



2018

Mybatis分享

分享人：王进（大辰java后台）



Mybatis简介

1

Mybatis核心内容

2

实现原理浅析

3

代码实践

4

目录

CONTENTS



一、Mybatis简介

- **MyBatis 是什么？**

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，建立 POJO 与数据库对象之间的一种关联。

- **为什么选择MyBatis？**

演变过程：

1.传统jdbc

读取配置->加载驱动->获取Connection->得到Statement,SQL预处理->通过Statement发送SQL执行->返回结果ResultSet->使用ResultSet读取数据，映射到POJO对象上->关闭数据库资源

通过jdbc，开发量较大，预处理设置参数和返回结果映射需要手动完成，sql注入，需要人为的关闭数据库链接资源，没有对链接进行有效的管理等

2.ORM框架（表对应类，字段对应类的属性，记录对应对象）

通过映射文件与POJO关系映射，数据库记录与对象一一对应，简化持久层的操作的框架。

1)Hiberante:完全关系映射，不用太关注sql操作简单，但是因为是属性全映射，属性太多会影响性能，不能动态sql,多表关联和复杂sql支持差，不能有效支撑存储过程，无法优化sql

2)Mybatis:POJO+SQL+映射规则，有效的解决了上面的缺点，但是需要较强的SQL能力，与Morphia操作mongo相比，在添加属性的时候要格外小心，一不小心就无法属性映射成功

总之 MyBatis 专注于SQL本身，是一个足够灵活的DAO层解决方案；适用于互联网项目的需求。



二、核心内容

预热：

Mybatis一般的业务代码实现：

- 1.编写Mapper接口
- 2.通过注解和Mapper.xml编写sql
- 3.业务层通过Mapper接口调用自定义的接口方法

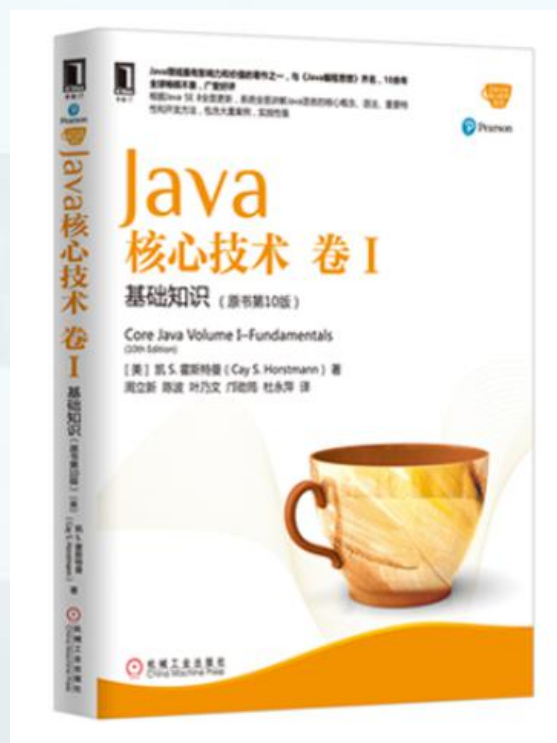
如：`List<String> allEnvelopIds = activityRedEnvelopeMapper.findAllEnvelopIdsByAid(aid);`

如何实现让一个自定义的Mapper接口在业务方法中调用自定义Mapper接口方法返回正确的数据？

主要用到技术：java注解/泛型/反射/动态代理，设计模式（单例，工厂，建造者，代理，适配器，责任链，装饰器，模板，命令），缓存，连接池等

1、java注解/泛型/反射/动态代理

动态代理: <http://www.importnew.com/26166.html>





2、Configuration

```
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3onfig.dtd">
<configuration>
  <!-- autoMappingBehavior should be set in each test case -->
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC">
        <property name="" value=""/>
      </transactionManager>
      <dataSource type="POOLED">
        <property name="driver" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:mem:automapping"/>
        <property name="username" value="sa"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/apache/ibatis/autoconstructor/AutoConstructorMapper.xml"/>
  </mappers>
</configuration>
```



1) Configuration类

上面配置文件的运行时表示，是Mybatis在进行数据库时的全局配置对象，程序启动的时候就初始化好，很多功能都依赖于这个配置里面的属性

Mybatis Configuration源码 部分属性：

protected Environment environment; TransactionFactory, DataSource

protected sdefaultExecutorType = ExecutorType.SIMPLE; sqlSession操作的时候所选择的执行器类型

protected Properties variables = new Properties(); configuration里面的配置属性

protected ObjectFactory objectFactory = new DefaultObjectFactory();属性映射

protected ObjectWrapperFactory objectWrapperFactory = new DefaultObjectWrapperFactory();

protected final MapperRegistry mapperRegistry = new MapperRegistry(this); addMappers 代理对象工厂

protected final InterceptorChain interceptorChain = new InterceptorChain(); 插件

protected final TypeHandlerRegistry typeHandlerRegistry = new TypeHandlerRegistry(); 类型处理器


protected final TypeAliasRegistry typeAliasRegistry = new TypeAliasRegistry(); 别名

protected final Map<String, MappedStatement> mappedStatements = new StrictMap<>("");

protected final Map<String, Cache> caches = new StrictMap<>("Caches collection");

protected final Map<String, ResultMap> resultMaps = new StrictMap<>("Result Maps collection");

protected final Map<String, ParameterMap> parameterMaps = new StrictMap<>("Parameter Maps collection");



常用配置

config xml文件包含的一些常用属性:配置 (properties), 设置(settings), 别名(typeAlias), 类型处理器(typeHandler), 对象工厂 (ObjectFactory), 插件 (plugins), 配置环境 (environments), 映射器 (mappers)

- properties:一些配置属性, 会被解析到Configuration的variables属性里面, 顺序方法参数指定>url指定>配置文件里面的key-value,SqlSessionFactoryBuilder源码查看propertiesElement可以知道
- settings:MyBatis 的运行时行为, 通常默认就好, 有特殊需要修改
- typeAlias: 指定别名, 存在的意义在于用来减少类完全限定名的冗余
- typeHandler: MyBatis 在预处理语句 (PreparedStatement) 中设置一个参数或者结果集中取出一个值时, 都会用类型处理器将获取的值以合适的方式转换成 Java 类型, 通常默认就好
- ObjectFactory:使用一个对象工厂 (ObjectFactory) 创建结果对象的新实例。
- plugins:MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用。默认情况下, MyBatis 允许使用插件来拦截的方法调用

包括: (通常是拦截StatementHandler)

Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)

StatementHandler (prepare, parameterize, batch, update, query)

ParameterHandler (getParameterObject, setParameters)

ResultSetHandler (handleResultSets, handleOutputParameters)

- environments:指定了数据源, 连接池, 事务属性
- mappers: 指定映射器,主要用来初始化两个东西, MapperRegistry , MappedStatement

SqlSessionFactoryBuilder生成sqlSessionFactory的时候需要选解析配置xml文件, 解析的过程会先实例化

Configuration然后给sqlSessionFactory单实例 (同一个数据源) 的全局对象, 可以把配置类对象理解为一个全局的上下文。

3、Mybatis核心组建

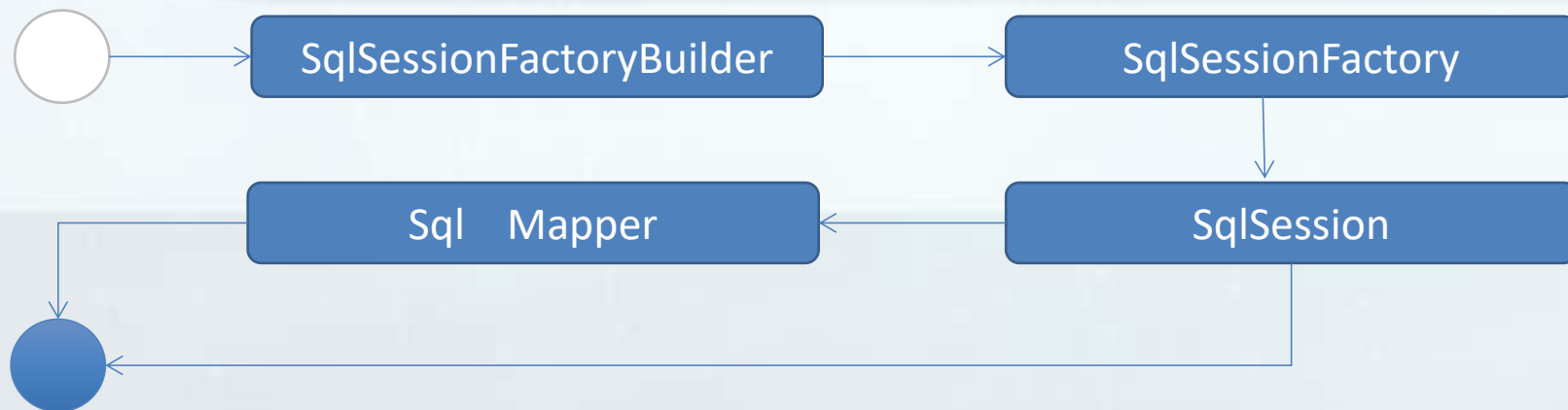
1)SqlSessionFactoryBuilder: 会根据配置信息或者代码生成SqlSessionFactory


2)SqlSessionFactory: 用工厂来参数SqlSession会话

3)SqlSession: 类似于JDBC的Connection,获取Mapper,发送SQL去执行并返回结果

4)Sql Mapper: 定义参数、描述缓存、描述SQL语句、定义查询结果和POJO的映射关系

关联关系:





生命周期:

SqlSessionFactoryBuilder:利用XML或者java编码获得资源来构建SqlSessionFactory,通过它可以构建多个SessionFactory。它的作用就是一个构建器,一旦我们构建了SqlSessionFactory,它的作用就已经完结了,失去了存在的意义。所以它的生命周期只存在于方法的内部。

SqlSessionFactory:作用是创建SqlSession,而SqlSession就是一个会话,相当于JDBC中的Connection对象。每次应用程序访问数据库,我们都需要SqlSessionFactory创建SqlSession,所以SqlSessionFactory应该在MyBatis应用的整个生命周期中。而如果我们多次创建同一个数据库的SqlSessionFactory,SqlSessionFactory会打开更多的数据库连接资源,那么连接资源就很快会被耗尽。因此SqlSessionFactory的责任是唯一的,它的责任就是创建SqlSession,所以应该采用单利模式。正确的做法是使得每一个数据库只对应一个SqlSessionFactory,管理好数据库资源的分配,避免过多的Connection被消耗。

SqlSession:一个会话,相当于JDBC的一个Connection对象,它的生命周期应该是在请求数据库处理事务的过程中。它是一个线程不安全的对象,在涉及多线程的时候我们需要特别小心,操作数据库需要注意其隔离级别,数据库锁等高级特效。此外,每次创建的SqlSession都必须及时关闭它,它的长期存在会使数据库连接池的活动资源减少,对系统性能的影响太大。它存活于一个应用的请求和操作,可以执行多条SQL,保证事务的一致性。

Mapper: Mapper是一个接口,而没有具体的实现类,通过产生一个Mapper的代理对象来是发送SQL,然后返回我们需要的结果,或者执行SQL从而修改数据库的数据,因此它应该在一个SqlSession事务方法之内,是一个方法级别的东西



4、SqlSession下的四大对象

1)Executor: 执行器是用来完成SqlSession指定的增删改查操作的，真正进行与数据库交互的对象；

- SimpleExecutor -- SIMPLE 就是普通的执行器。
- ReuseExecutor -- 执行相同的sql时就可以使用已经存在的预处理语句（prepared statements）
- BatchExecutor --它是批量执行器

2)StatementHandler: 使用Statement（PreparedStatement）向数据库发送SQL执行

- BaseStatementHandler: 一个抽象类，只是实现了一些不涉及具体操作的方法
- RoutingStatementHandler: 路由器，根据配置文件来路由选择具体实现类SimpleStatementHandler、CallableStatementHandler和PreparedStatementHandler
- SimpleStatementHandler: 就是直接使用普通的Statement对象，这样每次执行SQL语句都需要数据库对SQL进行预编译
- PreparedStatementHandler: 使用PreparedStatement执行，虽然初次创建PreparedStatement时开销比较大，但在多次处理SQL时只需要初始化一次，可以有效提高性能
- CallableStatementHandler: 使用CallableStatement执行，CallableStatement是用来执行存储过程的。

3)ParameterHandler: 用于对sql参数的处理，根据TypeHandler对参数进行设值

typeHandler.setParameter(preparedStatement, i , value, jdbcType);

4)ResultSetHandler: 对执行数据库操作后返回的结果集进行封装处理



5、SQL Mapper (<http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html>)

主要元素:

- select:定义查询语句, 常用

```
<select id="selectPerson" parameterType="int" parameterMap="废弃" resultType="hashmap"
resultMap="personResultMap" flushCache="false" useCache="true" timeout="10000" fetchSize="256"
statementType="PREPARED" resultSetType="FORWARD_ONLY">
```
- insert:插入语句

```
<insert id="insertAuthor" parameterType="domain.blog.Author" flushCache="true"
statementType="PREPARED" keyProperty="" keyColumn="" useGeneratedKeys="" timeout="20">
```
- update:更新语句

```
<update id="updateAuthor" parameterType="domain.blog.Author" flushCache="true"
statementType="PREPARED" timeout="20">
```
- delete:删除语句

```
<delete id="deleteAuthor" parameterType="domain.blog.Author" flushCache="true"
statementType="PREPARED" timeout="20">
```
- parameterMap:定义select,insert,update,delete元素的参数映射关系, 已废弃
- sql:定义一部分sql, 然后到各个地方引用它
- resultMap:用来定义数据库返回结果集与对象属性的对应关系
- cache:开启命名空间的二级缓存
- cache-ref: 从另一个命名空间引用缓存配置。



6、动态SQL (<http://www.mybatis.org/mybatis-3/zh/dynamic-sql.html>)

主要元素: (基于代码的动态SQL, 没有xml简单, 条件复杂, SQL的构造逻辑会很繁琐)

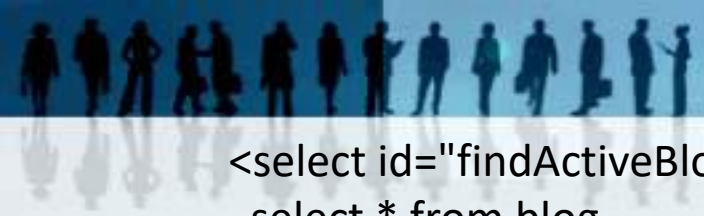
- **if:** 条件分支判断

```
<select id="findActiveBlogWithTitleLike"  resultType="Blog">
    select * from blog WHERE state = 'ACTIVE'
    <if test="title != null"> AND title like #{title} </if>
</select>
```

- **choose (when, otherwise):** 多条件分支判断

```
<select id="findActiveBlogLike"  resultType="Blog">
    select * from blog WHERE state = 'ACTIVE'
    <choose>
        <when test="title != null"> AND title like #{title} </when>
        <when test="author != null and author.name != null"> AND author_name like #{author.name} </when>
        <otherwise> AND featured = 1</otherwise>
    </choose>
</select>
```

- **trim (where, set):** 辅助处理一些元素的拼装问题



```
<select id="findActiveBlogLike"  resultType="Blog">
  select * from blog
  <where>
    <if test="state != null">AND state = #{state}  </if>
    <if test="title != null"> AND title like #{title}  </if>
    <if test="author != null and author.name != null"> AND author_name like #{author.name}  </if>
  </where>
</select>
<update id="updateAuthorIfNecessary">
  update Author
  <set>
    <if test="username != null">username=#{username},</if>
    <if test="password != null">password=#{password},</if>
    <if test="email != null">email=#{email},</if>
    <if test="bio != null">bio=#{bio}</if>
  </set>
  where id=#{id}
</update>
<trim prefix="WHERE" prefixoverride="AND |OR">
  <if test="name != null and name.length()>0"> AND name=#{name}</if>
  <if test="gender != null and gender.length()>0"> AND gender=#{gender}</if>
</trim>
```

- 
- **foreach**: 循环拼装, 用来处理in, 批量插入, 批量更新

```
<select id="selectPostIn" resultType="domain.blog.Post">
```

```
  SELECT * FROM POST P WHERE ID in
```

```
  <foreach item="item" index="index" collection="list"
```

```
    open="(" separator="," close=")">
```

```
    #{item}
```

```
  </foreach>
```

```
</select>
```

- **bind**: 可以从 OGNL 表达式中创建一个变量并将其绑定到上下文

```
<select id="selectBlogsLike" resultType="Blog">
```

```
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" />
```

```
  SELECT * FROM BLOG WHERE title LIKE #{pattern}
```

```
</select>
```

一个SQL注入:

```
select * from ${tableName} where name = #{name}
```

在这个例子中, 如果表名为

```
user; delete user; --
```

则动态解析之后 sql 如下:

```
select * from user; delete user; -- where name = ?;
```

7、插件Plguin

mybatis支持插件来插入自定义的处理过程，所有的plugin都需实现Interceptor接口，自定义的处理过程可以在Executor，ParameterHandler，ResultSetHandler，StatementHandler四个处理过程中插入

原理是在使用这四中类型处理数据的时候使用的都是经过plugin处理过的代理对象。同一个处理过程支持配置多个plugin，则plugin的执行顺序是根据包装的顺序，从最外部向内部执行，直到执行到目标对象的调用方法。包装的顺序是根据配置顺序，也就是说配置越靠前，包装的越深，越后执行。

若：拦截器的加载顺序

interceptor1

interceptor2

interceptor3

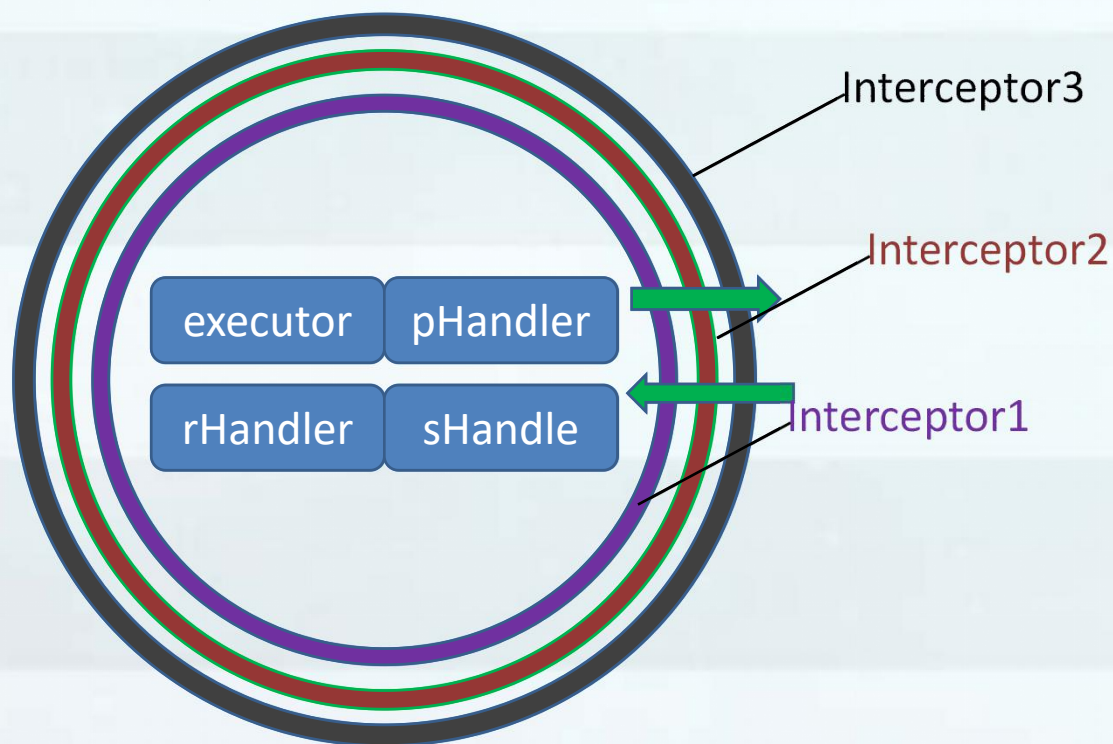
那么处理的顺序是

interceptor3

interceptor2

interceptor1

exccutor....





interceptor:

Object intercept(Invocation invocation) throws Throwable; //代理对象中调用的插件的自定义代码

Object plugin(Object target); //生成插件处理过的代理对象

void setProperties(Properties properties);

InterceptorChain:

在Configuration new Executor, ParameterHandler, ResultSetHandler, StatementHandler的时候执行拦截器的调用链去生成代理对象

```
public Object pluginAll(Object target) {  
    for (Interceptor interceptor : interceptors) {  
        target = interceptor.plugin(target); 产生target的代理对象  
    }  
    return target;  
}
```

Plugin:其实是一个实现了InvocationHandler的类

wrap(...): Proxy.newProxyInstance(target.getClass().getClassLoader(), interfaces, new Plugin(target, interceptor, signatureMap))

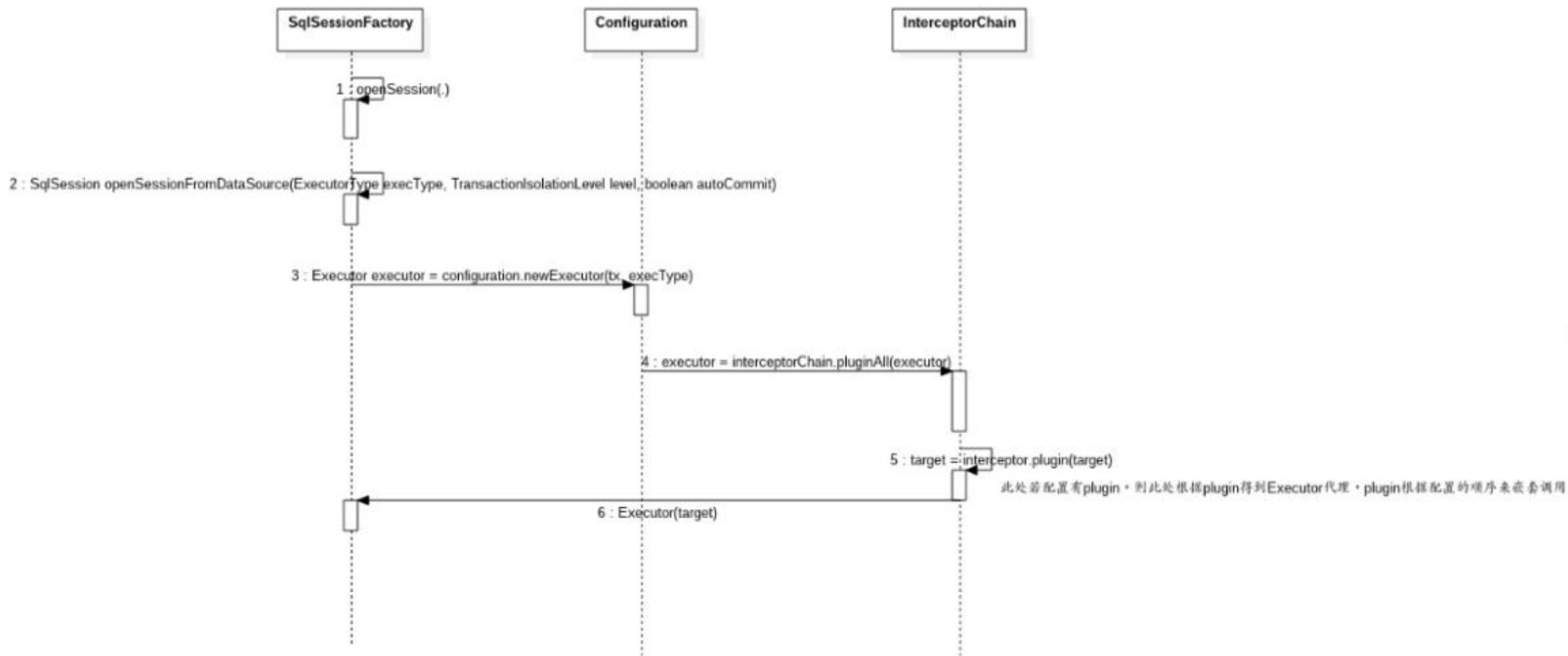
invoke(.....):

```
if (methods != null && methods.contains(method)) { //先执行plugin中的方法，最后再执行目标对象调用的方法  
    return interceptor.intercept(new Invocation(target, method, args));  
}
```

Invocation:

proceed() : return method.invoke(target, args);

时序图:





8、事务，缓存，连接池

org.apache.ibatis.transaction

-----org.apache.ibatis.transaction.jdbc

-----JdbcTransaction.java

-----JdbcTransactionFactory.java

-----org.apache.ibatis.transaction.managed

-----ManagedTransaction.java

-----ManagedTransactionFactory.java

-----Transaction.java

-----TransactionException.java

-----TransactionFactory.java

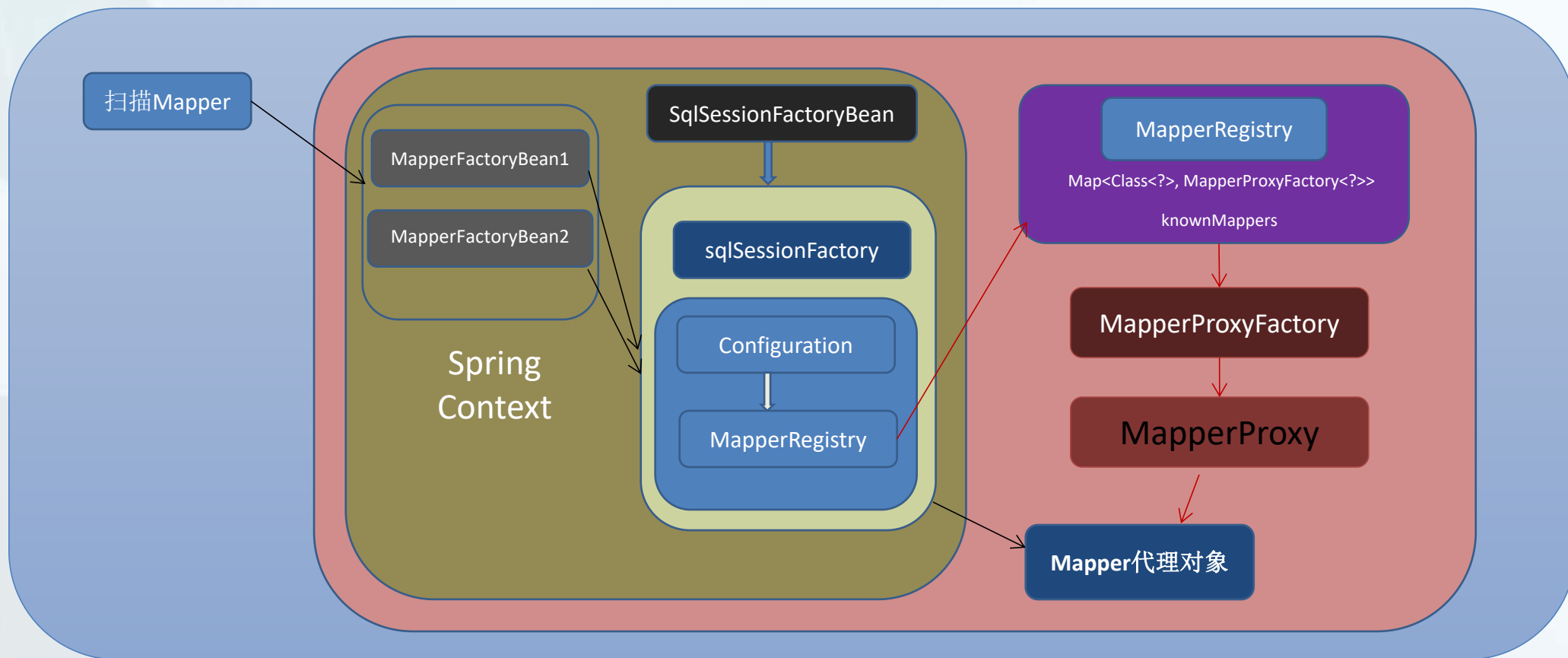


```
public static void insertStudent(){
    try{
        SqlSession sqlSession=sqlSessionFactory.openSession();
        StudentDAO studentDAO=sqlSession.getMapper(StudentDAO.class);
        Student student=new Student();
        student.setName("赵四");
        student.setAge(60);
        student.setGender(GenderEnum.MALE);
        student.setNumber("1960053011");
        studentDAO.insertStudent(student);
        sqlSession.commit();
    }catch(Exception e){
        sqlSession.rollback();
    }finally{
        if(sqlSession!=null){
            sqlSession.close();
        }
    }
}
```


9、Mybatis-Spring

MyBatis-Spring是什么

我们使用Spring和MyBatis作为我们的开发框架时，在搭建开发环境的时候，都会做一个Spring与MyBatis的整合，使用到的就是MyBatis-Spring这个中间件，MyBatis-Spring中间件帮我们把mapper接口和mapper.xml文件对应的代理类注册到Spring中，因此，我们在service层中就能根据类型注入，将对应mapper接口的代理类注入到service层中，我们才能够调用到对应的方法,整体原理：



- 
- 配置SqlSessionFactoryBean或者new，产生sqlSessionFactory对象：

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="mapperLocations" value="classpath:/**/*.xml"/>
  <property name="configLocation" value="classpath:mybatis-configuration.xml"/>
</bean>
```

参考MybatisAutoConfiguration

@Bean

@ConditionalOnMissingBean

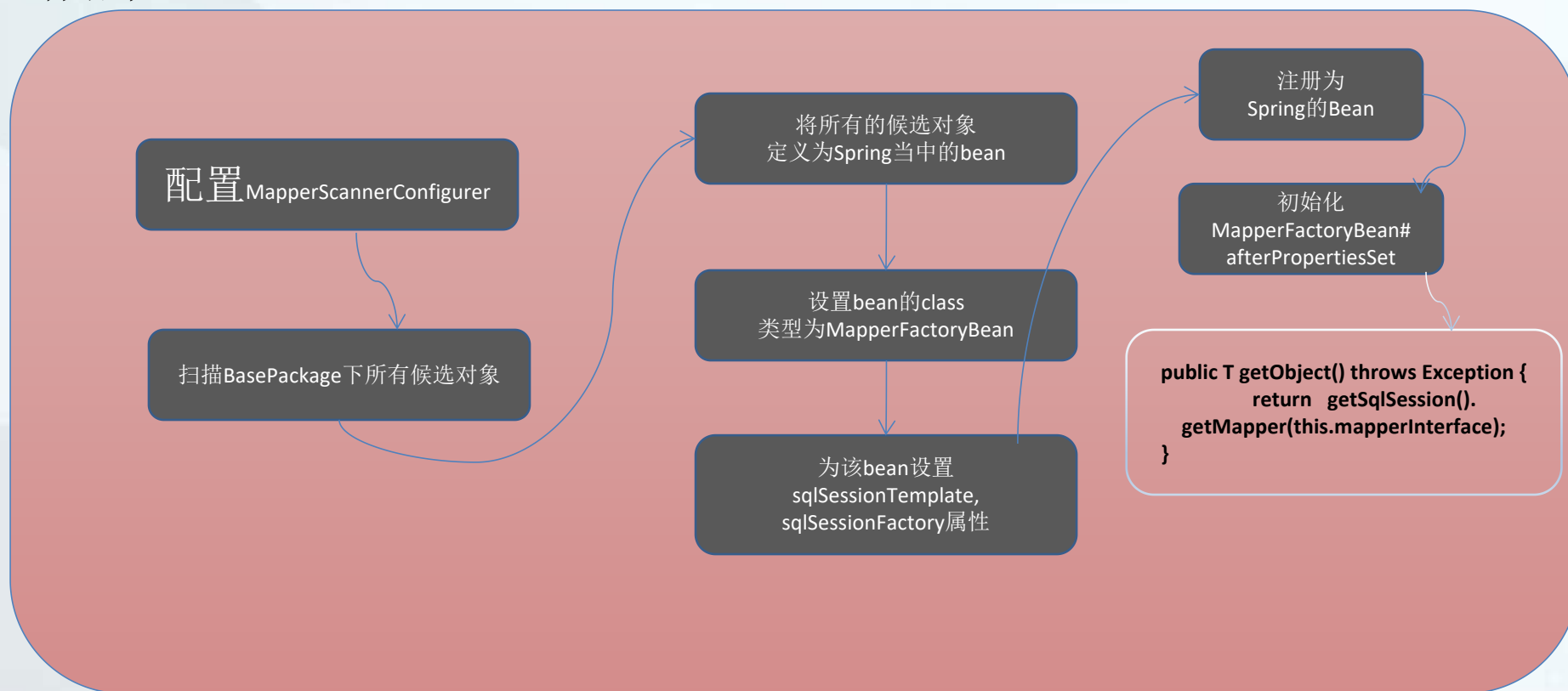
```
public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {
    SqlSessionFactoryBean factory = new SqlSessionFactoryBean();
    set....;
    factory.getObject();
}
```

->buildSqlSessionFactory();类比Mybatis SqlSessionFactory的构造过程，
SqlSessionFactoryBuilder.build(configuration);

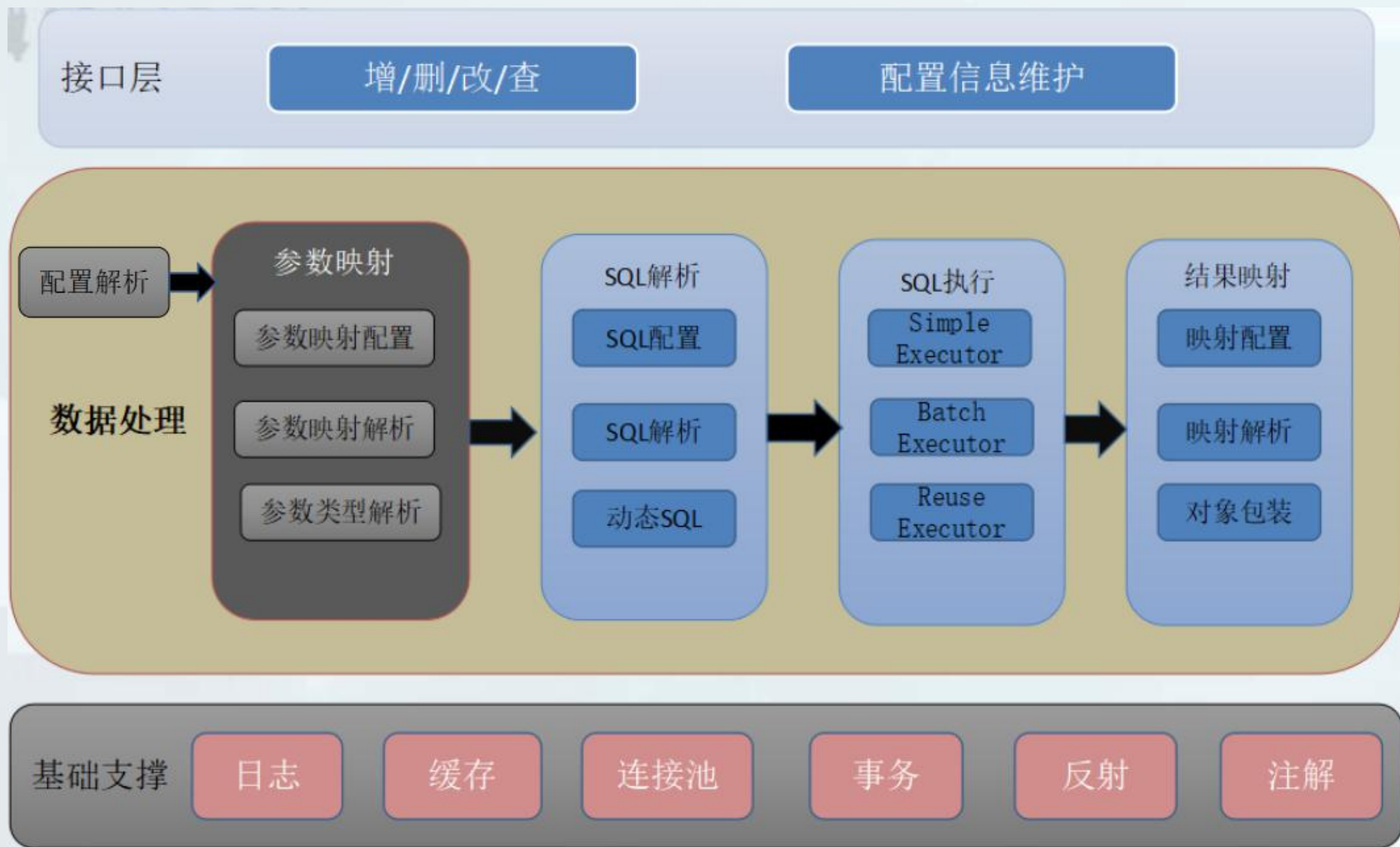
- 扫描Mapper接口和Mapper.xml文件，注册MapperFactoryBean的

```
<!-- mapper definition -->
<bean id="mapperScannerConfigurer" class="com.hand.hap.mybatis.spring.MapperScannerConfigurer">
    <property name="basePackage" value="*. **. mapper" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
```

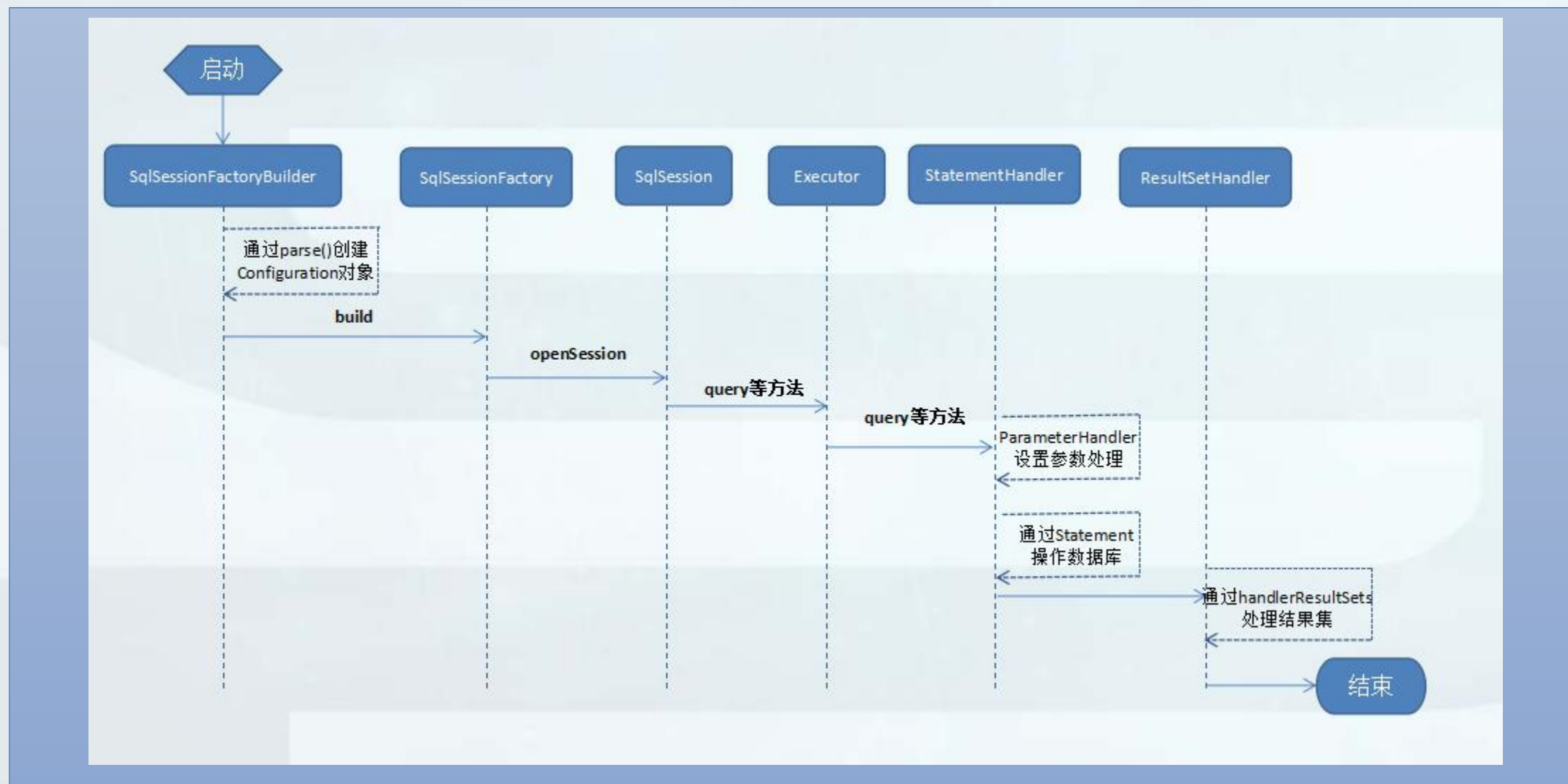
整体流程:



三、原理浅析



Mybatis整体流程图:






四、实践

Demo源码地址: <https://github.com/kedewaiwai/mybatis-demo.git>

1.环境准备

```
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>1.1.1</version>
</dependency>
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>4.1.5</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.0.23</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```



2.配置

```
spring.application.name=Winter  
server.port=9006  
logging.path=.  
logging.level=info
```

```
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  
spring.datasource.url=jdbc:mysql://192.168.3.162:3306/mybatis_demo1?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true  
spring.datasource.username=root  
spring.datasource.password=123456  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
spring.datasource.initialSize=5  
spring.datasource.minIdle=5  
spring.datasource.maxActive=200  
spring.datasource.maxWait=60000
```

```
second.spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  
second.spring.datasource.url=jdbc:mysql://192.168.3.166:3306/mybatis_demo2?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true  
second.spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
second.spring.datasource.username=root  
second.spring.datasource.password=123456  
service.winter.workerId=31
```

@Configuration

@MapperScan(basePackages = "com.dachen.demo.mybatis.dao.primary", sqlSessionRef = "sqlSessionFactory")

public class PrimaryDruidConfiguration {.....}

@Configuration

@MapperScan(basePackages = "com.dachen.demo.mybatis.dao.second", sqlSessionRef = "secondSqlSessionFactory")

public class SecondDruidConfiguration {.....}



3. Mapper接口编写和实现

1) 插入

2) 更新

3) 查询(sql注入)

```
<select id="selectUserByNameAndTime" resultType="com.dachen.demo.mybatis.module.User">
    <bind name="pattern" value="'\%' + name + '%\''+';delete from demo_user;' />
    select id,name,age,telephone,title,address,createTime from demo_user
    <trim prefix="WHERE" prefixOverrides="AND|OR">
        <if test="name != null and name.length()>0">AND name like ${pattern}</if>
        <if test="time != null and time>0"><![CDATA[ AND createTime < #{time}]]></if>
    </trim>
</select>
```

4) 批量插入

5) 批量更新

6) 一对多关联查询

7) 一对一关联查询

8) 分页插件查询



谢谢
大家的参与!