1. MapperProxy的invoke(Object proxy, Method method, Object[] args)方法，也是jdk动态代理实现的，从mapper接口到sqlSession的跳跃

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
 try {
 if (Object.class.equals(method.getDeclaringClass())) {
 return method.invoke(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);
}
```

2. MapperMethod的execute(SqlSession sqlSession, Object[] args)方法

```
public Object execute(SqlSession sqlSession, Object[] args) {
 Object result;
 switch (command.getType()) {
 case INSERT: {
 Object param = method.convertArgsToSqlCommandParam(args);
 result = rowCountResult(sqlSession.insert(command.getName(), param));
 break;
 }
 case UPDATE: {
  Object param = method.convertArgsToSqlCommandParam(args);
  // 因为我们的拦截是拦截的update方法，所以会执行这个
  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();
  break;
  default:
  throw new BindingException("Unknown execution method for: " + command.getName());
  }
  if (result == null && method.getReturnType().isPrimitive() && !method.returnsVoid()) {
  throw new BindingException("Mapper method '" + command.getName()
  + " attempted to return null from a method with a primitive return type (" + method.getReturnType() + ").");
  }
  return result;
 }
```

3. DefaultSqlSession的update(String statement, Object parameter)方法

```
public int update(String statement, Object parameter) {
  try {
  dirty = true;
  MappedStatement ms = configuration.getMappedStatement(statement);
  // 这边的executor对象被代理了，所以会进到InvocationHandler接口实现类的invoke方法中
  // 这里是进入到Plugin的invoke() 方法，ms, wrapCollection(parameter)就是args的封装
  return executor.update(ms, wrapCollection(parameter));
  } catch (Exception e) {
```

```
  throw ExceptionFactory.wrapException("Error updating database. Cause: "
+ e, e);
  } finally {
  ErrorContext.instance().reset();
  }
 }
```

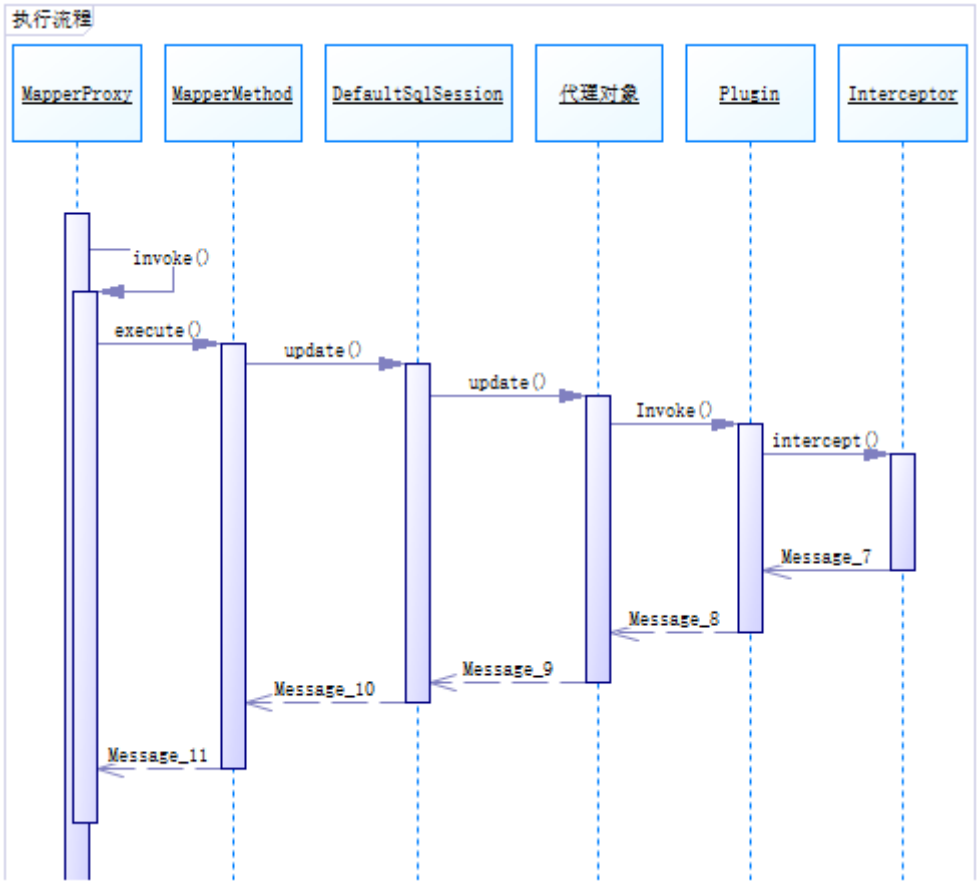
4. Plugin.invoke(Object proxy, Method method, Object[] args)方法，这是实现代理逻辑的地方

```
 public Object invoke(Object proxy, Method method, Object[] args) throws T
hrowable {
  try {
  // 从拦截器签名中获取拦截方法
  Set<Method> methods = signatureMap.get(method.getDeclaringClass());
  // 如果当前方法在拦截方法集合中
  if (methods != null && methods.contains(method)) {
  // 执行拦截操作
  // target被拦截对象，一个拦截时是Executor对象
  // methon拦截方法
   // args 方法参数 args[0]是ms对象 args[1]是传入参数
   return interceptor.intercept(new Invocation(target, method, args));
  }
  return method.invoke(target, args);
  } catch (Exception e) {
  throw ExceptionUtil.unwrapThrowable(e);
  }
 }
```

5. interceptor.intercept(new Invocation(target, method, args))进入自定义拦截器

```
public Object intercept(Invocation invocation) throws Throwable {
 System.out.println("拦截器拦截");
 Object result = invocation.proceed();
 return result;
}
```

6. 执行流程


执行流程
MapperProxy
MapperMethod
DefaultSqlSession
代理对象
Plugin
Interceptor
invoke()
execute()
update()
update()
Invoke()
intercept()
Message_7
Message_8
Message_9
Message_10
Message_11