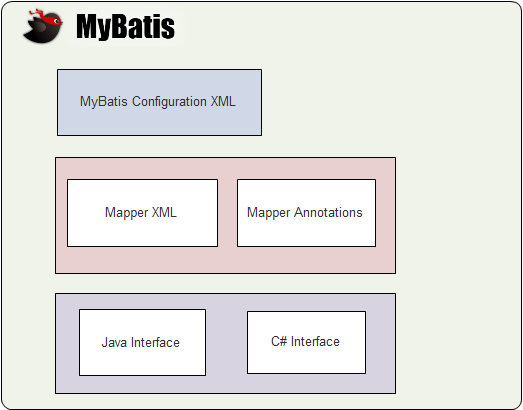
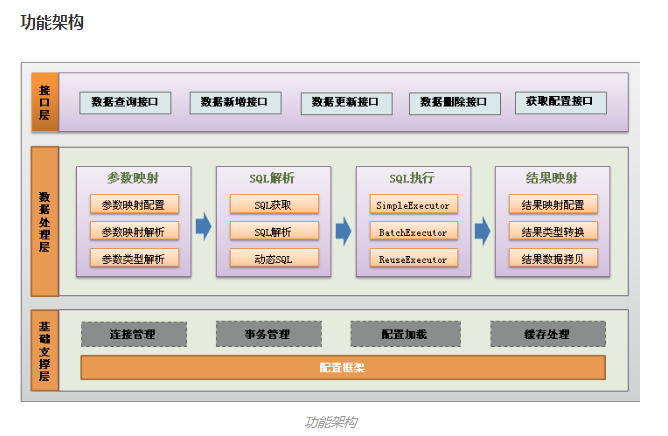
# 第一部分 知识准备篇

## 第一章 MyBatis概览

MyBatis总体结构图



[Java框架篇---Mybatis 入门](http://www.jianshu.com/p/3f507c7ec3da)

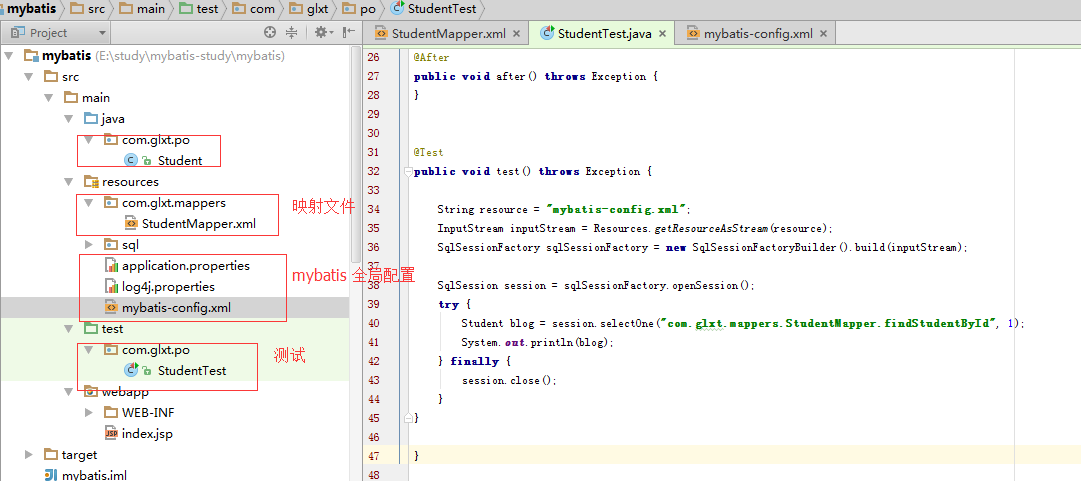


## 第二章 MyBatis初体验

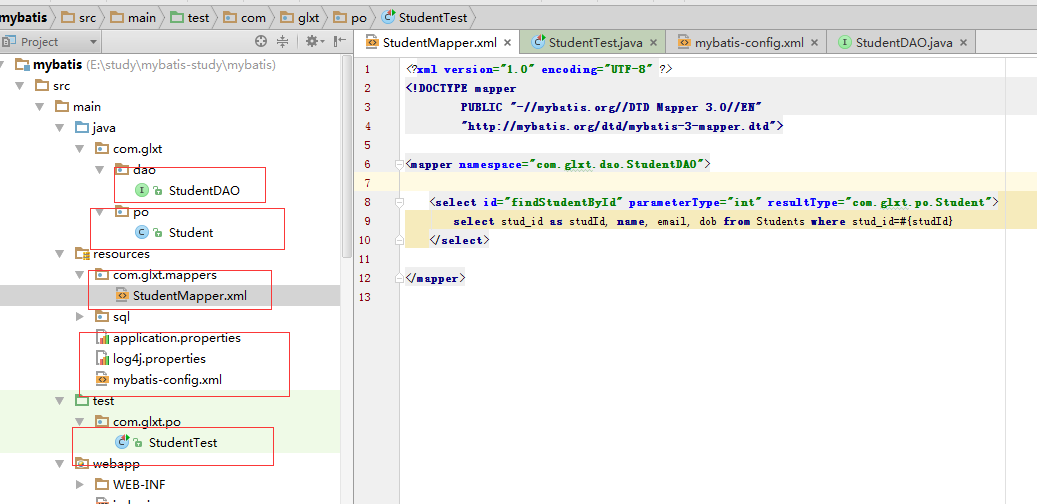
MyBatis 最关键的组成部分是 SqlSessionFactory，我们可以从中获取 SqlSession，并执行映射的 SQL 语句。  
SqlSessionFactory 对象可以通过基于 XML 的配置信息或者 Java API 创建。

### 官方示例

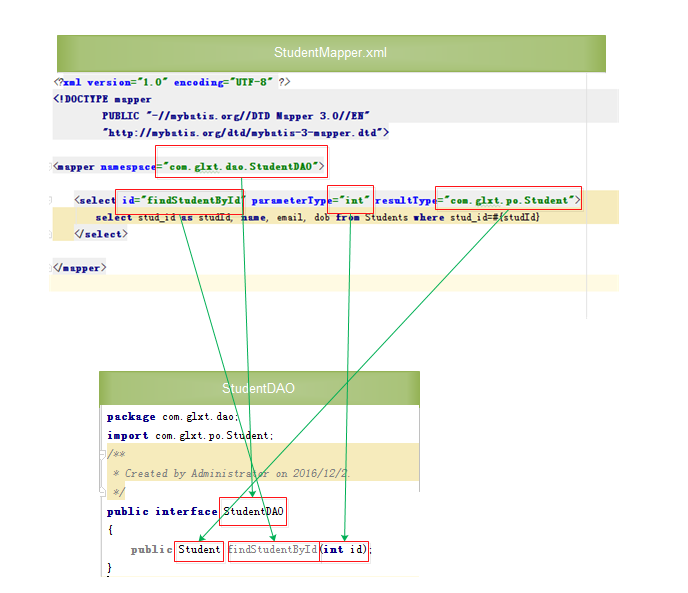
#### 1、不使用接口映射情况



#### 2、使用接口映射情况

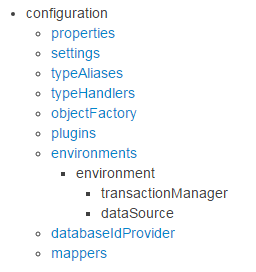


* **StudentMapper.xml与StudentDAO映射关系**



### MyBatis全局配置（Configuration）

The MyBatis configuration（配置） contains settings （设置）and properties（属性） that have a dramatic effect on how MyBatis behaves. 结构如下：



#### environments

##### environment

必须配置的

例如

|  |
| --- |
| <environments default="development">  <environment id="development">  <transactionManager type="JDBC"/>  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver"/>  <property name="url" value="jdbc:mysql://localhost:3306/mybatis\_test"/>  <property name="username" value="root"/>  <property name="password" value="root"/>  </dataSource>  </environment>  </environments> |

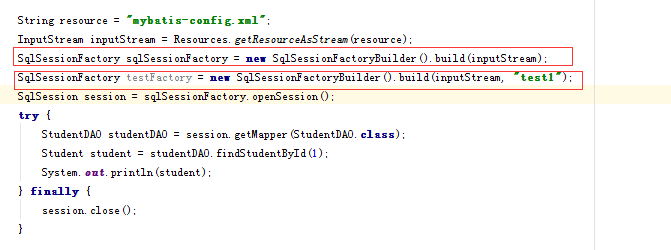
MyBatis 支持配置多个 dataSource 环境，可以将应用部署到不同的环境上，如 DEV(开发环境) ， TEST（测试换将），QA（质量评估环境） ,UAT( 用户验收环境),PRODUCTION（生产环境），可以通过将默认 environment 值设置成想要的environment id 值。

在上述的配置中，默认的环境 environment 被设置成 development。当需要将程序部署到生产服务器上时，你不需要修改什么配置，只需要将默认环境 environment 值设置成生产环境的 environment id 属性即可。

多个环境配置



同时创建多个SqlSessionFactory实例测试



注意：

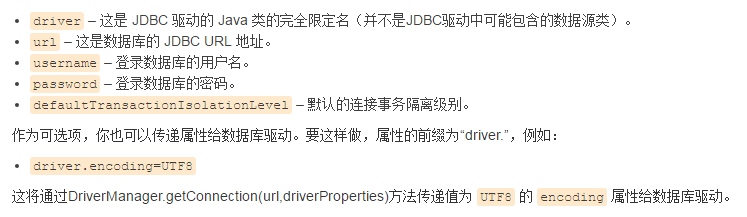
创建多个SqlSessionFactory实例，失败。暂时只能创建单个实例。

##### dataSource

* dataSource元素被用来配置数据库连接属性。
* dataSource 元素使用标准的 JDBC 数据源接口来配置 JDBC 连接对象的资源。

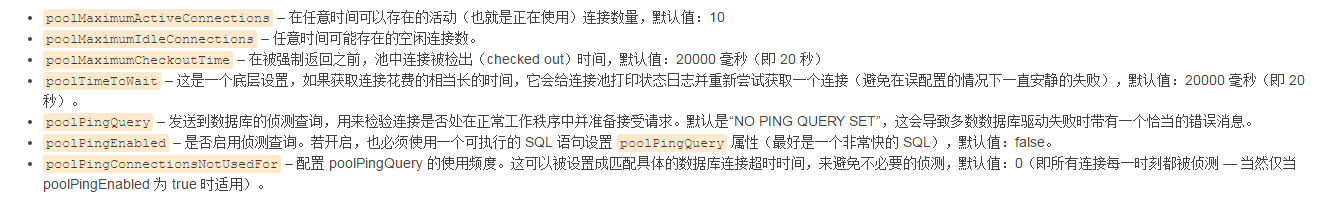
有三种内建的数据源类型：UNPOOLED、POOLED、JNDI。

**UNPOOLED**– 这个数据源的实现只是每次被请求时打开和关闭连接。UNPOOLED 类型的数据源仅仅需要配置以下 5 种属性：

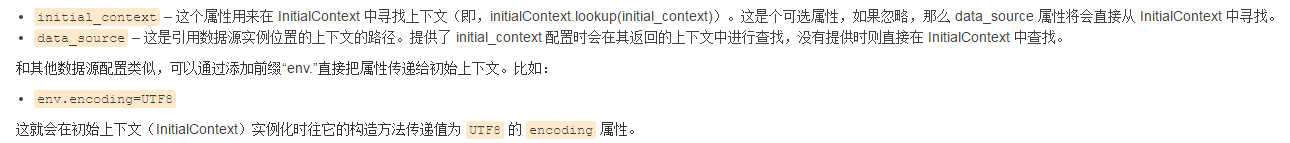


**POOLED**– 这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来，避免了创建新的连接实例时所必需的初始化和认证时间。 这是一种使得并发 Web 应用快速响应请求的流行处理方式。

除了上述提到 UNPOOLED 下的属性外，会有更多属性用来配置 POOLED 的数据源：



**JNDI**– 这个数据源的实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的引用。这种数据源配置只需要两个属性：



##### TransactionManager

MyBatis支持两种类型的事务管理器：JDBC和MANAGED。

* JDBC事务管理器被用作当应用程序负责管理数据库连接的生命周期（提交、回退等等）的时候。例如，部署到 Apache Tomcat 的应用程序，需要应用程序自己管理事务。
* MANAGED事 务 管 理 器 是 当 由 应 用 服 务 器 负 责 管 理 数 据 库 连 接 生 命 周 期 的 时 候 使 用。例如，当一个 JavaEE 的应用程序部署在类似 JBoss，WebLogic，  
  GlassFish 应用服务器上时，它们会使用 EJB 进行应用服务器的事务管理能力。

Managed 是托管的意思，即是应用本身不去管理事务，而是把事务管理交给应用所在的服务  
器进行管理。

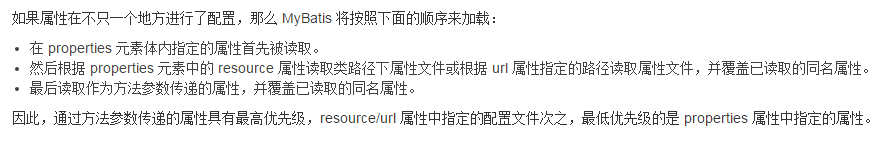
注意：如果你正在使用 Spring + MyBatis，则没有必要配置事务管理器， 因为 Spring 模块会使用自带的管理器来覆盖前面的配置。

#### properties

|  |
| --- |
| <properties resource="application.properties">  <property name="password" value="root" />  </properties>  <environments default="development">  <environment id="development">  <transactionManager type="JDBC"/>  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${password}"/>  </dataSource>  </environment>  </environments> |

* 这个例子中password会由properties元素中设置的相应值来替换。
* driver、url、username属性将会由application.properties文件中对应的值来替换。这样就为配置提供了诸多灵活选择。

注意：



#### settings

这是 MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为。

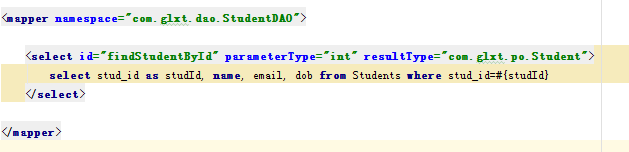
参照中文官网：[settings](http://www.mybatis.org/mybatis-3/zh/configuration.html#settings)

#### typeAliases

在 SQLMapper 配置文件中，对于 resultType 和 parameterType 属性值，我们需要使用 JavaBean 的完全限定名。

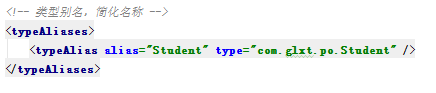
我们可以为完全限定名取一个别名（ alias），然后其需要使用完全限定名的地方使用别名，而不是到处使用完全限定名。

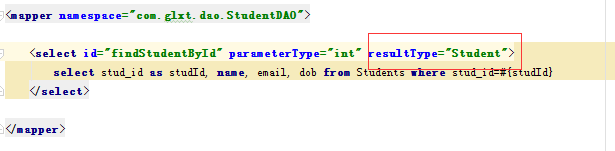
* 不使用别名之前



* 使用别名后

mybatis:



mapper: 

**使用包名方式：**

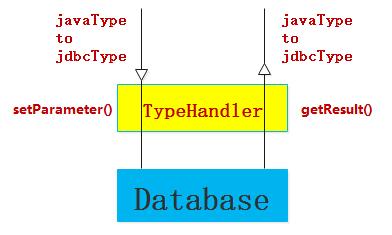
也可以指定一个包名，MyBatis 会在包名下面搜索需要的 Java Bean，比如:



每一个在包 com.glxt.po中的 Java Bean，在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名（mybatis3.4.1不区分首字母的大小写）。 比如 com.glxt.po.Student 的别名为 Student/student；若有注解，则别名为其注解值。

#### typeHandlers

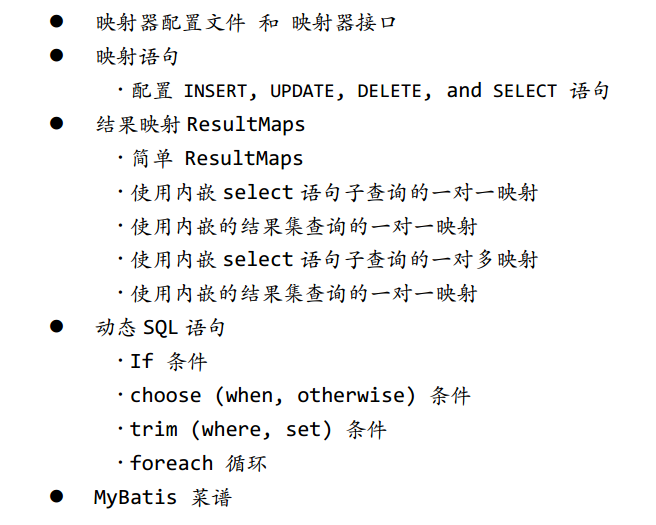
无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型。



### 使用XML配置SQL映射器

在代码里直接嵌套 SQL 语句是很差的编码实践，并且维护起来困难。 MyBaits 使用了映射器配置文件或注解来配置 SQL语句。

#### 概览



#### 映射器配置文件和映射器接口

详见 [使用接口映射情况](#_2、使用接口映射情况)

不使用接口映射，容易出错，你需要检查映射器配置文件中的定义，以保证你的输入参数类型和结果返回类型是有效的。

#### 映射语句

##### 查询（select）

多数应用都是查询比修改要频繁，对每个插入、更新或删除操作，通常对应多个查询操作。这是mybatis的基本原则之一，也是将焦点和努力放到查询和结果映射的原因。

简单查询的select元素是非常简单的。比如：

|  |
| --- |
| <**select id="findStudentById" parameterType="int" resultType="Student"**>  select stud\_id as studId, name, email, dob from Students where stud\_id=#{studId} </**select**> |

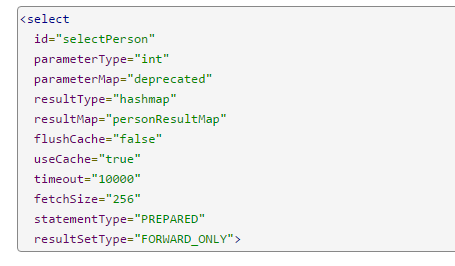
注意参数符号：

#{studId}

这就告诉MyBatis创建一个预处理语句参数，通过JDBC，这样的一个参数在SQL中会由一个“？”来标识，并传递到一个新的预处理语句中，就像这样：

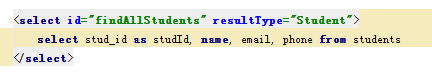
|  |
| --- |
| *// Similar JDBC code, NOT MyBatis..* String selectStudent = **"select stud\_id as studId, name, email, dob from Students where stud\_id=?"**; PreparedStatement ps = conn.prepareStatement(selectStudent); ps.setInt(1, studId); |

select 元素有很多属性允许你配置，来决定每条语句的作用细节。

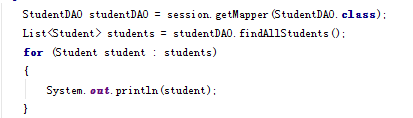




简单版的查询所有结果







##### insert, update 和 delete

数据变更语句 insert，update 和 delete 的实现非常接近.



参照：[insert\_update\_and\_delete](http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html#insert_update_and_delete)

注意：插入、更新、删除操作需要手动提交session。才能成功执行到数据库。

|  |
| --- |
| StudentDAO studentDAO = session.getMapper(StudentDAO.**class**); Student student = **new** Student(33, **"lisi"**, **"liggi@126.com"**, **new** Date()); PhoneNumber phoneNumber = **new** PhoneNumber(**"110"**, **"01"**, **"345"**); student.setPhone(phoneNumber); studentDAO.insertStudent(student); session.commit(); *// 注意需要手动提交* |

#### 结果集映射ResultMaps

resultMap 元素是 MyBatis 中最重要最强大的元素。它就是让你远离 90%的需要从结果 集中取出数据的 JDBC 代码的那个东西, 而且在一些情形下允许你做一些 JDBC 不支持的事 情。 事实上, 编写相似于对复杂语句联合映射这些等同的代码, 也许可以跨过上千行的代码。 ResultMap 的设计就是简单语句不需要明确的结果映射,而很多复杂语句确实需要描述它们 的关系。

ResultMaps 被用来 将 SQL SELECT 语句的结果集映射到 JavaBeans 的属性中。我们可以定义结果集映射 ResultMaps并且在一些 SELECT 语句上引用 resultMap。 MyBatis 的结果集映射 ResultMaps 特性非常强大， 你可以使用它将简单的SELECT 语句映射到复杂的一对一和一对多关系的 SELECT 语句上。

##### 简单的ResultMaps

映射文件：

|  |
| --- |
| <**resultMap id="studentResultMap" type="Student"**>  <**id property="studId" column="stud\_id"**></**id**>  <**result property="name" column="name"**></**result**>  <**result property="email" column="email"**></**result**> </**resultMap**> <**select id="selectStudent" parameterType="int" resultMap="studentResultMap"**>  select stud\_id, name, email from Students where stud\_id=#{studId} </**select**> |

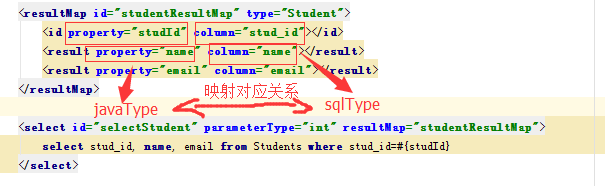
接口：

|  |
| --- |
| **public** List<Student> selectStudent(**int** id);  或者  **public** Student selectStudent(**int** id); |

测试：

|  |
| --- |
| StudentDAO studentDAO = session.getMapper(StudentDAO.**class**); List<Student> students = studentDAO.selectStudent(3); // 或者Student student = … **for** (Student student : students) {  System.***out***.println(student); } |

注意：映射文件中**resultMap与resultType只能使用其中一个。**



##### 拓展ResultMaps

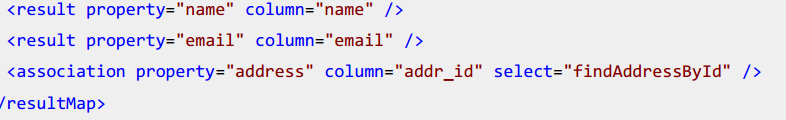
#### 一对一映射

##### 使用嵌套结果 ResultMap 实现一对一关系映射



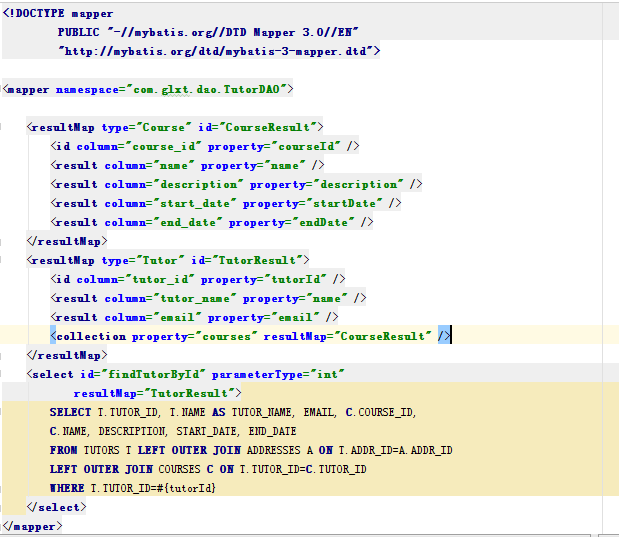
##### 使用嵌套查询实现一对一关系映射





#### 一对多映射

##### 使用内嵌结果 ResultMap 实现一对多映射



##### 使用嵌套 Select 语句实现一对多映射

