如果是ibatis的时代，直接通过sqlSession对象调用方法，然后执行器等后续执行；mybatis的时候，我们可以不用操作sqlSession直接使用mapper接口。

1. sqlSeesion.getMapper(Class<T> c)返回的对象是什么

```
SysRoleMapper sysRoleMapper = sqlSession.getMapper(SysRoleMapper.class);
```

在MapperProxyFactory中发现这是通过jdk生成的代理对象

```
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);
}
```

根据对jdk动态代理的理解，这个mapper的逻辑会落在代理执行程序上（newProxyInstance的第三个参数，实现了InvocationHandler），可以看到这个类是MapperProxy

2. MapperProxy的Invoke方法

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
 try {
 // method.getDeclaringClass()返回执行方法的对象
 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);
 }
```

2.1 if (Object.class.equals(method.getDeclaringClass())) 这一句判断的是调用的是mapper接口的方法还是Object的方法，method.getDeclaringClass()表示方法所属的类或接口。

2.2 else if (isDefaultMethod(method)) 判断是不是jdk1.8中有默认实现的方法

2.3 正常调用mapper接口中的方法，是会落在最后两句的

final MapperMethod mapperMethod = cachedMapperMethod(method);

return mapperMethod.execute(sqlSession, args);

3. mapperMethod.execute(sqlSession, 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);
  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();
```

```
36   break;
37   default:
38   throw new BindingException("Unknown execution method for: " + command.g
etName());
39   }
40   if (result == null && method.getReturnType().isPrimitive() && !method.r
eturnsVoid()) {
41   throw new BindingException("Mapper method '" + command.getName()
42   + " attempted to return null from a method with a primitive return type
(" + method.getReturnType() + ").");
43   }
44   return result;
45  }
```

execute方法中，下面两句就是本次执行的落点

Object param = method.convertArgsToSqlCommandParam(args);

result = sqlSession.selectOne(command.getName(), param);

其中第一句是将method的参数专称map对象，以便后续使用，具体规则如下

```
1  public ParamNameResolver(Configuration config, Method method) {
2   final Class<?>[] paramTypes = method.getParameterTypes();
3   final Annotation[][] paramAnnotations =
method.getParameterAnnotations();
4   final SortedMap<Integer, String> map = new TreeMap<Integer, String>();
5   int paramCount = paramAnnotations.length;
6   // get names from @Param annotations
7   for (int paramIndex = 0; paramIndex < paramCount; paramIndex++) {
8   if (isSpecialParameter(paramTypes[paramIndex])) {
9   // skip special parameters
10   continue;
11   }
12   String name = null;
13   for (Annotation annotation : paramAnnotations[paramIndex]) {
14   if (annotation instanceof Param) {
15   hasParamAnnotation = true;
16   name = ((Param) annotation).value();
17   break;
18   }
19   }
20   if (name == null) {
21   // @Param was not specified.
22   if (config.isUseActualParamName()) {
```

```
23    name = getActualParamName(method, paramIndex);
24    }
25    if (name == null) {
26    // use the parameter index as the name ("0", "1", ...)
27    // gcode issue #71
28    name = String.valueOf(map.size());
29    }
30    }
31    map.put(paramIndex, name);
32    }
33    names = Collections.unmodifiableSortedMap(map);
34  }
```

最终的结果，如果配置了@Param优先使用，否则使用参数名作为map的key（useActualParamName可以配置，默认为true），如果useActualParamName配置为false，则使用“0”，“1”作为key。

然后就是执行sqlSession的selectOne方法了

至此就从mybatis进入了itbatis，mapper接口的的作用就完了