1. 解析mybatis.xml配置文件，构建SqlSessionFactory

```
String resource = "mybatis.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
```

1.1 mybatis.xml的配置信息都配解析存储在Configure对象中

```
// SqlSessionBuilder
public SqlSessionFactory build(InputStream inputStream, String environment, Properties properties) {
 try {
 XMLConfigBuilder parser = new XMLConfigBuilder(inputStream, environment, properties);
 return build(parser.parse());
 } catch (Exception e) {
 throw ExceptionFactory.wrapException("Error building SqlSession.", e);
 } finally {
 ErrorContext.instance().reset();
  try {
  inputStream.close();
  } catch (IOException e) {
  // Intentionally ignore. Prefer previous error.
  }
  }
 }


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

private void parseConfiguration(XNode root) {
  try {
  //issue #117 read properties first
  propertiesElement(root.evalNode("properties"));
```

```
33   Properties settings = settingsAsProperties(root.evalNode("settings"));
34   loadCustomVfs(settings);
35   typeAliasesElement(root.evalNode("typeAliases"));
36   pluginElement(root.evalNode("plugins"));
37   objectFactoryElement(root.evalNode("objectFactory"));
38   objectWrapperFactoryElement(root.evalNode("objectWrapperFactory"));
39   reflectorFactoryElement(root.evalNode("reflectorFactory"));
40   settingsElement(settings);
41   // read it after objectFactory and objectWrapperFactory issue #631
42   environmentsElement(root.evalNode("environments"));
43   databaseIdProviderElement(root.evalNode("databaseIdProvider"));
44   typeHandlerElement(root.evalNode("typeHandlers"));
45   mapperElement(root.evalNode("mappers"));
46   } catch (Exception e) {
47   throw new BuilderException("Error parsing SQL Mapper Configuration. Cau
se: " + e, e);
48   }
49 }
50
51
52 // 解析完创建SqlSessionFactory
53 public SqlSessionFactory build(Configuration config) {
54   return new DefaultSqlSessionFactory(config);
55 }
```

1.2 获取SqlSession

```
1  private SqlSession openSessionFromDataSource(ExecutorType execType, Trans
actionIsolationLevel level, boolean autoCommit) {
2    Transaction tx = null;
3    try {
4    final Environment environment = configuration.getEnvironment();
5    final TransactionFactory transactionFactory = getTransactionFactoryFromE
nvironment(environment);
6    tx = transactionFactory.newTransaction(environment.getDataSource(), leve
l, autoCommit);
7    final Executor executor = configuration.newExecutor(tx, execType);
8    return new DefaultSqlSession(configuration, executor, autoCommit);
9    } catch (Exception e) {
10   closeTransaction(tx); // may have fetched a connection so lets call clo
se()
```

```
11   throw ExceptionFactory.wrapException("Error opening session. Cause: " +
e, e);
12   } finally {
13   ErrorContext.instance().reset();
14   }
15  }
```

创建SqlSession的过程，其实是对jdbc的Connection的一次再封装，Connection-->Transaction-->Executor-->SqlSeesion

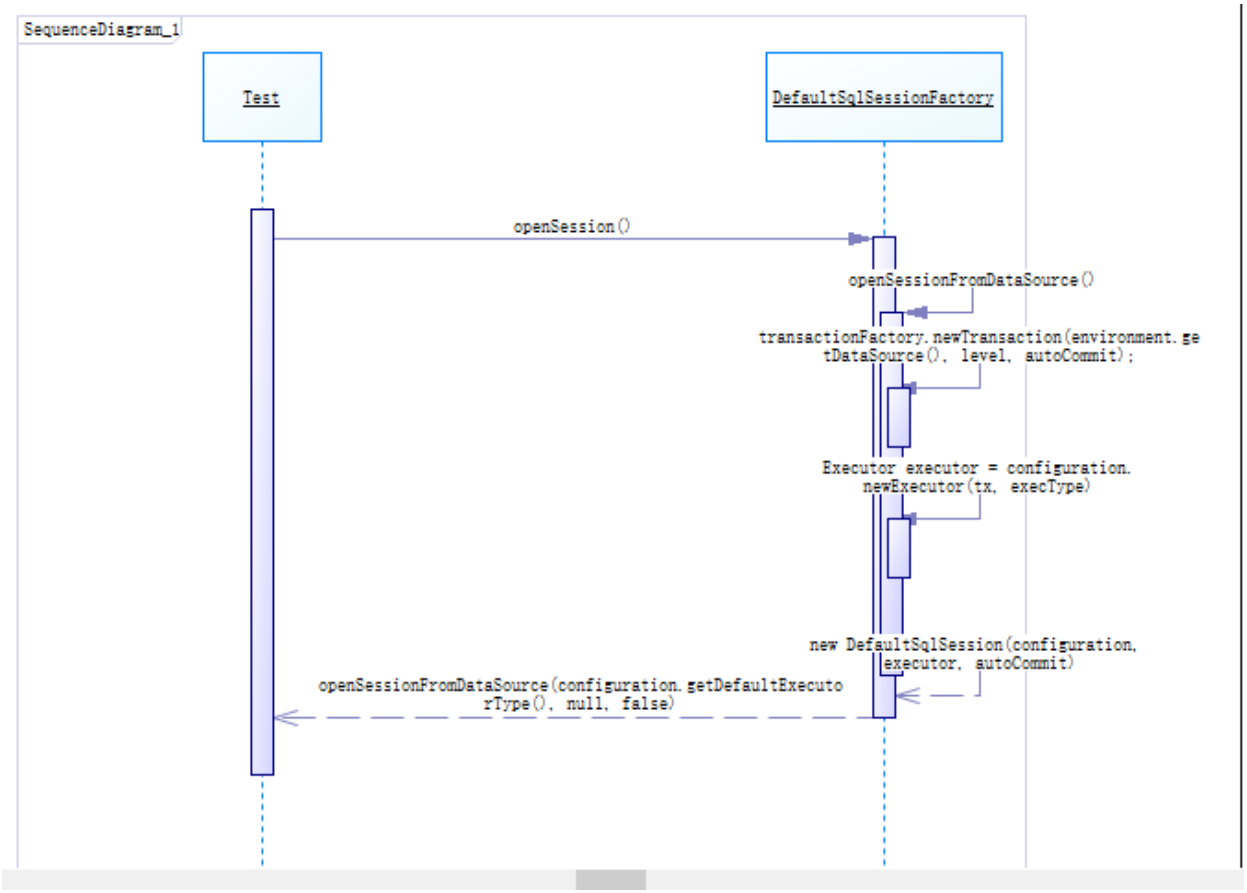
Executor有三种类型：SIMPLE 就是普通的执行器；REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行批量更新。

configuration.newExecutor(tx, execType)会根据execType创建出其中一种执行器，如下：

```
1  public Executor newExecutor(Transaction transaction, ExecutorType executo
rType) {
2   executorType = executorType == null ? defaultExecutorType :
executorType;
3   executorType = executorType == null ? ExecutorType.SIMPLE :
executorType;
4   Executor executor;
5   if (ExecutorType.BATCH == executorType) {
6   executor = new BatchExecutor(this, transaction);
7   } else if (ExecutorType.REUSE == executorType) {
8   executor = new ReuseExecutor(this, transaction);
9   } else {
10   executor = new SimpleExecutor(this, transaction);
11   }
12   if (cacheEnabled) {
13   executor = new CachingExecutor(executor);
14   }
15   executor = (Executor) interceptorChain.pluginAll(executor);
16   return executor;
17  }
```

最后，将执行器和配置对象封装到SqlSession中返回

1.3 调用过程如下


SequenceDiagram_1
Test
DefaultSqlSessionFactory
openSession()
openSessionFromDataSource()
transactionFactory.newTransaction(environment.ge
tDataSource(), level, autoCommit);
Executor executor = configuration.
newExecutor(tx, execType)
new DefaultSqlSession(configuration,
executor, autoCommit)
openSessionFromDataSource(configuration.getDefaultExecuto
rType(), null, false)