原理探究

C-2 创建: 张林伟, 最后修改: 张林伟 2018-09-26 20:59

> Mybatis 提供了 Mapper 接口的代理对象，在执行 Mapper 接口方法时，实际执行的是 Mybatis 的代理对象，代理对象在 invoke 方法内获取 Mapper 接口类全名+方法全名 作为 statement 的 ID，然后通过 ID 去 Statement 匹配注册的 SQL，然后使用 SqlSession 执行这个 SQL。所以，这也解释了为什么 Mybatis 映射文件需要 namespace 和 id ， 前者是类全名，后者是方法名。

利用了 JDK 动态代理：JDK动态代理

启动流程：

Spring 配置文件配置 SqlSessionFactoryBean

```
<!-- 集成myBaits框架,配置sqlSessionFatory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <!-- 配置扫描Domain的包路径 -->
    <property name="typeAliasesPackage" value="com.meituan.service.mobile.audit.dao.bean"/>
    <!-- 配置扫描Mapper XML的位置 -->
    <property name="mapperLocations">
        <list>
            <value>classpath*:dao/mapper/**/*.xml</value>
        </list>
    </property>
    <!-- 配置mybatis配置文件的位置 -->
    <property name="configLocation" value="classpath:dal/mybatis-config.xml"/>
</bean>
```

SqlSessionFactoryBean 调用 buildSqlSessionFactory() 创建 SqlSessionFactory

```
public void afterPropertiesSet() throws Exception {
    Assert.notNull(this.dataSource,  message: "Property 'dataSource' is required");
    Assert.notNull(this.sqlSessionFactoryBuilder,  message: "Property 'sqlSessionFactoryBuilder' is required");
    this.sqlSessionFactory = this.buildSqlSessionFactory();
}

protected SqlSessionFactory buildSqlSessionFactory() throws IOException {
    XMLConfigBuilder xmlConfigBuilder = null;
    Configuration configuration;
    if (this.configLocation != null) {
        xmlConfigBuilder = new XMLConfigBuilder(this.configLocation.getInputStream(), (String)null, this.configura
        configuration = xmlConfigBuilder.getConfiguration();
    } else {
        if (logger.isDebugEnabled()) {
            logger.debug( s: "Property 'configLocation' not specified, using default MyBatis Configuration");
        }

        configuration = new Configuration();
        configuration.setVariables(this.configurationProperties);
    }

    if (this.objectFactory != null) {
```

mapper加载

在调用 buildSqlSessionFactory() 过程中，会调用 XMLConfigBuilder 的 parse() 方法

```
if (xmlConfigBuilder != null) {
    try {
        xmlConfigBuilder.parse();
        if (logger.isDebugEnabled()) {
            logger.debug( s: "Parsed configuration file: '" + this.configLocation + "'");
        }
    } catch (Exception var23) {
        throw new NestedIOException("Failed to parse config resource: " + this.configLocation, var23);
    } finally {
        ErrorContext.instance().reset();
    }
}
```

在执行 parse() 时，会执行 mapperElement() 方法

```java
public Configuration parse() {
    if (this.parsed) {
        throw new BuilderException("Each XMLConfigBuilder can only be used once.");
    } else {
        this.parsed = true;
        this.parseConfiguration(this.parser.evalNode( expression: "/configuration"));
        return this.configuration;
    }
}

private void parseConfiguration(XNode root) {
    try {
        this.propertiesElement(root.evalNode("properties"));
        this.typeAliasesElement(root.evalNode("typeAliases"));
        this.pluginElement(root.evalNode("plugins"));
        this.objectFactoryElement(root.evalNode("objectFactory"));
        this.objectWrapperFactoryElement(root.evalNode("objectWrapperFactory"));
        this.settingsElement(root.evalNode("settings"));
        this.environmentsElement(root.evalNode("environments"));
        this.databaseIdProviderElement(root.evalNode("databaseIdProvider"));
        this.typeHandlerElement(root.evalNode("typeHandlers"));
        this.mapperElement(root.evalNode("mappers"));
    } catch (Exception var3) {
        throw new BuilderException("Error parsing SQL Mapper Configuration. Cause: " + var3, var3);
    }
}
```

在执行 mapperElement() 方法时，会执行Configuration 的 addMapper() 方法

```java
private void mapperElement(XNode parent) throws Exception {
    if (parent != null) {
        Iterator i$ = parent.getChildren().iterator();

        while(true) {
            while(i$.hasNext()) {
                XNode child = (XNode)i$.next();
                String resource;
                if ("package".equals(child.getName())) {
                    resource = child.getStringAttribute( name: "name");
                    this.configuration.addMappers(resource);
                } else {
                    resource = child.getStringAttribute( name: "resource");
                    String url = child.getStringAttribute( name: "url");
                    String mapperClass = child.getStringAttribute( name: "class");
                    XMLMapperBuilder mapperParser;
                    InputStream inputStream;
                    if (resource != null && url == null && mapperClass == null) {
                        ErrorContext.instance().resource(resource);
                        inputStream = Resources.getResourceAsStream(resource);
                        mapperParser = new XMLMapperBuilder(inputStream, this.configuration, resource, this.configura
                        mapperParser.parse();
                    } else if (resource == null && url != null && mapperClass == null) {
                        ErrorContext.instance().resource(url);
                        inputStream = Resources.getUrlAsStream(url);
                        mapperParser = new XMLMapperBuilder(inputStream, this.configuration, url, this.configuration.
                        mapperParser.parse();
                    } else {
                        if (resource != null || url != null || mapperClass == null) {
                            throw new BuilderException("A mapper element may only specify a url, resource or class, b
                        }

                        Class<?> mapperInterface = Resources.classForName(mapperClass);
                        this.configuration.addMapper(mapperInterface);
                    }
                }
            }

            return;
        }
    }
}
```

MapperProxy 对象由 MapperProxyFactory 创建

```java
public class MapperProxyFactory<T> {

  private final Class<T> mapperInterface;
  private final Map<Method, MapperMethod> methodCache = new ConcurrentHashMap<Method, MapperMethod>();

  public MapperProxyFactory(Class<T> mapperInterface) {
    this.mapperInterface = mapperInterface;
  }

  public Class<T> getMapperInterface() {
    return mapperInterface;
  }

  public Map<Method, MapperMethod> getMethodCache() {
    return methodCache;
  }

  @SuppressWarnings("unchecked")
  protected T newInstance(MapperProxy<T> mapperProxy) {
    return (T) Proxy.newProxyInstance(mapperInterface.getClassLoader(), new Class[] { mapperInterface }, mapperProxy);
  }

  public T newInstance(SqlSession sqlSession) {
    final MapperProxy<T> mapperProxy = new MapperProxy<T>(sqlSession, mapperInterface, methodCache);
    return newInstance(mapperProxy);
  }

}
```

执行流程：

biz 层调用 mapper 层方法

```
*/
@Service
public class UserBiz {

    @Autowired
    private IUserMapper userMapper;

    public List<UserDO> queryAllUsers() {
        return userMapper.queryAllUsers();
    }
}
```

进入 MapperProxy 类中 invoke 方法

```
UserBiz.java  MapperProxy.java
 * @author Eduardo Macarron
 */
public class MapperProxy<T> implements InvocationHandler, Serializable {

  private static final long serialVersionUID = -6424540398559729838L;
  private final SqlSession sqlSession;  sqlSession: SqlSessionTemplate@6152
  private final Class<T> mapperInterface;  mapperInterface: "interface com.example.demodal.mapper.user.IUserMapper"
  private final Map<Method, MapperMethod> methodCache;  methodCache:  size = 1

  public MapperProxy(SqlSession sqlSession, Class<T> mapperInterface, Map<Method, MapperMethod> methodCache) {
    this.sqlSession = sqlSession;
    this.mapperInterface = mapperInterface;
    this.methodCache = methodCache;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  proxy: "org.apache.ibatis.binding.MapperProxy@56b3e44c"  method: "public
    try {
      if (Object.class.equals(method.getDeclaringClass())) {  method: "public abstract java.util.List com.example.demodal.mapper.user.IUserMapper.queryAllUsers
        return method.invoke( obj: this, args);
      } else if (isDefaultMethod(method)) {
        return invokeDefaultMethod(proxy, method, args);
      }
    } catch (Throwable t) {
      throw ExceptionUtil.unwrapThrowable(t);
    }
    final MapperMethod mapperMethod = cachedMapperMethod(method);
    return mapperMethod.execute(sqlSession, args);
  }
```

最后使用 cachedMapperMethod 方法从 methodCache 中获取 mapperMethod

```
private final Map<Method, MapperMethod> methodCache;  methodCache:  size = 1
```

然后执行 mapperMethod，返回结果

```
public Object execute(SqlSession sqlSession, Object[] args) {  sqlSession: SqlSessionTemplate@6152  args: null
  Object result;
  switch (command.getType()) {  command: MapperMethod$SqlCommand@7671
    case INSERT: {
    Object param = method.convertArgsToSqlCommandParam(args);
      result = rowCountResult(sqlSession.insert(command.getName(), param));
      break;
    }
    case UPDATE: {
      Object param = method.convertArgsToSqlCommandParam(args);
      result = rowCountResult(sqlSession.update(command.getName(), param));
      break;
    }
    case DELETE: {
      Object param = method.convertArgsToSqlCommandParam(args);
      result = rowCountResult(sqlSession.delete(command.getName(), param));
      break;
    }
    case SELECT:
      if (method.returnsVoid() && method.hasResultHandler()) {
        executeWithResultHandler(sqlSession, args);
        result = null;
      } else if (method.returnsMany()) {
        result = executeForMany(sqlSession, args);
      } else if (method.returnsMap()) {
        result = executeForMap(sqlSession, args);
      } else if (method.returnsCursor()) {
        result = executeForCursor(sqlSession, args);
      } else {
        Object param = method.convertArgsToSqlCommandParam(args);
        result = sqlSession.selectOne(command.getName(), param);
      }
      break;
    case FLUSH:
      result = sqlSession.flushStatements();
```

参考资料：

https://my.oschina.net/heweipo/blog/824912