mybatis的拦截是通过jdk的动态代理实现的，代理了Executor对象，下面先捋一下Executor代理的流程。

1. Executor对象初始化

```java
private SqlSession openSessionFromDataSource(ExecutorType execType, TransactionIsolationLevel level, boolean autoCommit) {
  Transaction tx = null;
  try {
  final Environment environment = configuration.getEnvironment();
  final TransactionFactory transactionFactory = getTransactionFactoryFromEnvironment(environment);
  tx = transactionFactory.newTransaction(environment.getDataSource(), level, autoCommit);
  // Executor对象是通过Configuration对象生成的
  final Executor executor = configuration.newExecutor(tx, execType);
  return new DefaultSqlSession(configuration, executor, autoCommit);
   } catch (Exception e) {
   closeTransaction(tx); // may have fetched a connection so lets call close()
   throw ExceptionFactory.wrapException("Error opening session. Cause: " + e, e);
   } finally {
   ErrorContext.instance().reset();
   }
 }
```

2. Configuration.newExecutor(tx, execType)

```java
public Executor newExecutor(Transaction transaction, ExecutorType executorType) {
  executorType = executorType == null ? defaultExecutorType : executorType;
  executorType = executorType == null ? ExecutorType.SIMPLE : executorType;
  Executor executor;
  if (ExecutorType.BATCH == executorType) {
  executor = new BatchExecutor(this, transaction);
  } else if (ExecutorType.REUSE == executorType) {
  executor = new ReuseExecutor(this, transaction);
  } else {
   executor = new SimpleExecutor(this, transaction);
   }
   if (cacheEnabled) {
```

```
executor = new CachingExecutor(executor);
}
// executor通过了层层代理，有多少个拦截器，就代理了多少次
executor = (Executor) interceptorChain.pluginAll(executor);
return executor;
}
```

3. interceptorChain.pluginAll(executor)

```
public Object pluginAll(Object target) {
 // interceptors是通过xml解析时解析<plugins>标签得到的
 for (Interceptor interceptor : interceptors) {
 // 下面就是要生成代理对象， 不需要自己实现。调用Plugin.wrap()就可以了
 target = interceptor.plugin(target);
 }
 return target;
}
```

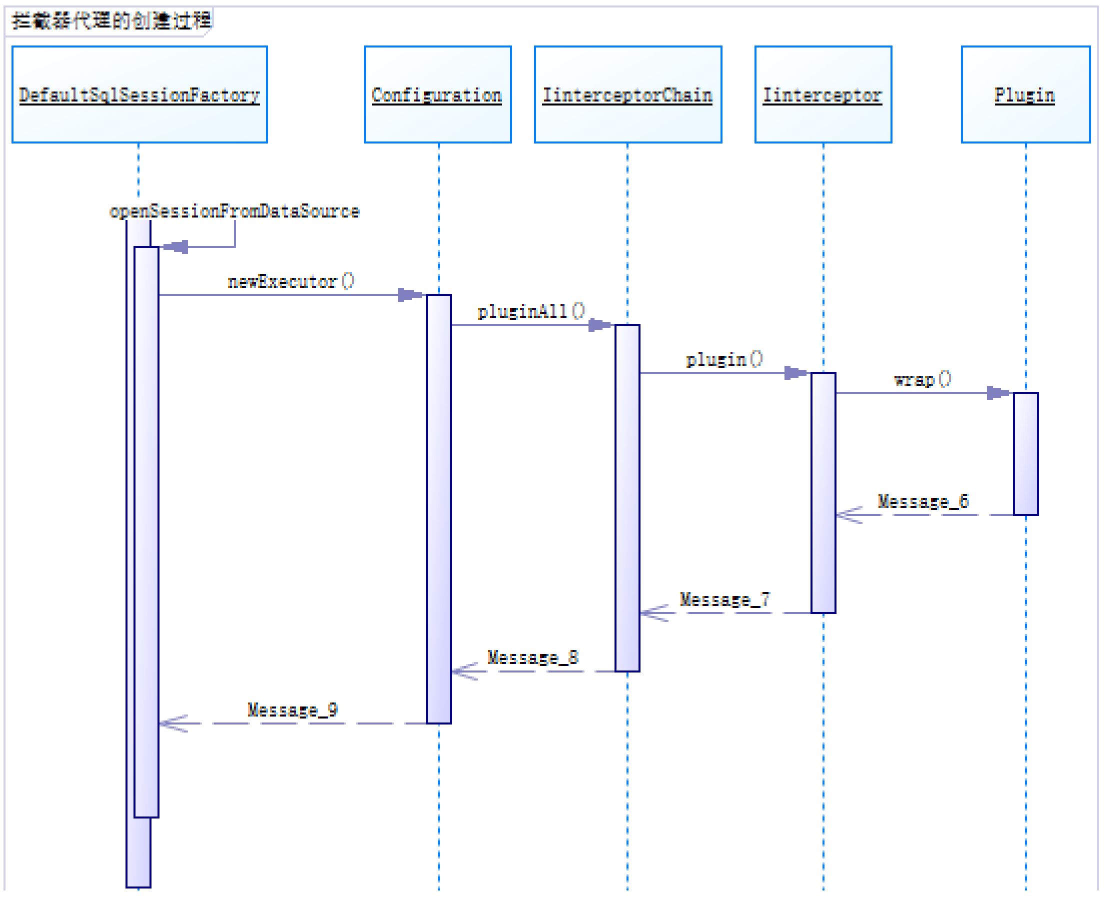
4. 先来看看Plugin.wrap(Object target, Interceptor interceptor)方法

```
/**
* @target 被代理对象，第一次是Executor对象
* @interceptor 拦截器
*/
public static Object wrap(Object target, Interceptor interceptor) {
 // 获取拦截上的签名
 /**
 @Intercepts({
 @Signature(type = Executor.class,
  method = "update",
  args = {MappedStatement.class, Object.class})
  })
  */
  Map<Class<?>, Set<Method>> signatureMap = getSignatureMap(interceptor);
  Class<?> type = target.getClass();
  // 获取被代理对象的接口Executor
  Class<?>[] interfaces = getAllInterfaces(type, signatureMap);
  // 如果有接口，创建代理对象，jdk的动态代理是基于接口实现的
  if (interfaces.length > 0) {
  return Proxy.newProxyInstance(
  type.getClassLoader(),
  interfaces,
  // InvocationHandler的实现类，后续会进入Plugin实现类的invoke方法
  new Plugin(target, interceptor, signatureMap));
  }
  return target;
 }
```

5. 最后再来看自己实现的拦截器

```
@Intercepts({
 @Signature(type = Executor.class,
 method = "update",
 args = {MappedStatement.class, Object.class})
})
public class BaseFieldInterceptor implements Interceptor {

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

  public Object plugin(Object target) {
  // 创建代理对象，直接调用方法即可，内部已经帮我门实现了
  return Plugin.wrap(target, this);
  }

  public void setProperties(Properties properties) {

  }
}
```

6. 创建流程图


拦截器代理的创建过程
DefaultSqlSessionFactory
Configuration
IinterceptorChain
Iinterceptor
Plugin
openSessionFromDataSource
newExecutor()
pluginAll()
plugin()
wrap()
Message_6
Message_7
Message_8
Message_9