x.jar

-configfile \temp\generatorConfig.xml

-overwrite

java

-cp mybatis-generator-core-x.x.x.jar org.mybatis.generator.api.ShellRunner -configfile generatorConfig.xml

-overwrite

### Java代码

List<String> warnings = **new** ArrayList<String>();

**boolean** overwrite = **true**;

File configFile = **new** File("mbg.xml");

ConfigurationParser cp = **new** ConfigurationParser(warnings);

Configuration config = cp.parseConfiguration(configFile);

DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);

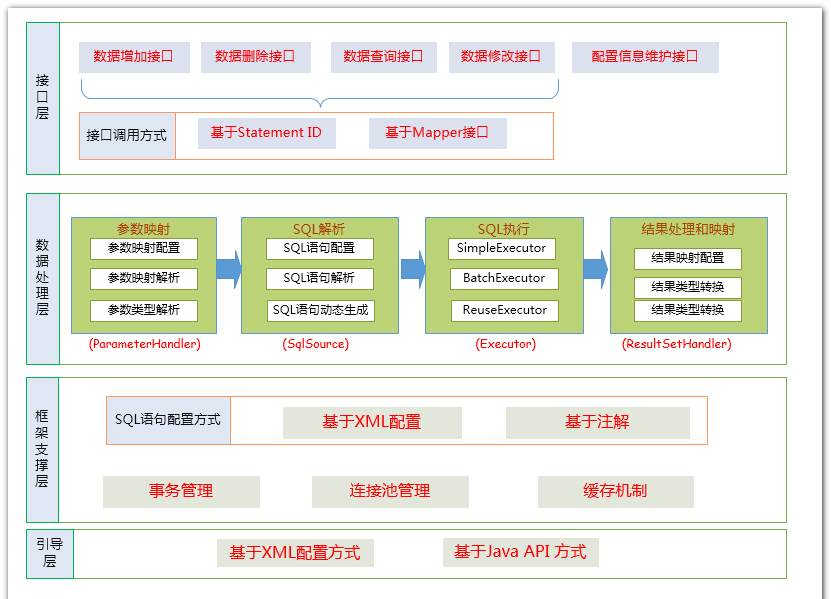
MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config,

callback, warnings);

myBatisGenerator.generate(**null**);

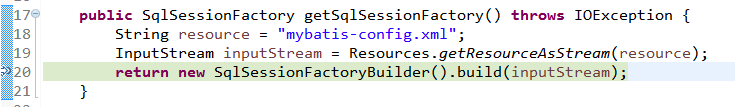
# 运行原理

## 分层结构

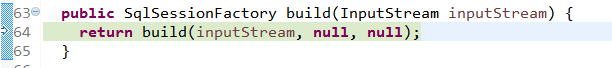


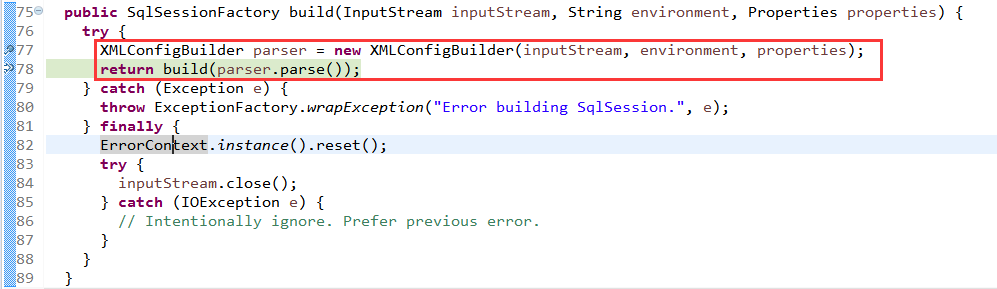
## 运行流程

### 获取sqlSessionFactory对象



SqlSessionFactotyBuilder.build

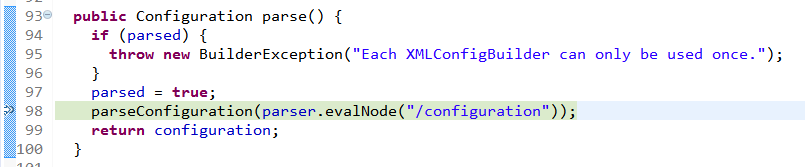




XMLConfigBuilder

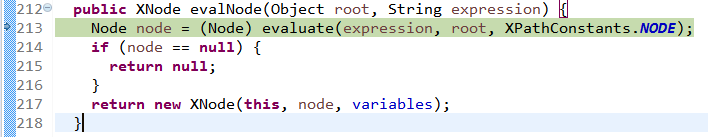


XMLConfigBuilder.parse

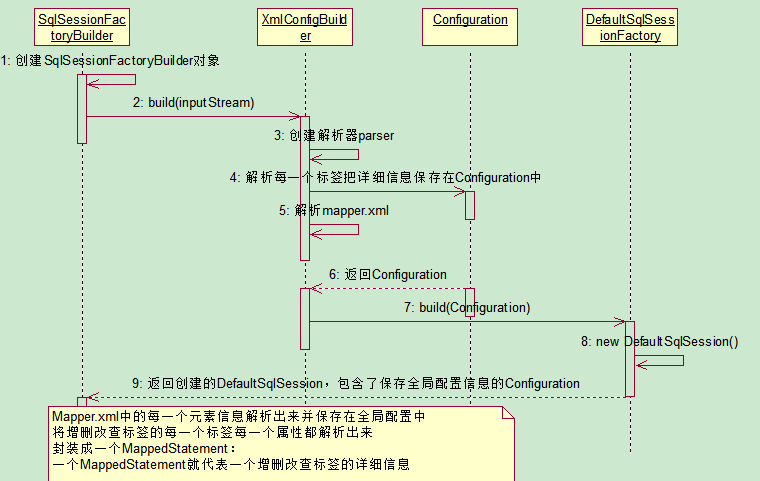


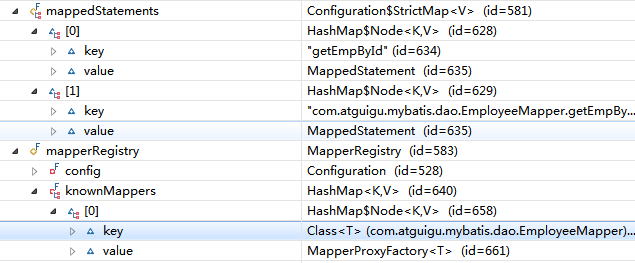
XPathParser



SqlSessionFactotyBuilder



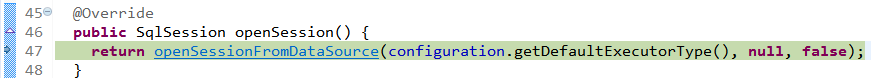


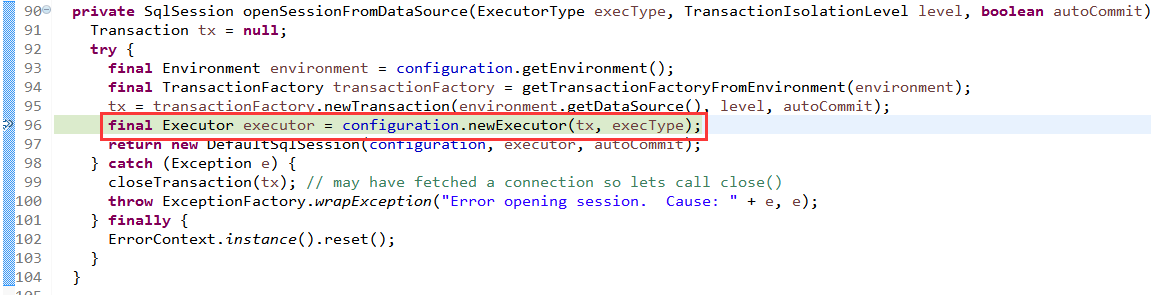


### 获取sqlSession对象

SqlSession openSession = sqlSessionFactory.openSession();

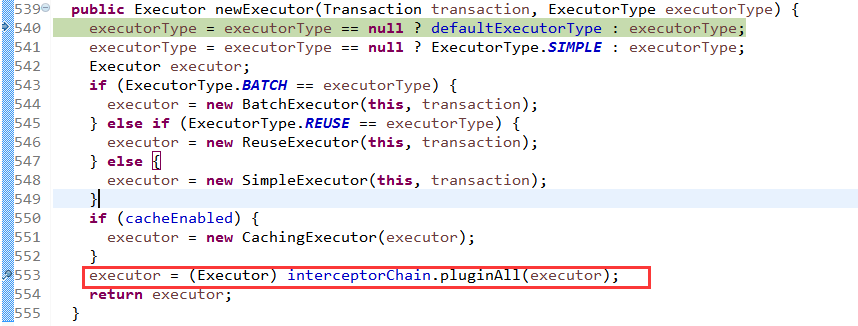
DefaultSqlSessionFactory



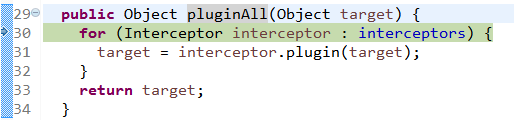


Configuration

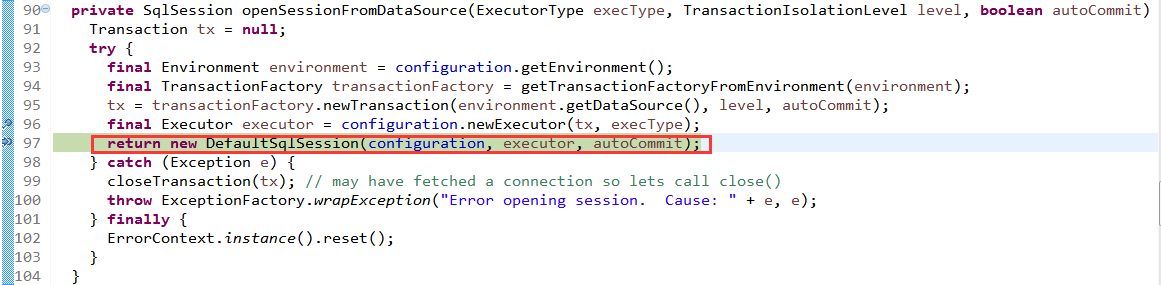
**protected** ExecutorType defaultExecutorType = ExecutorType.***SIMPLE***;

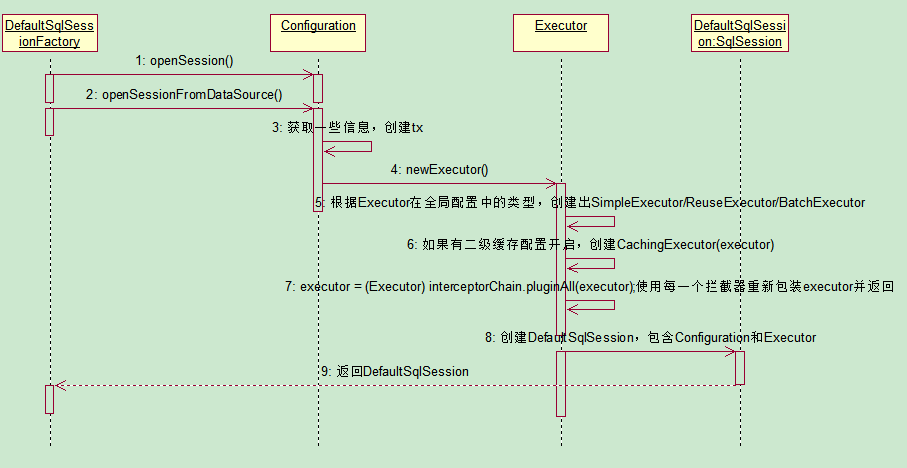


InterceptorChain



DefaultSqlSessionFactory

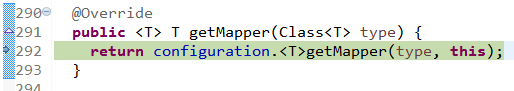




### 获取接口的代理对象（MapperProxy）

EmployeeMapper mapper = openSession.getMapper(EmployeeMapper.**class**);

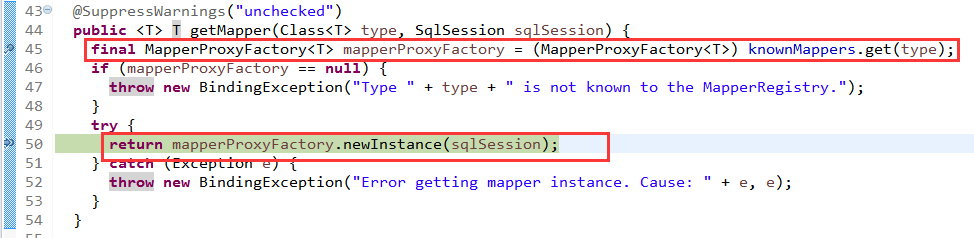
DefaultSqlSession



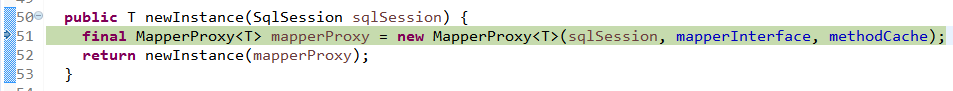
Configuration



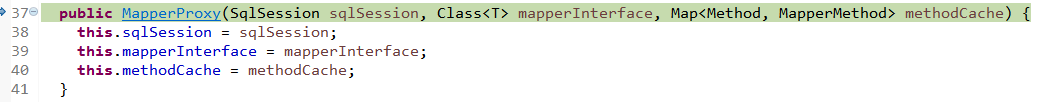
MapperRegistry



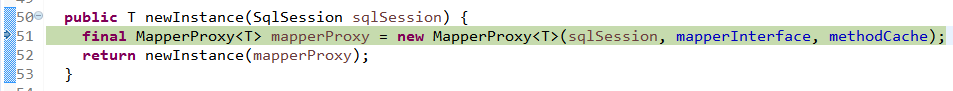
MapperProxyFactory

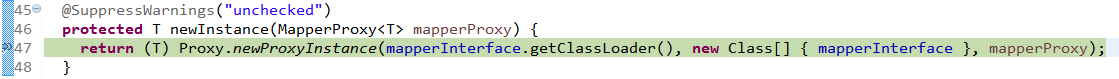


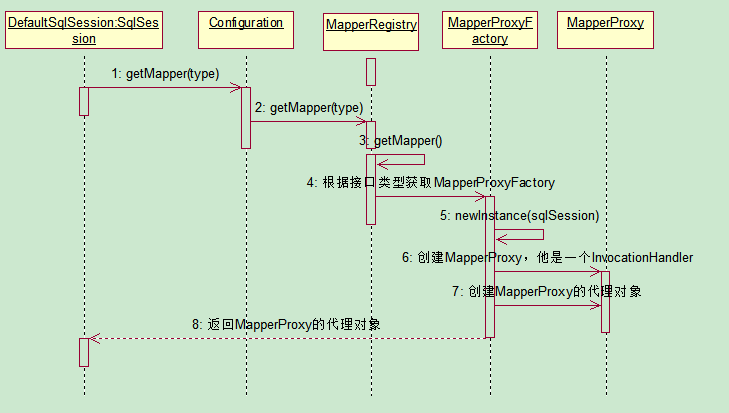
MapperProxy



MapperProxyFactory

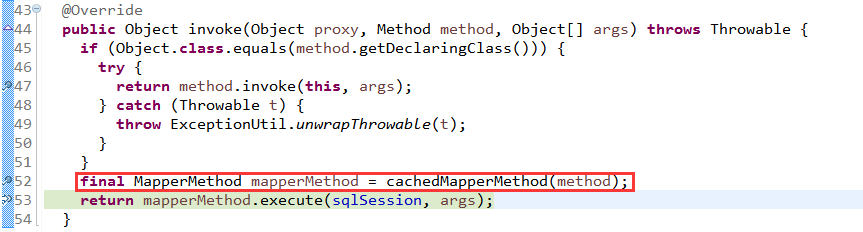


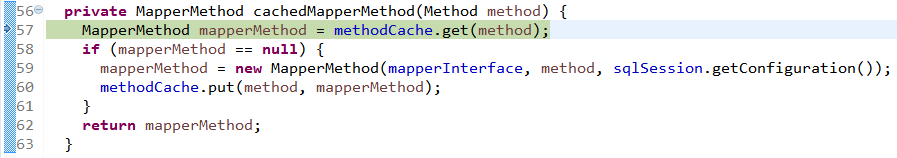


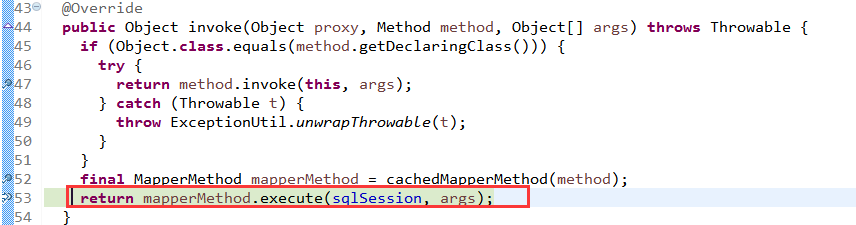


### 执行增删改查方法

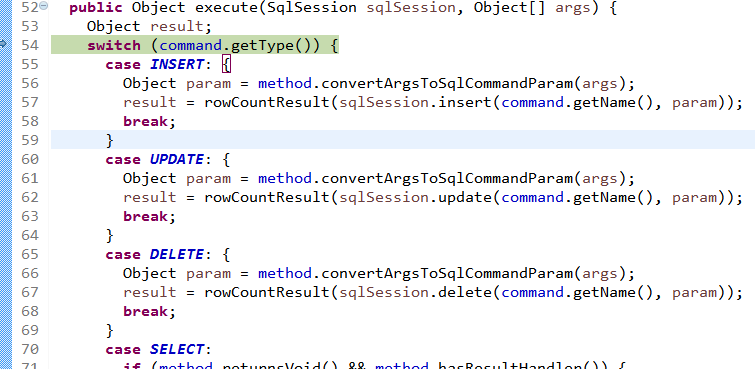
MapperProxy<T> **implements** InvocationHandler

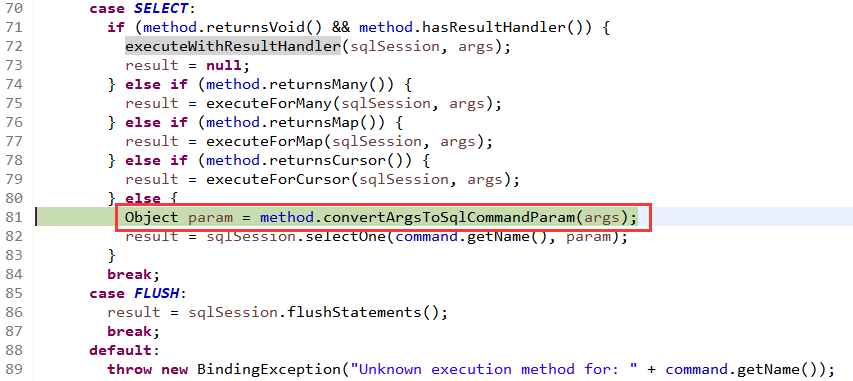






MapperMethod

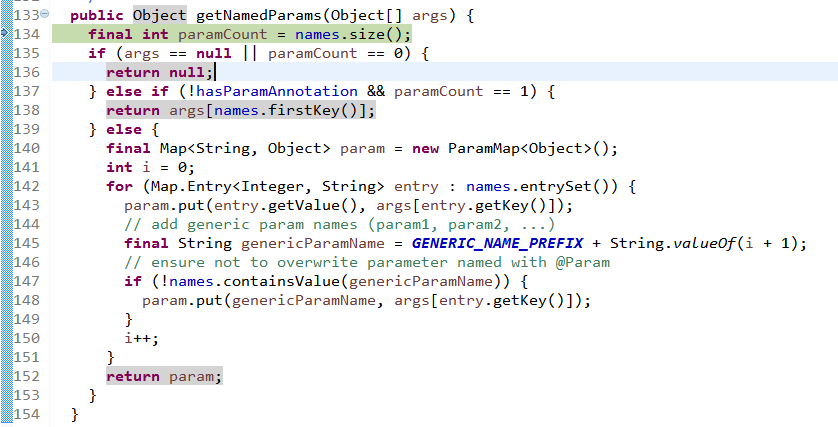




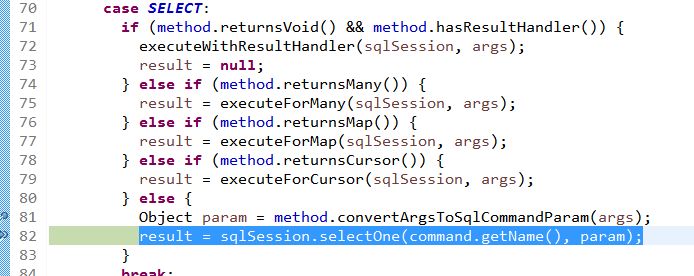
MethodSignature



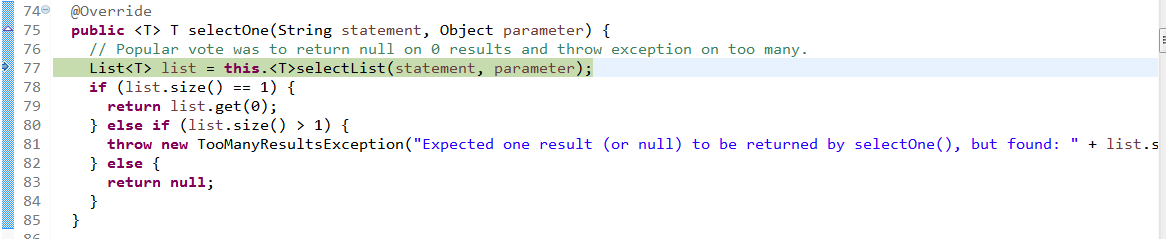
ParamNameResolver

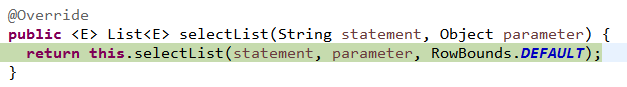


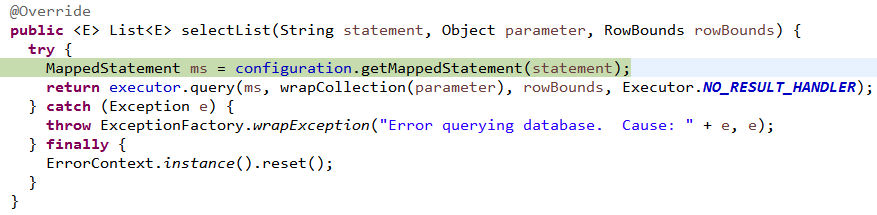
MapperMethod.execute



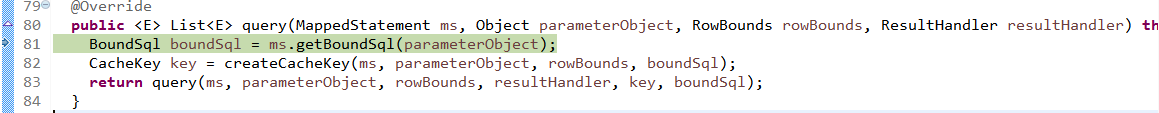
DefaultSqlSession



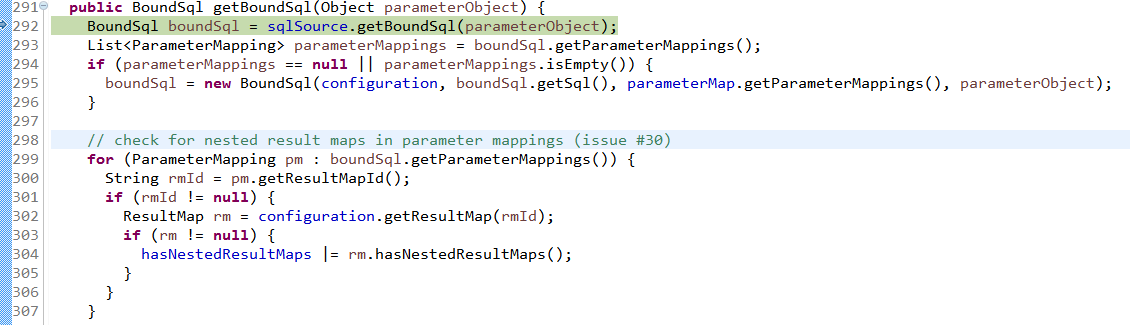




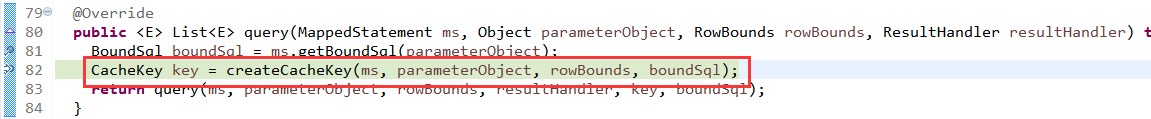
CachingExecutor

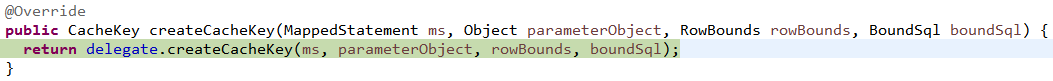


MappedStatement



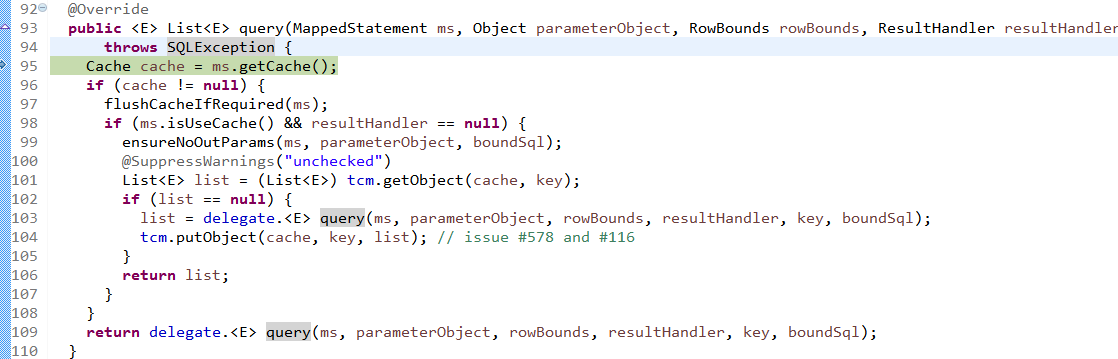
CachingExecutor

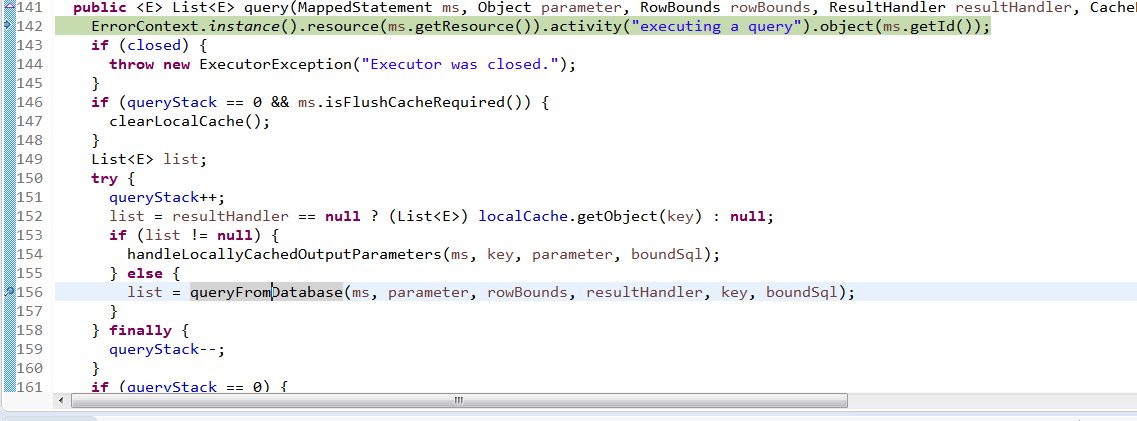


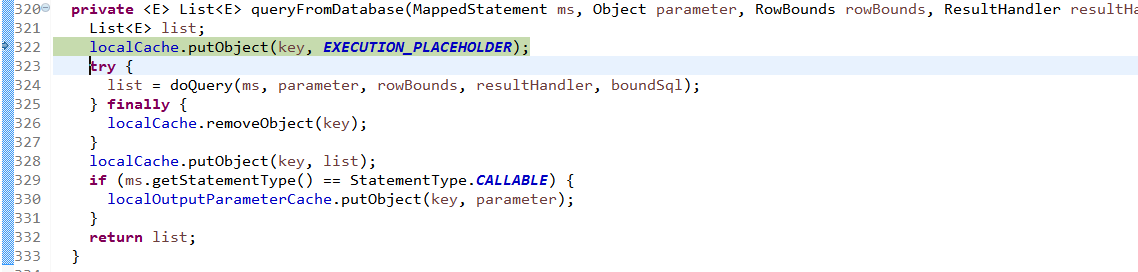


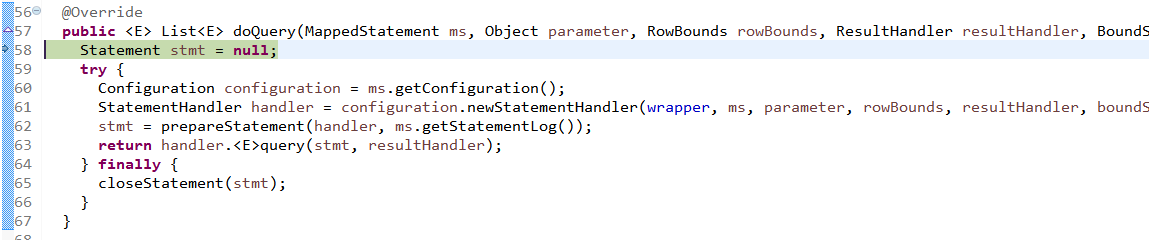
SimpleExecutor(BaseExecutor).createCacheKey()

CachingExecutor.query(MappedStatement, Object, RowBounds, ResultHandler, CacheKey, BoundSql)

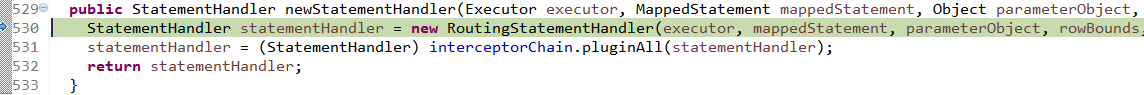
SimpleExecutor(BaseExecutor).query(MappedStatement, Object, RowBounds, ResultHandler, CacheKey, BoundSql) line: 142

SimpleExecutor(BaseExecutor).queryFromDatabase(MappedStatement, Object, RowBounds, ResultHandler, CacheKey, BoundSql) line: 322

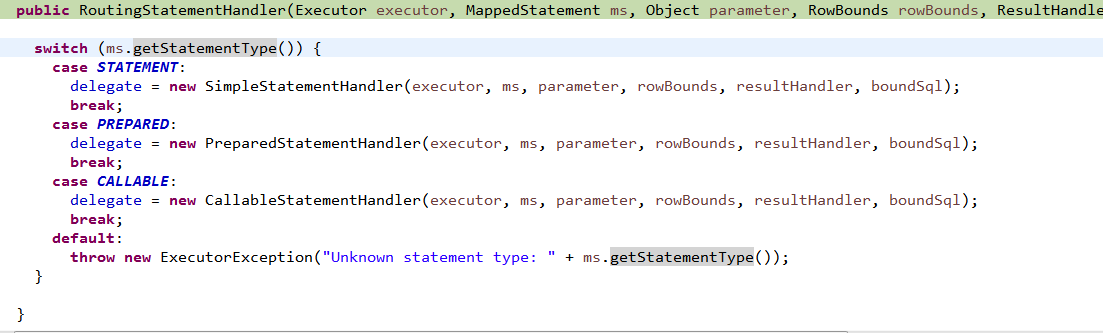
SimpleExecutor.doQuery(MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 58



Configuration.newStatementHandler(Executor, MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 530



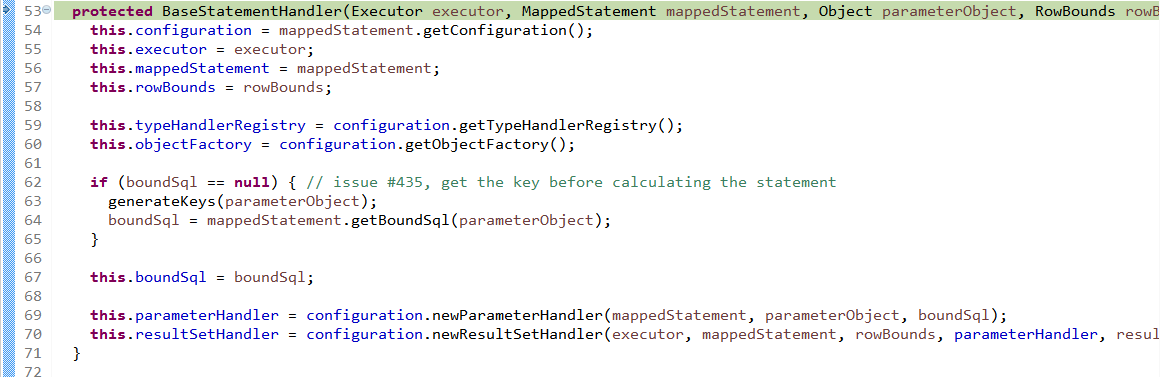
RoutingStatementHandler.<init>(Executor, MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 39



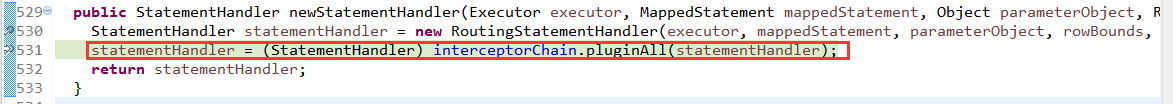
PreparedStatementHandler.<init>(Executor, MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 40



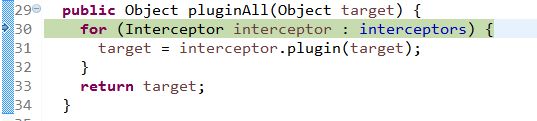
PreparedStatementHandler(BaseStatementHandler).<init>(Executor, MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 53



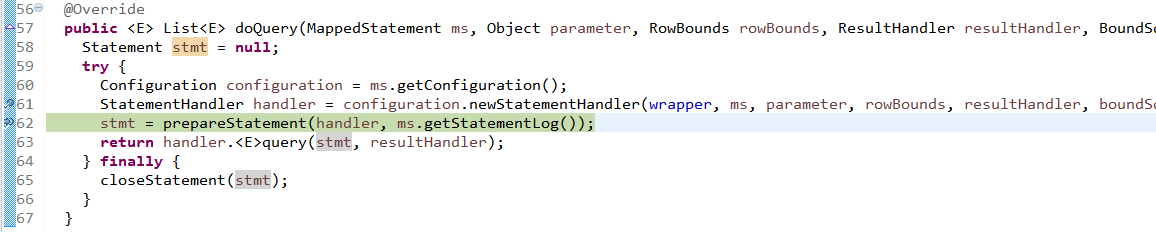
Configuration.newStatementHandler(Executor, MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 531



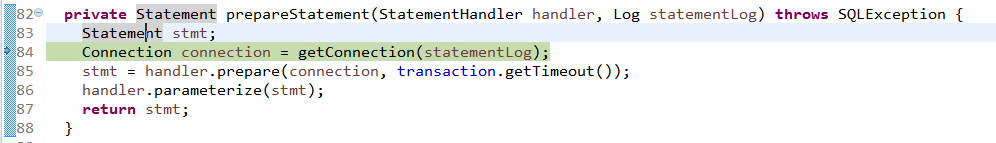
InterceptorChain.pluginAll(Object) line: 30

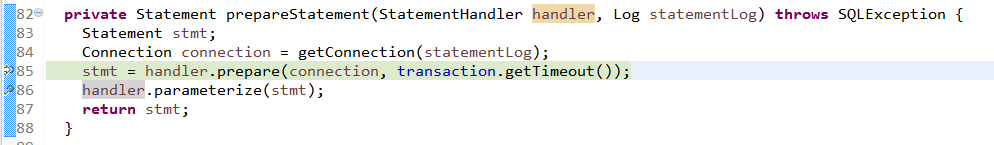


SimpleExecutor.doQuery(MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 61

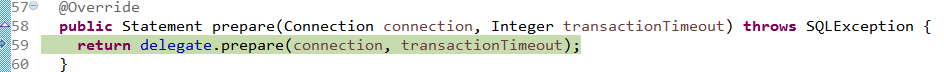


SimpleExecutor.prepareStatement(StatementHandler, Log) line: 84

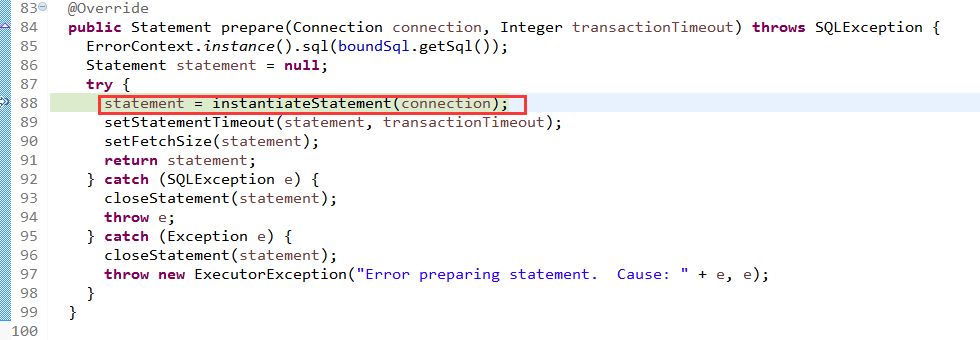




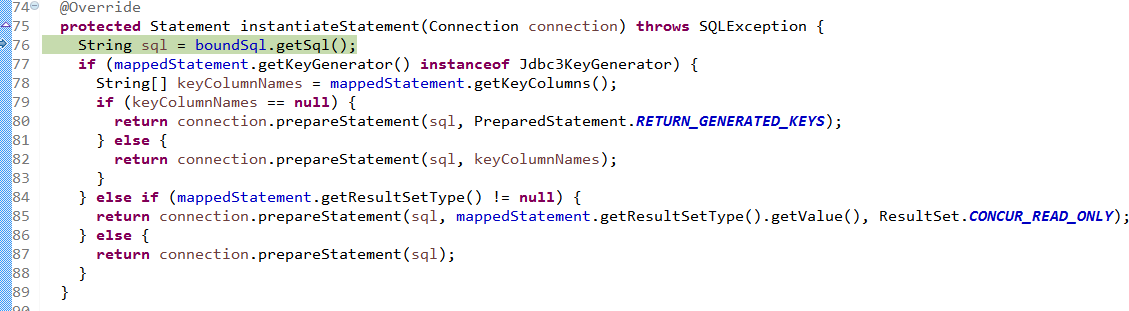
RoutingStatementHandler.prepare(Connection, Integer) line: 59



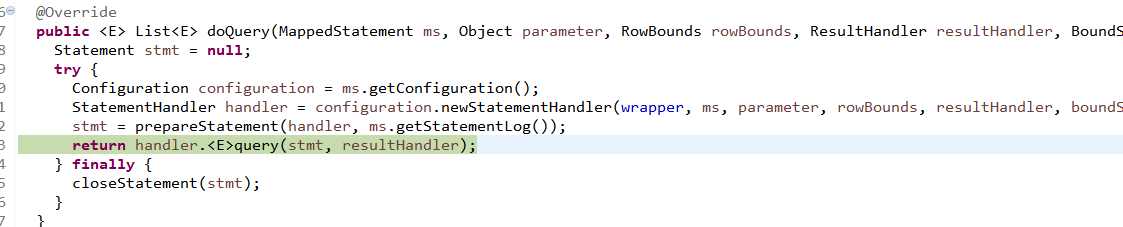
PreparedStatementHandler(BaseStatementHandler).prepare(Connection, Integer) line: 85



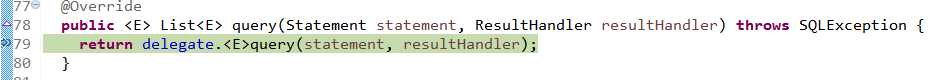
PreparedStatementHandler.instantiateStatement(Connection) line: 76



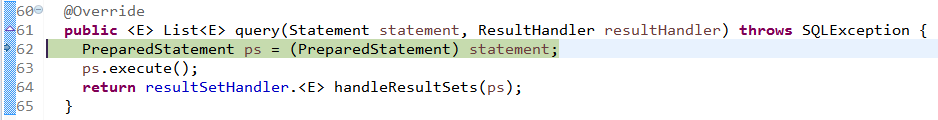
SimpleExecutor.doQuery(MappedStatement, Object, RowBounds, ResultHandler, BoundSql) line: 63



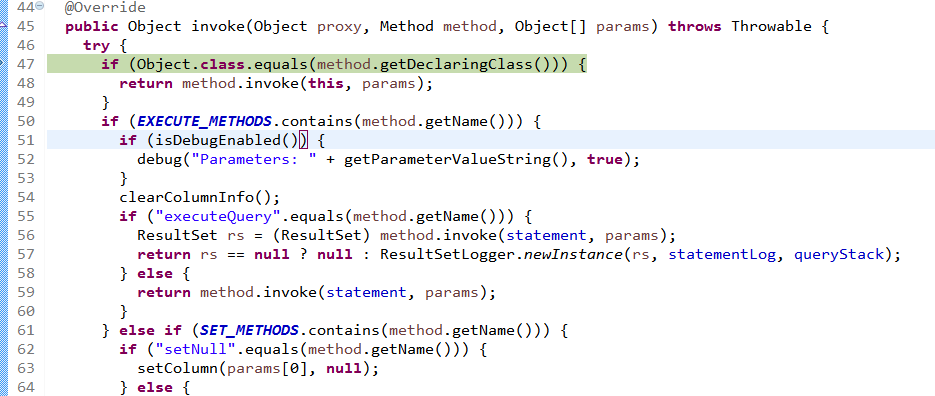
RoutingStatementHandler.query(Statement, ResultHandler) line: 79



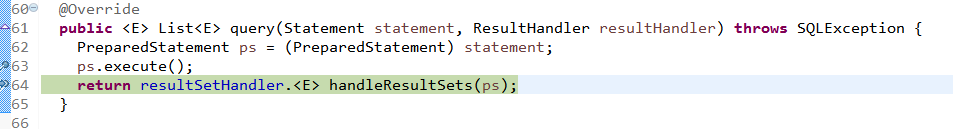
PreparedStatementHandler.query(Statement, ResultHandler) line: 62



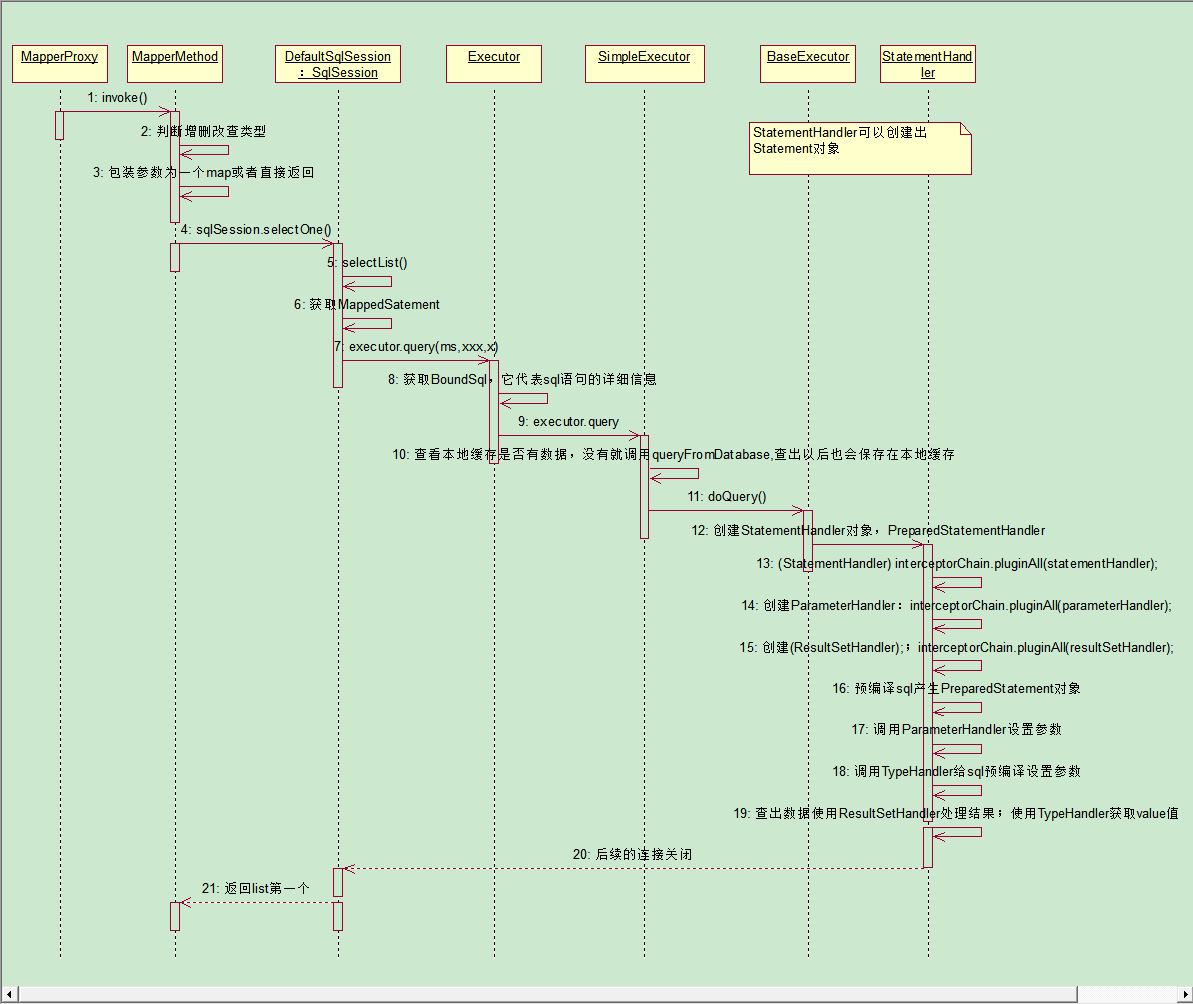
PreparedStatementLogger.invoke(Object, Method, Object[]) line: 47

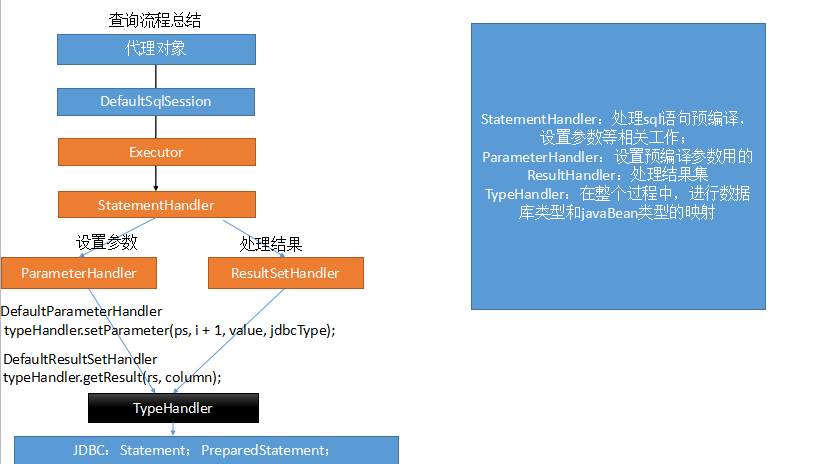


PreparedStatementHandler.query(Statement, ResultHandler) line: 64



DefaultResultSetHandler.handleResultSets(Statement) line: 171





## 四大对象

### ParameterHandler

### SqlSource

### Executor

### ResultSetHandler

# 插件开发

# 拓展

## 分页

## 批量

## 存储过程

## 自定义类型处理器

# QBC查询【了解】

Query By Criteria

SQL

select id, last\_name, email, gender, d\_id from tbl\_employee

WHERE ( last\_name like ? and gender = ? ) or email like "%e%"

Criteria

EmployeeMapper mapper = openSession.getMapper(EmployeeMapper.**class**);

EmployeeExample example = **new** EmployeeExample();

Criteria criteria = example.createCriteria();

criteria.andLastNameLike("%e%");

criteria.andGenderEqualTo("1");

Criteria criteria2 = example.createCriteria();

criteria2.andEmailLike("%e%");

example.or(criteria2);

List<Employee> list = mapper.selectByExample(example);

# 参考

<http://www.mybatis.org/mybatis-3/zh/>

<http://www.mybatis.org/generator/>