# mybatis-config.xml

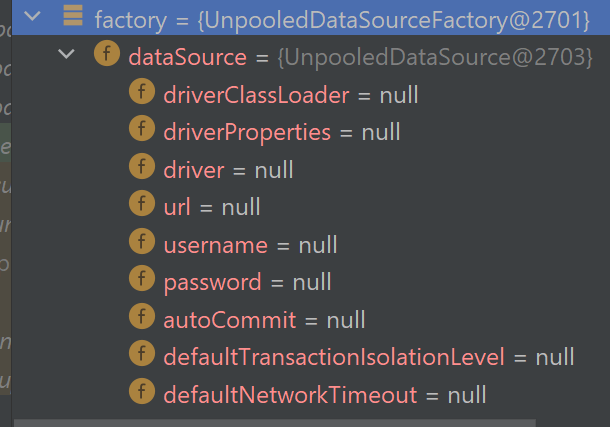
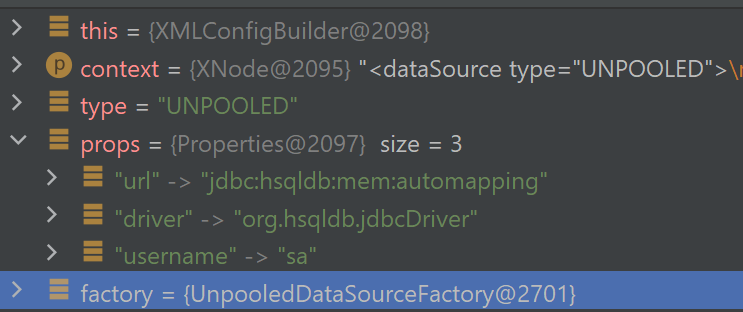
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE configuration  
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-config.dtd">  
  
<configuration>  
  
 <!-- autoMappingBehavior should be set in each test case -->  
  
 <environments default="development">  
 <environment id="development">  
 <transactionManager type="JDBC">  
 <property name="" value=""/>  
 </transactionManager>  
 <dataSource type="UNPOOLED">  
 <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>

# org.apache.ibatis.builder.xml.XMLConfigBuilder#environmentsElement

private void environmentsElement(XNode context) throws Exception {  
 if (context != null) {  
 if (environment == null) {  
 environment = context.getStringAttribute("default");  
 }  
 for (XNode child : context.getChildren()) {  
 String id = child.getStringAttribute("id");  
 if (isSpecifiedEnvironment(id)) {  
 TransactionFactory txFactory = transactionManagerElement(child.evalNode("transactionManager"));  
 DataSourceFactory dsFactory = dataSourceElement(child.evalNode("dataSource")); <-#1  
 DataSource dataSource = dsFactory.getDataSource();  
 Environment.Builder environmentBuilder = new Environment.Builder(id)  
 .transactionFactory(txFactory)  
 .dataSource(dataSource);  
 configuration.setEnvironment(environmentBuilder.build());  
 break;  
 }  
 }  
 }  
}

# #1: org.apache.ibatis.builder.xml.XMLConfigBuilder#dataSourceElement

private DataSourceFactory dataSourceElement(XNode context) throws Exception {  
 if (context != null) {  
 String type = context.getStringAttribute("type");  
 Properties props = context.getChildrenAsProperties();  
 DataSourceFactory factory = (DataSourceFactory) resolveClass(type).getDeclaredConstructor().newInstance();  
 factory.setProperties(props); <-##1  
 return factory;  
 }  
 throw new BuilderException("Environment declaration requires a DataSourceFactory.");  
}



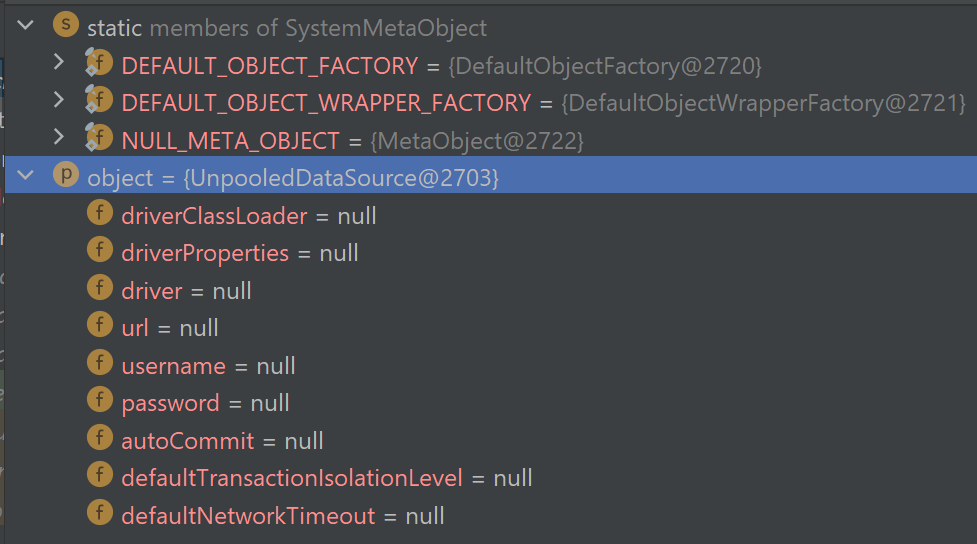
# ##1: org.apache.ibatis.datasource.unpooled.UnpooledDataSourceFactory#setProperties

public void setProperties(Properties properties) {  
 Properties driverProperties = new Properties();  
 MetaObject metaDataSource = SystemMetaObject.*forObject*(dataSource); <-###1  
 for (Object key : properties.keySet()) {  
 String propertyName = (String) key;  
 if (propertyName.startsWith(*DRIVER\_PROPERTY\_PREFIX*)) {  
 String value = properties.getProperty(propertyName);  
 driverProperties.setProperty(propertyName.substring(*DRIVER\_PROPERTY\_PREFIX\_LENGTH*), value);  
 } else if (metaDataSource.hasSetter(propertyName)) {  
 String value = (String) properties.get(propertyName);  
 Object convertedValue = convertValue(metaDataSource, propertyName, value);  
 metaDataSource.setValue(propertyName, convertedValue);  
 } else {  
 throw new DataSourceException("Unknown DataSource property: " + propertyName);  
 }  
 }  
 if (driverProperties.size() > 0) {  
 metaDataSource.setValue("driverProperties", driverProperties);  
 }  
}

# ###1: org.apache.ibatis.reflection.SystemMetaObject#forObject

public static final ObjectFactory *DEFAULT\_OBJECT\_FACTORY* = new DefaultObjectFactory(); <-####1  
public static final ObjectWrapperFactory *DEFAULT\_OBJECT\_WRAPPER\_FACTORY* = new DefaultObjectWrapperFactory(); <-####2

public static MetaObject forObject(Object object) {  
 return MetaObject.*forObject*(object, *DEFAULT\_OBJECT\_FACTORY*, *DEFAULT\_OBJECT\_WRAPPER\_FACTORY*, new DefaultReflectorFactory() <-####3); <-####4  
}



# ####1: org.apache.ibatis.reflection.factory.DefaultObjectFactory

public class DefaultObjectFactory implements ObjectFactory, Serializable <-####1i

# ####1i: org.apache.ibatis.reflection.factory.ObjectFactory

*/\*\*  
 \* MyBatis uses an ObjectFactory to create all needed new Objects.  
 \*  
 \** ***@author*** *Clinton Begin  
 \*/*public interface ObjectFactory {  
  
 */\*\*  
 \* Sets configuration properties.  
 \** ***@param*** *properties configuration properties  
 \*/* default void setProperties(Properties properties) {  
 // NOP  
 }  
  
 */\*\*  
 \* Creates a new object with default constructor.  
 \*  
 \** ***@param*** <*T*>  
 *\* the generic type  
 \** ***@param*** *type  
 \* Object type  
 \** ***@return*** *the t  
 \*/* <T> T create(Class<T> type);  
  
 */\*\*  
 \* Creates a new object with the specified constructor and params.  
 \*  
 \** ***@param*** <*T*>  
 *\* the generic type  
 \** ***@param*** *type  
 \* Object type  
 \** ***@param*** *constructorArgTypes  
 \* Constructor argument types  
 \** ***@param*** *constructorArgs  
 \* Constructor argument values  
 \** ***@return*** *the t  
 \*/* <T> T create(Class<T> type, List<Class<?>> constructorArgTypes, List<Object> constructorArgs);  
  
 */\*\*  
 \* Returns true if this object can have a set of other objects.  
 \* It's main purpose is to support non-java.util.Collection objects like Scala collections.  
 \*  
 \** ***@param*** <*T*>  
 *\* the generic type  
 \** ***@param*** *type  
 \* Object type  
 \** ***@return*** *whether it is a collection or not  
 \** ***@since*** *3.1.0  
 \*/* <T> boolean isCollection(Class<T> type);  
  
}

# ####2: org.apache.ibatis.reflection.wrapper.DefaultObjectWrapperFactory

public class DefaultObjectWrapperFactory implements ObjectWrapperFactory <-####2i

# ####2i: org.apache.ibatis.reflection.wrapper.ObjectWrapperFactory

public interface ObjectWrapperFactory {  
  
 boolean hasWrapperFor(Object object);  
  
 ObjectWrapper getWrapperFor(MetaObject metaObject, Object object);  
  
}

# ####3: org.apache.ibatis.reflection.DefaultReflectorFactory

public class DefaultReflectorFactory implements ReflectorFactory <-####3i

# ####3i: org.apache.ibatis.reflection.ReflectorFactory

public interface ReflectorFactory {  
  
 boolean isClassCacheEnabled();  
  
 void setClassCacheEnabled(boolean classCacheEnabled);  
  
 Reflector findForClass(Class<?> type);  
}

# ####4: org.apache.ibatis.reflection.MetaObject#forObject

public static MetaObject forObject(Object object, ObjectFactory objectFactory, ObjectWrapperFactory objectWrapperFactory, ReflectorFactory reflectorFactory) {  
 if (object == null) {  
 return SystemMetaObject.*NULL\_META\_OBJECT*; <-#####1  
 } else {  
 return new MetaObject(object, objectFactory, objectWrapperFactory, reflectorFactory); <-#####2  
 }  
}

# #####1: org.apache.ibatis.reflection.SystemMetaObject#NULL\_META\_OBJECT

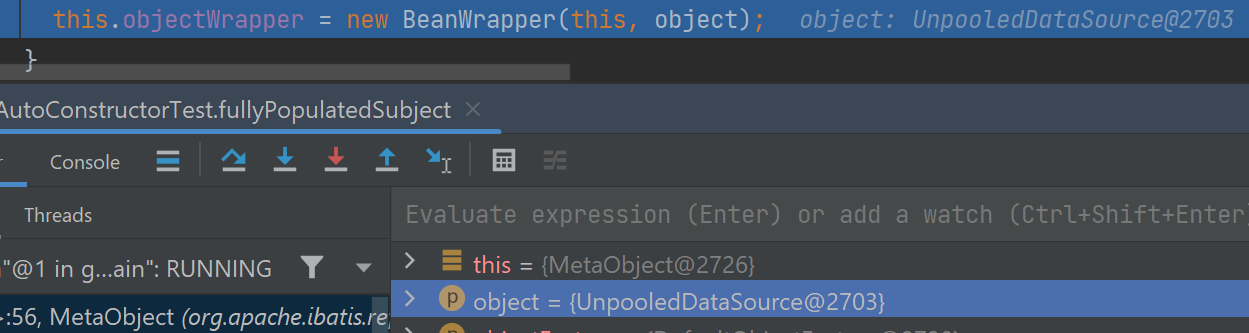
public static final MetaObject *NULL\_META\_OBJECT* = MetaObject.*forObject*(new NullObject(), *DEFAULT\_OBJECT\_FACTORY*, *DEFAULT\_OBJECT\_WRAPPER\_FACTORY*, new DefaultReflectorFactory());

private static class NullObject {  
}

# #####2: org.apache.ibatis.reflection.MetaObject#MetaObject

private final Object originalObject;  
private final ObjectWrapper objectWrapper;  
private final ObjectFactory objectFactory;  
private final ObjectWrapperFactory objectWrapperFactory;  
private final ReflectorFactory reflectorFactory;

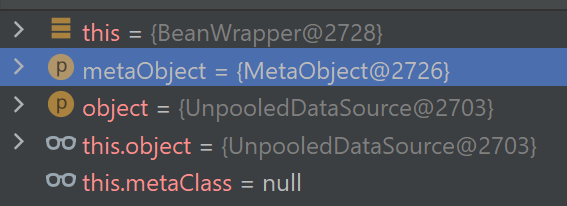
private MetaObject(Object object, ObjectFactory objectFactory, ObjectWrapperFactory objectWrapperFactory, ReflectorFactory reflectorFactory) {  
 this.originalObject = object;  
 this.objectFactory = objectFactory;  
 this.objectWrapperFactory = objectWrapperFactory;  
 this.reflectorFactory = reflectorFactory;  
  
 if (object instanceof ObjectWrapper) {  
 this.objectWrapper = (ObjectWrapper) object;  
 } else if (objectWrapperFactory.hasWrapperFor(object)) {  
 this.objectWrapper = objectWrapperFactory.getWrapperFor(this, object);  
 } else if (object instanceof Map) {  
 this.objectWrapper = new MapWrapper(this, (Map) object);  
 } else if (object instanceof Collection) {  
 this.objectWrapper = new CollectionWrapper(this, (Collection) object);  
 } else {  
 this.objectWrapper = new BeanWrapper(this, object); <-######1 The debug will go here.  
 }  
}



# ######1: org.apache.ibatis.reflection.wrapper.BeanWrapper#BeanWrapper

public class BeanWrapper extends BaseWrapper

private final Object object;  
private final MetaClass metaClass;  
  
public BeanWrapper(MetaObject metaObject, Object object) {  
 super(metaObject); <-#######1  
 this.object = object;  
 this.metaClass = MetaClass.*forClass*(object.getClass(), metaObject.getReflectorFactory() <-#######2); //<-#######3  
}



# #######1: org.apache.ibatis.reflection.wrapper.BaseWrapper#BaseWrapper

public abstract class BaseWrapper implements ObjectWrapper <-#######1i

protected final MetaObject metaObject;

protected BaseWrapper(MetaObject metaObject) {  
 this.metaObject = metaObject;  
}

# #######1i: org.apache.ibatis.reflection.wrapper.ObjectWrapper

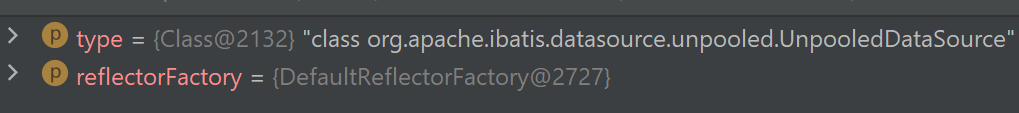
public interface ObjectWrapper {  
  
 Object get(PropertyTokenizer prop);  
  
 void set(PropertyTokenizer prop, Object value);  
  
 String findProperty(String name, boolean useCamelCaseMapping);  
  
 String[] getGetterNames();  
  
 String[] getSetterNames();  
  
 Class<?> getSetterType(String name);  
  
 Class<?> getGetterType(String name);  
  
 boolean hasSetter(String name);  
  
 boolean hasGetter(String name);  
  
 MetaObject instantiatePropertyValue(String name, PropertyTokenizer prop, ObjectFactory objectFactory);  
  
 boolean isCollection();  
  
 void add(Object element);  
  
 <E> void addAll(List<E> element);  
  
}

# #######2: org.apache.ibatis.reflection.MetaObject#getReflectorFactory

public ReflectorFactory getReflectorFactory() {  
 return reflectorFactory; // Written on #####2  
}

# #######3: org.apache.ibatis.reflection.MetaClass#forClass

public static MetaClass forClass(Class<?> type, ReflectorFactory reflectorFactory) {  
 return new MetaClass(type, reflectorFactory); <-########1  
}



# ########1: org.apache.ibatis.reflection.MetaClass#MetaClass

private final ReflectorFactory reflectorFactory;  
private final Reflector reflector;  
  
private MetaClass(Class<?> type, ReflectorFactory reflectorFactory) {  
 this.reflectorFactory = reflectorFactory;  
 this.reflector = reflectorFactory.findForClass(type); <-#########1  
}

# #########1: org.apache.ibatis.reflection.DefaultReflectorFactory#findForClass

private boolean classCacheEnabled = true;  
private final ConcurrentMap<Class<?>, Reflector> reflectorMap = new ConcurrentHashMap<>();

@Override  
public Reflector findForClass(Class<?> type) {  
 if (classCacheEnabled) {  
 // synchronized (type) removed see issue #461  
 return MapUtil.*computeIfAbsent*(reflectorMap, type, Reflector::new); <-##########1 The debug will go here.  
 } else {  
 return new Reflector(type);  
 }  
}

# ##########1: org.apache.ibatis.reflection.Reflector

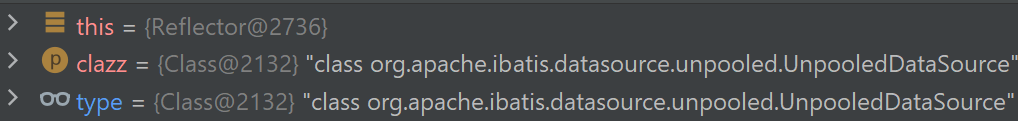
private final Class<?> type;

private final String[] readablePropertyNames;  
private final String[] writablePropertyNames;

private final Map<String, Invoker> setMethods = new HashMap<>();  
private final Map<String, Invoker> getMethods = new HashMap<>();

private Map<String, String> caseInsensitivePropertyMap = new HashMap<>();

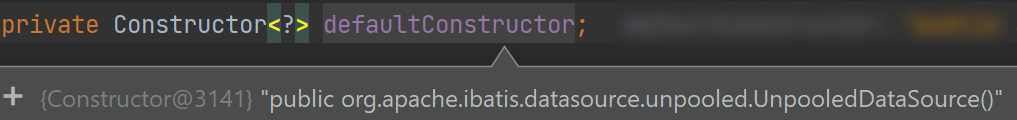
public Reflector(Class<?> clazz) {  
 type = clazz;  
 addDefaultConstructor(clazz); <-###########1  
 Method[] classMethods = getClassMethods(clazz); <-###########2  
 addGetMethods(classMethods); <-###########3  
 addSetMethods(classMethods); <-###########4  
 addFields(clazz); <-###########5  
 readablePropertyNames = getMethods.keySet().toArray(new String[0]);  
 writablePropertyNames = setMethods.keySet().toArray(new String[0]);  
 for (String propName : readablePropertyNames) {  
 caseInsensitivePropertyMap.put(propName.toUpperCase(Locale.*ENGLISH*), propName);  
 }  
 for (String propName : writablePropertyNames) {  
 caseInsensitivePropertyMap.put(propName.toUpperCase(Locale.*ENGLISH*), propName);  
 }  
}



# ###########1: org.apache.ibatis.reflection.Reflector#addDefaultConstructor

private Constructor<?> defaultConstructor;

private void addDefaultConstructor(Class<?> clazz) {  
 Constructor<?>[] constructors = clazz.getDeclaredConstructors();  
 Arrays.*stream*(constructors).filter(constructor -> constructor.getParameterTypes().length == 0)  
 .findAny().ifPresent(constructor -> this.defaultConstructor = constructor);  
}

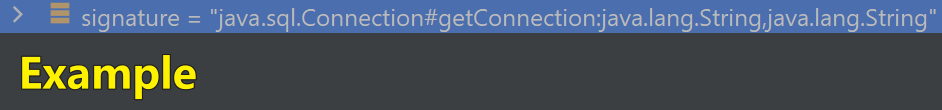


# ###########2: org.apache.ibatis.reflection.Reflector#getClassMethods

*/\*\*  
 \* This method returns an array containing all methods  
 \* declared in this class and any superclass.  
 \* We use this method, instead of the simpler <code>Class.getMethods()</code>,  
 \* because we want to look for private methods as well.  
 \*  
 \** ***@param*** *clazz The class  
 \** ***@return*** *An array containing all methods in this class  
 \*/*private Method[] getClassMethods(Class<?> clazz) {  
 Map<String, Method> uniqueMethods = new HashMap<>();  
 Class<?> currentClass = clazz;  
 while (currentClass != null && currentClass != Object.class) {  
 addUniqueMethods(uniqueMethods, currentClass.getDeclaredMethods()); <-############1  
  
 // we also need to look for interface methods -  
 // because the class may be abstract  
 Class<?>[] interfaces = currentClass.getInterfaces();  
 for (Class<?> anInterface : interfaces) {  
 addUniqueMethods(uniqueMethods, anInterface.getMethods()); <-############1  
 }  
  
 currentClass = currentClass.getSuperclass();  
 }  
  
 Collection<Method> methods = uniqueMethods.values();  
  
 return methods.toArray(new Method[0]);  
}

# ############1: org.apache.ibatis.reflection.Reflector#addUniqueMethods

private void addUniqueMethods(Map<String, Method> uniqueMethods, Method[] methods) {  
 for (Method currentMethod : methods) {  
 if (!currentMethod.isBridge()) {  
 String signature = getSignature(currentMethod); <-#############1  
 // check to see if the method is already known  
 // if it is known, then an extended class must have  
 // overridden a method  
 if (!uniqueMethods.containsKey(signature)) {  
 uniqueMethods.put(signature, currentMethod);  
 }  
 }  
 }  
}



# #############1: org.apache.ibatis.reflection.Reflector#getSignature

private String getSignature(Method method) {  
 StringBuilder sb = new StringBuilder();  
 Class<?> returnType = method.getReturnType();  
 if (returnType != null) {  
 sb.append(returnType.getName()).append('#');  
 }  
 sb.append(method.getName());  
 Class<?>[] parameters = method.getParameterTypes();  
 for (int i = 0; i < parameters.length; i++) {  
 sb.append(i == 0 ? ':' : ',').append(parameters[i].getName());  
 }  
 return sb.toString();  
}

# ###########3: org.apache.ibatis.reflection.Reflector#addGetMethods

private void addGetMethods(Method[] methods) {  
 Map<String, List<Method>> conflictingGetters = new HashMap<>();  
 Arrays.*stream*(methods).filter(m -> m.getParameterTypes().length == 0 && PropertyNamer.*isGetter*(m.getName()))  
 .forEach(m -> addMethodConflict(conflictingGetters, PropertyNamer.*methodToProperty*(m.getName()), m) <-############1);  
 resolveGetterConflicts(conflictingGetters); <-############2   
}

# ############1: org.apache.ibatis.reflection.Reflector#addMethodConflict

private void addMethodConflict(Map<String, List<Method>> conflictingMethods, String name, Method method) {  
 if (isValidPropertyName(name)) { <-#############1  
 List<Method> list = MapUtil.*computeIfAbsent*(conflictingMethods, name, k -> new ArrayList<>());  
 list.add(method);  
 }  
}

# #############1: org.apache.ibatis.reflection.Reflector#isValidPropertyName

private boolean isValidPropertyName(String name) {  
 return !(name.startsWith("$") || "serialVersionUID".equals(name) || "class".equals(name));  
}

# ############2: org.apache.ibatis.reflection.Reflector#resolveGetterConflicts

private void resolveGetterConflicts(Map<String, List<Method>> conflictingGetters) {  
 for (Entry<String, List<Method>> entry : conflictingGetters.entrySet()) {  
 Method winner = null;  
 String propName = entry.getKey();  
 boolean isAmbiguous = false;  
 for (Method candidate : entry.getValue()) {  
 if (winner == null) {  
 winner = candidate;  
 continue;  
 }  
 Class<?> winnerType = winner.getReturnType();  
 Class<?> candidateType = candidate.getReturnType();  
 if (candidateType.equals(winnerType)) {  
 if (!boolean.class.equals(candidateType)) {  
 isAmbiguous = true;  
 break;  
 } else if (candidate.getName().startsWith("is")) {  
 winner = candidate;  
 }  
 } else if (candidateType.isAssignableFrom(winnerType)) {  
 // OK getter type is descendant  
 } else if (winnerType.isAssignableFrom(candidateType)) {  
 winner = candidate;  
 } else {  
 isAmbiguous = true;  
 break;  
 }  
 }  
 addGetMethod(propName, winner, isAmbiguous); <-#############1  
 }  
}