起源

iBATIS一词来源于“internet”和“abatis”的组合，是一个由Clinton Begin在2001年发起的开放源代码项目。最初侧重于密码软件的开发，现在是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO），同时还提供一个利用这个框架开发的JPetStore实例。

相对Hibernate和Apache OJB等“一站式”ORM解决方案而言，ibatis 是一种“半

自动化”的ORM实现。

所谓“半自动”，可能理解上有点生涩。纵观目前主流的 ORM，无论 Hibernate 还是Apache OJB，都对数据库结构提供了较为完整的封装，提供了从 POJO 到数据库表的全套映射机制。程序员往往只需定义好了POJO 到数据库表的映射关系，即可通过 Hibernate或者 OJB 提供的方法完成持久层操作。程序员甚至不需要对 SQL 的熟练掌握，Hibernate/OJB 会根据制定的存储逻辑，自动生成对应的 SQL 并调用 JDBC 接口加以执行。

大多数情况下（特别是对新项目，新系统的开发而言），这样的机制无往不利，大有一统天下的势头。但是，在一些特定的环境下，这种一站式的解决方案却未必灵光。

在笔者的系统咨询工作过程中，常常遇到以下情况：

1．系统的部分或全部数据来自现有数据库，处于安全考虑，只对开发团队提供几条Select SQL（或存储过程）以获取所需数据，具体的表结构不予公开。

2．开发规范中要求，所有牵涉到业务逻辑部分的数据库操作，必须在数据库层由存储过程实现（就笔者工作所面向的金融行业而言，工商银行、中国银行、交通银行，都在开发规范中严格指定）

3．系统数据处理量巨大，性能要求极为苛刻，这往往意味着我们必须通过经过高度优化的SQL语句（或存储过程）才能达到系统性能设计指标。

面对这样的需求，再次举起 Hibernate 大刀，却发现刀锋不再锐利，甚至无法使用，奈何？恍惚之际，只好再摸出JDBC 准备拼死一搏……，说得未免有些凄凉，直接使用 JDBC进行数据库操作实际上也是不错的选择，只是拖沓的数据库访问代码，乏味的字段读取操作令人厌烦。

“半自动化”的ibatis，却刚好解决了这个问题。 这里的“半自动化”，是相对Hibernate等提供了全面的数据库封装机制的“全自动化”ORM 实现而言，“全自动”ORM 实现了 POJO 和数据库表之间的映射，以及 SQL 的自动生成和执行。而ibatis 的着力点，则在于POJO 与 SQL之间的映射关系。也就是说，ibatis

并不会为程序员在运行期自动生成 SQL 执行。具体的 SQL 需要程序员编写，然后通过映射配置文件，将SQL所需的参数，以及返回的结果字段映射到指定 POJO。

使用ibatis 提供的ORM机制，对业务逻辑实现人员而言，面对的是纯粹的 Java对象，这一层与通过 Hibernate 实现 ORM 而言基本一致，而对于具体的数据操作，Hibernate会自动生成SQL 语句，而ibatis 则要求开发者编写具体的 SQL 语句。相对Hibernate等“全自动”ORM机制而言，ibatis 以 SQL开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。作为“全自动”ORM实现的一种有益补充，ibatis 的出现显得别具意义。