resultType

resultType 与 parameterType 的使用基本一致。

1、基本数据类型

```
public int count();
```

```
<select id="count" resultType="int">
    select count(*) from people
</select>
```

2、包装类

```
public Integer count();
```

```
<select id="count" resultType="java.lang.Integer">
    select count(*) from people
</select>
```

3、String

```
public String findNameById(Integer id);
```

```
<select id="findNameById" parameterType="java.lang.Integer"
resultType="java.lang.String">
    select name from people where id = #{id}
</select>
```

4、POJO

```
public People findById(Integer id);
```

```
<select id="findById" parameterType="java.lang.Integer"
resultType="com.southwind.entity.People">
    select * from people where id = #{id}
</select>
```

多表关联查询

实际开发中最常用的是：一对多和多对多

一对多

1、建表

```
use test;
create table `t_classes`(
  `id` int(11) NOT NULL primary key auto_increment,
  `name` varchar(11) default null
);

create table `t_student`(
  `id` int(11) not null primary key auto_increment,
  `name` varchar(11) default null,
  `cid` int(11) default null,
  key `cid` (`cid`),
  constraint `t_student_ibfk_1` foreign key (`cid`) references
`t_classes`(`id`)
)
```

2、SQL

```
select s.id sid,s.name sname,c.id cid,c.name cname from t_student s,t_classes
c where s.id = 1 and s.cid = c.id
```

3、创建实体类

```
package com.southwind.entity;

import lombok.Data;

@Data
public class Student {
    private Integer id;
    private String name;
    private Classes classes;
}
```

@Data帮助我们生成getter和setter以及tostring方法

```java
package com.southwind.entity;

import lombok.Data;

import java.util.List;

@Data
public class Classes {
    private Integer id;
    private String name;
    private List<Student> students;
}
```

4、StudentRepository

```java
package com.southwind.repository;

import com.southwind.entity.Student;

public interface StudentRepository {
    public Student findById(Integer id);
}
```

5、StudentRepository.xml

resultType 直接将结果集与实体类进行映射，结果集的字段名与实体类的成员变量名相等则映射。

resultMap 可以对结果集进行二次封装，根据需求来完成结果集数据到实体类的映射。

二次封装就是把c.id,c.name封装映射成实体类

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.southwind.repository.StudentRepository">

    <resultMap id="studentMap" type="com.southwind.entity.Student">
        <id column="sid" property="id"></id>
        <result column="sname" property="name"></result>
        <association property="classes"
javaType="com.southwind.entity.Classes">
            <id property="id" column="cid"></id>
            <result property="name" column="cname"></result>
        </association>
    </resultMap>

    <select id="findById" parameterType="java.lang.Integer"
resultMap="studentMap">
        select s.id sid,s.name sname,c.id cid,c.name cname from t_student
s,t_classes c where s.id = 1 and s.cid = c.id
    </select>
```

id是与下面的resultmap名关联

这个例子是通过多的一方查询一的一方

多个学生对应一个班级

结果集中的col

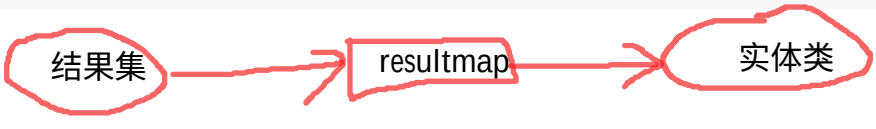
实体类的属性

resultmap最终还是与一个实体类对应起来

这个type就是那个要对应的实体类

association把结果集中的c.id和c.name映射到student中的class属性中

我们可以使用resultmap来对返回的结果进行二次封装



```
</mapper>
```

6、ClassesRepository

```
package com.southwind.repository;

import com.southwind.entity.Classes;

public interface ClassesRepository {
    public Classes findById(Integer id);
}
```

7、ClassesRepository.xml　　association是用于一对一和多对一，而collection是用于一对多的关系

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.southwind.repository.ClassesRepository">

    <resultMap id="classesMap" type="com.southwind.entity.Classes">
        <id property="id" column="cid"></id>
        <result property="name" column="cname"></result>
        <collection property="students" ofType="com.southwind.entity.Student">
            <id property="id" column="sid"></id>
            <result property="name" column="sname"></result>
        </collection>
    </resultMap>

    <select id="findById" parameterType="java.lang.Integer"
resultMap="classesMap">
        select c.id cid,c.name cname,s.id sid,s.name sname from t_classes
c,t_student s where c.id = 1 and c.id = s.cid
    </select>

</mapper>
```

通过一的一方查询多的一方一个班级对应多个学生

多个学生用colletction集合然后把这个学生集合映射到class中的list

collection 和 association 的区别

collection 是将结果集封装成一个集合对象（多个目标对象）

association 是将结果集封装成一个实体类的对象（一个目标对象）

collection 是通过 ofType 设置数据类型，association 是通过 javaType 设置数据类型。

多对多

多对多是双向的一对多关系　　把一对多理解明白这个就是双向的一对多

1、建表

```
create table `t_account`(
  `id` int(11) not null primary key auto_increment,
  `name` varchar(11) default null
);

create table `t_course`(
  `id` int(11) not null primary key auto_increment,
  `name` varchar(11) default null
);

create table `account_course`(
  `id` int(11) not null primary key auto_increment,
  `aid` int(11) default null,
  `cid` int(11) default null,
  key `aid`(`aid`),
  key `cid`(`cid`),
  constraint `account_course_ibfk_1` foreign key (`aid`) references
`t_account`(`id`),
  constraint `account_course_ibfk_2` foreign key (`cid`) references
`t_course`(`id`)
);
```

2、创建实体类

```
package com.southwind.entity;

import lombok.Data;

import java.util.List;

@Data
public class Account {          Account对多个Course
    private Integer id;
    private String name;
    private List<Course> courses;
}
```

```java
package com.southwind.entity;

import lombok.Data;

import java.util.List;

@Data
public class Course {
    private Integer id;
    private String name;
    private List<Account> accounts;
}
```

Course对多个Account

3、AccountRepository

```java
package com.southwind.repository;

import com.southwind.entity.Account;

public interface AccountRepository {
    public Account findById(Integer id);
}
```

4、AccountRepository.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.southwind.repository.AccountRepository">

    <resultMap id="accoutMap" type="com.southwind.entity.Account">
        <id column="aid" property="id"></id>
        <result column="aname" property="name"></result>
        <collection property="courses" ofType="com.southwind.entity.Course">
            <id column="cid" property="id"/>
            <result column="cname" property="name"/>
        </collection>
    </resultMap>

    <select id="findById" parameterType="java.lang.Integer"
resultMap="accoutMap">
        select a.id aid,a.name aname,c.id cid,c.name cname from t_account
a,account_course ac,t_course c where a.id = #{id} and a.id = ac.aid and c.id =
ac.cid
    </select>

</mapper>
```

5、CourseRepository

```
package com.southwind.repository;

import com.southwind.entity.Course;

public interface CourseRepository {
    public Course findById(Integer id);
}
```

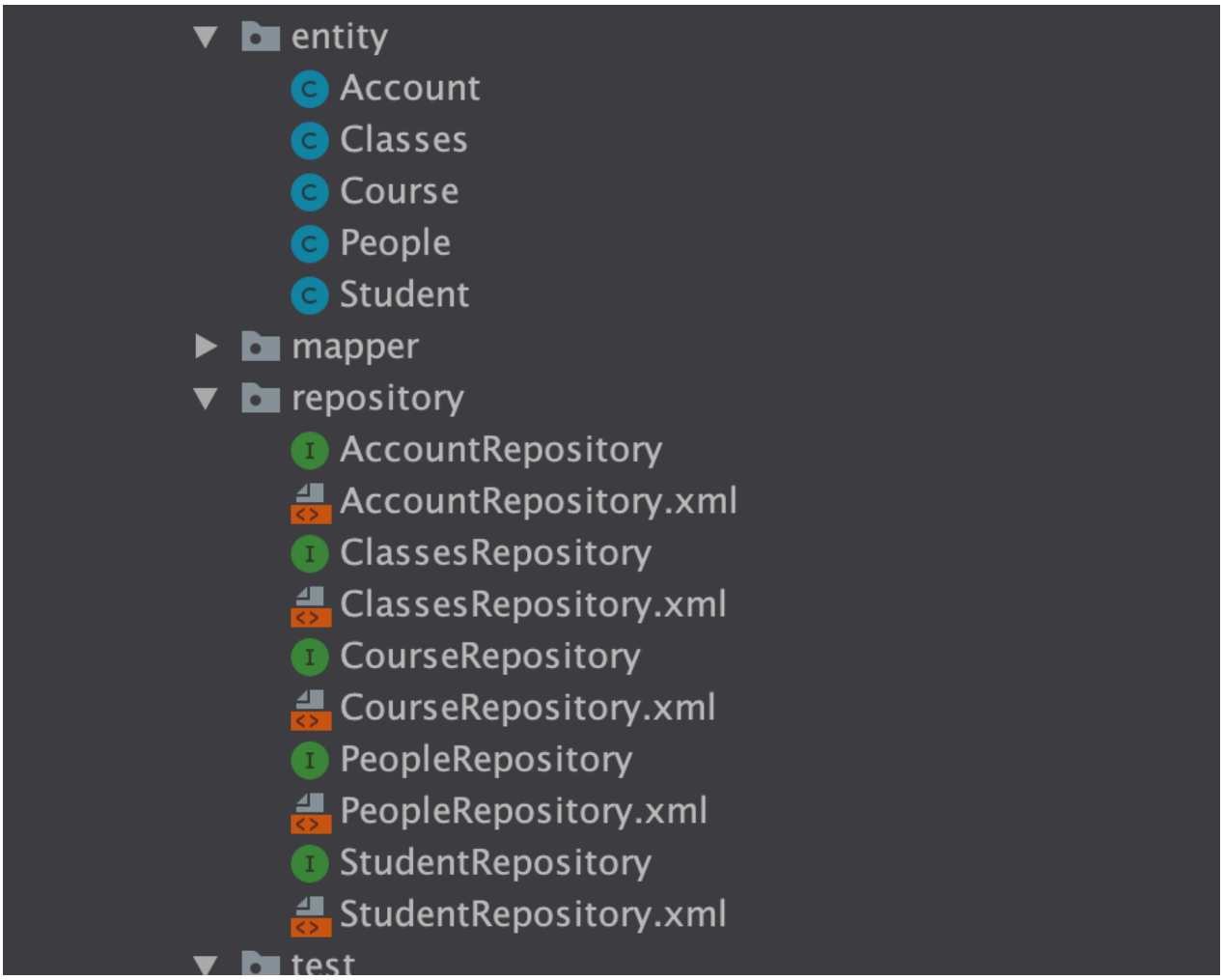
6、CourseRepository.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.southwind.repository.CourseRepository">

    <resultMap id="courseMap" type="com.southwind.entity.Course">
        <id column="cid" property="id"></id>
        <result column="cname" property="name"></result>
        <collection property="accounts" ofType="com.southwind.entity.Account">
            <id column="aid" property="id"/>
            <result column="aname" property="name"/>
        </collection>
    </resultMap>

    <select id="findById" parameterType="java.lang.Integer"
resultMap="courseMap">
        select a.id aid,a.name aname,c.id cid,c.name cname from t_account
a,account_course ac,t_course c where c.id = #{id} and a.id = ac.aid and c.id =
ac.cid
    </select>

</mapper>
```

MyBatis 逆向工程

MyBatis 是半自动化的 ORM 框架，SQL 语句需要开发者自定义，SQL 需要单独定义在 Mapper.xml 中，与 Mapper 接口对应，使用 MyBatis 进行开发的基本配置：

- 实体类
- Mapper 接口
- Mapper.xml



这种方法的缺陷是如果参与业务的表太多，每张表的业务都需要自定义 SQL、创建实体类、Maper 接口，工作量较大。

MyBatis 框架可以自动根据数据表，帮助开发者生成实体类、Mapper 接口、Mapper.xml，这就是逆向工程。

逆向工程概念

逆向工程是 MyBatis 提供的一种自动化配置方案，针对数据表自动生成 MyBatis 所需要的各种资源（实体类、Mapper 接口、Mapper.xml），但是逆向工程只针对于单表，如果数据表之间有级联关系，逆向工程无法自动生成级联关系。

使用逆向工程

MyBatis 逆向工程的组件是 MyBatis Generator，简称 MBG，是专为 MyBatis 框架定制的代码自动生成解决方案，MBG 可以根据数据表结构快速生成对应的实体类、Mapper接口、Mapper.xml，并且支持基本的 CRUD 操作，但是业务逻辑相对复杂的操作就需要开发者手动完成。

1、创建 Maven 工程，pom.xml 添加相关依赖。

```
<dependencies>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
```

```
        <version>3.4.5</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis.generator</groupId>
        <artifactId>mybatis-generator-core</artifactId>
        <version>1.3.2</version>
    </dependency>
</dependencies>
```

2、创建目标表 t_account。

```
create table `t_account`(
  `id` int(11) not null primary key auto_increment,
  `name` varchar(11),
  `password` varchar(11),
  `age` int(11)
)
```

3、创建 MBG 配置文件 generatorConfig.xml

- jdbcConnection 配置数据库连接信息
- javaModelGenerator 配置 JavaBean 的生成策略
- sqlMapGenerator 配置 SQL 映射文件生成策略
- javaClientGenerator 配置 Mapper 接口的生成策略
- table 配置要逆向解析的数据表（tableName：表名，domainObjectName：实体类名）

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
        PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <context id="testTables" targetRuntime="MyBatis3">
        <jdbcConnection
                driverClass="com.mysql.cj.jdbc.Driver"
                connectionURL="jdbc:mysql://localhost:3306/test?
useUnicode=true&amp;characterEncoding=UTF-8"
                userId="root"
                password="root"
        ></jdbcConnection>
        <javaModelGenerator targetPackage="com.southwind.entity"
targetProject="./src/main/java"></javaModelGenerator>
```

```
        <sqlMapGenerator targetPackage="com.southwind.repository"
targetProject="./src/main/java"></sqlMapGenerator>
        <javaClientGenerator type="XMLMAPPER"
targetPackage="com.southwind.repository" targetProject="./src/main/java">
</javaClientGenerator>
        <table tableName="t_account" domainObjectName="Account"></table>
    </context>
</generatorConfiguration>
```

4、创建 GeneratorMain 类，执行自动生成资源的代码。

```
package com.southwind.test;

import org.mybatis.generator.api.MyBatisGenerator;
import org.mybatis.generator.config.Configuration;
import org.mybatis.generator.config.xml.ConfigurationParser;
import org.mybatis.generator.exception.InvalidConfigurationException;
import org.mybatis.generator.exception.XMLParserException;
import org.mybatis.generator.internal.DefaultShellCallback;

import java.io.File;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class GeneratorMain {
    public static void main(String[] args) {
        List<String> warings = new ArrayList<String>();
        boolean overwrite = true;
        String genCig = "/generatorConfig.xml";
        File configFile = new
File(GeneratorMain.class.getResource(genCig).getFile());
        ConfigurationParser configurationParser = new
ConfigurationParser(warings);
        Configuration configuration = null;
        try {
            configuration =
configurationParser.parseConfiguration(configFile);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (XMLParserException e) {
            e.printStackTrace();
        }
        DefaultShellCallback callback = new DefaultShellCallback(overwrite);
        MyBatisGenerator myBatisGenerator = null;
        try {
            myBatisGenerator = new
MyBatisGenerator(configuration,callback,warings);
```

```
        } catch (InvalidConfigurationException e) {
            e.printStackTrace();
        }
        try {
            myBatisGenerator.generate(null);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```