二

22 基于 MyBatis 的衍生框架一览

在前面的课时中，我们深入分析了 MyBatis 的内核，了解了 MyBatis 处理一条 SQL 的完整流程，剖析了 MyBatis 中动态 SQL、结果集映射、缓存等核心功能的实现原理。在日常工作中，除了单纯使用 MyBatis 之外，还可能会涉及 MyBatis 的衍生框架，这一讲我们就来介绍一下工作中常用的 MyBatis 衍生框架。

MyBatis-Generator

虽然使用 MyBatis 编写 DAO 层已经非常方便，但是我们还是要编写 Mapper 接口和相应的 Mapper.xml 配置文件。为了进一步节省编码时间，我们**可以选择 MyBatis-Generator 工具自动生成 Mapper 接口和 Mapper.xml 配置文件**。

这里我们通过一个简单示例介绍一下 MyBatis-Generator 工具的基本功能。

MyBatis-Generator 目前最新的版本是 1.4.0 版本，首先我们需要下载这个最新的 zip 包，并进行解压，得到 mybatis-generator-core-1.4.0.jar 这个 jar 包。

由于我们本地使用的是 MySQL 数据库，所以需要准备一个 mysql-connector-java 的 jar 包，我们可以从本地的 Maven 仓库中获得，具体的目录是：.m2/repository/mysql/mysql-connector-java/，在这个目录中选择一个最新版本的 jar 包拷贝到 mybatis-generator-core-1.4.0.jar 同目录下。

接下来，我们需要编写一个 generatorConfig.xml 配置文件，其中会告诉 MyBatis-Generator 去连接哪个数据库、连接数据库的用户名和密码分别是什么、需要根据哪些表生成哪些配置文件和类，以及这些生成文件的存放位置。下面是一个 generatorConfig.xml 配置文件的完整示例：

```
<?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>
    <!-- 使用的数据库驱动jar包 -->
    <classPathEntry location="mysql-connector-java-8.0.22.jar"/>
    <!-- 指定数据库地址、数据库用户名和密码 -->
    <context id="DB2Tables" targetRuntime="MyBatis3">
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                connectionURL="jdbc:mysql://localhost:3306/test"
                userId="root"  password="xxx">
        </jdbcConnection>
        <javaTypeResolver>
            <property name="forceBigDecimals" value="false"/>
        </javaTypeResolver>
        <!-- 生成的Model类存放位置 -->
        <javaModelGenerator targetPackage="org.example" targetProject="src">
            <!-- 是否支持生成子package -->
            <property name="enableSubPackages" value="true"/>
            <!-- 对String进行操作时，会添加trim()方法进行处理 -->
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>
        <!-- 生成的Mapper.xml映射配置文件的存放位置-->
        <sqlMapGenerator targetPackage="org.example.mapper" targetProject="src">
            <property name="enableSubPackages" value="true"/>
        </sqlMapGenerator>
        <!-- 生成的Mapper接口的存放位置-->
        <javaClientGenerator type="XMLMAPPER" targetPackage="org.example.mapper"
                             targetProject="src">
            <property name="enableSubPackages" value="true"/>
        </javaClientGenerator>
```

```
        <!-- 数据库表与Model类之间的映射关系，根据t_customer表进行映射-->
        <table schema="test" tableName="t_customer" domainObjectName="Customer"
               enableCountByExample="false" enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false" selectByExampleQueryId="false">
        </table>
    </context>
</generatorConfiguration>
```

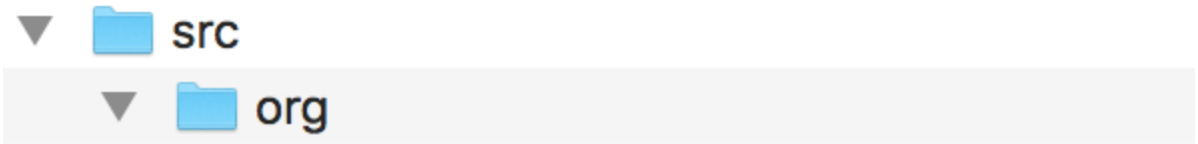
然后，我们准备一下数据库中的表，在 MySQL 中建立一个 test 数据库，并创建 t_customer 表，使用到的建库建表语句如下：

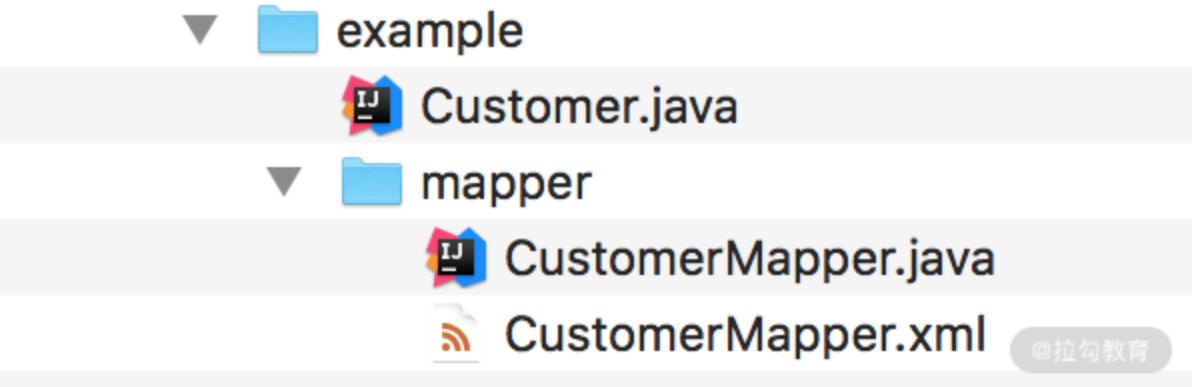
```
create databases test; # 创建数据库
use test;
DROP TABLE IF EXISTS `t_customer`; # 删除已有的t_customer表
CREATE TABLE `t_customer` ( # 创建t_customer表
  `id` int(255) NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `account` bigint(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

最后，我们在 mybatis-generator-core-1.4.0.jar 包同目录下新建一个 src 目录，存放生成的代码，然后执行如下命令，逆向生成需要的代码：

```
java -jar mybatis-generator-core-1.4.0.jar -configfile generatorConfig.xml
```

命令正常执行完成之后，可以看到 src 目录下生成的文件如下图所示：





MyBatis-Generator 工具类生成结果图

生成的 Customer.java 类是一个 Model 类（或者说 Domain 类），包含了 id、name、password、account 属性；CustomerMapper.xml 是 Customer 对应的 Mapper.xml 配置文件，其中定义了按照 id 进行查询和删除的 select、delete 语句，以及全字段写入和更新的 insert、update 语句；CustomerMapper 接口中包含了与 CustomerMapper.xml 对应的方法。该示例中生成的代码并不复杂，在你生成代码之后，也希望你能够自己分析一下。

MyBatis 分页插件

MyBatis 本身提供了 RowBounds 参数，可以实现分页的效果，但是在前面[第 14 讲]中我们提到过，通过 RowBounds 方式实现分页的时候，本质是将整个结果集数据加载到内存中，然后在内存中过滤出需要的数据，这其实也是我们常说的“内存分页”。而真正的分页是为了解决数据量太大，无法直接加载到内存或无法直接传输的问题，显然“内存分页”并没有解决这个问题。

你如果用过 MySQL 的话，应该知道我们常用 limit 方式进行分页，例如下面这条 select 语句：

```
select * from t_customer limit 5,10;
```

使用 Oracle 实现分页时，则需要用 rownum 实现，可见在不同数据库中实现物理分页的写法各不相同。

如果我们想屏蔽底层数据库的分页 SQL 语句的差异，同时使用 MyBatis 的 RowBounds 参数实现“物理分页”，可以考虑使用 MyBatis 的分页插件PageHelper。PageHelper 的使用比较简单，只需要在 pom.xml 中引入 PageHelper 依赖包，并在 mybatis-config.xml 配置文件中配置 PageInterceptor 插件即可，核心配置如下：

```
<plugins>
```

```
    <plugin interceptor="com.github.pagehelper.PageInterceptor">
        <property name="helperDialect" value="mysql"/>
        </plugin>
</plugins>
```

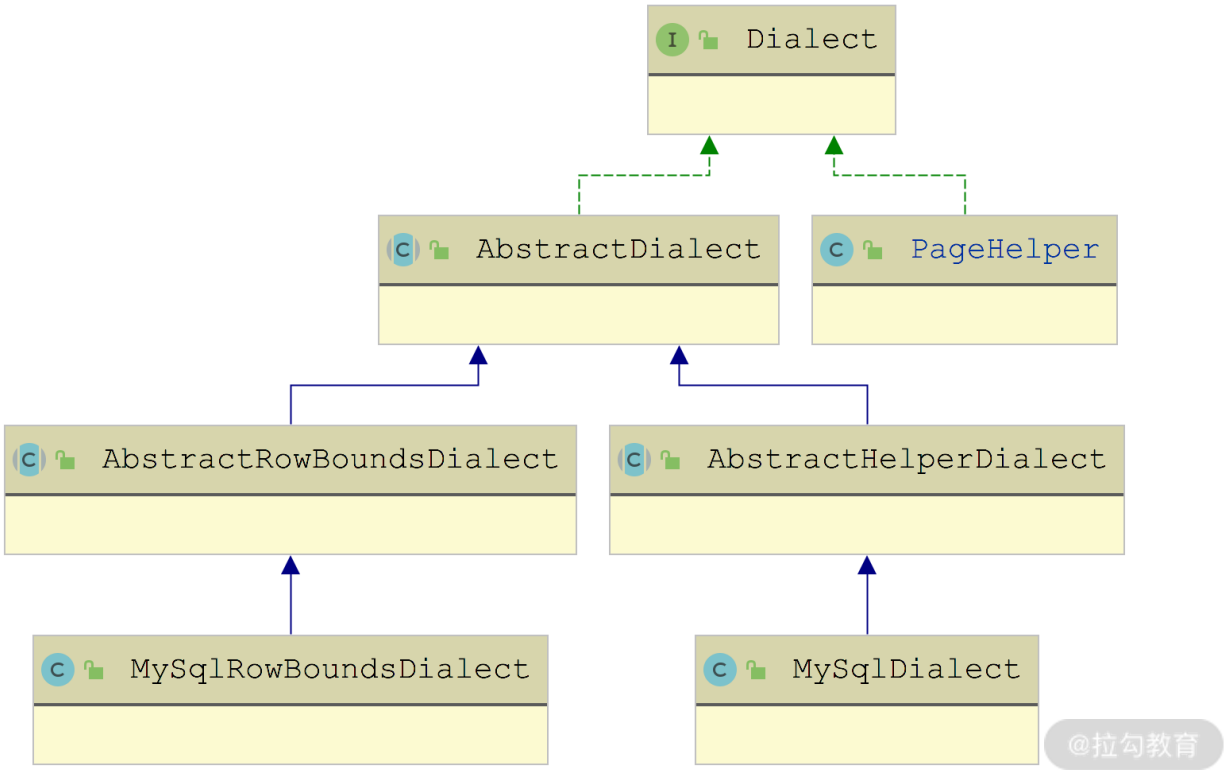
PageHelper 核心原理是使用 MyBatis 的插件机制，整个插件的入口是在 PageInterceptor。

在 PageInterceptor 初始化的时候，会根据配置的 helperDialect 属性以及 MyBatis 使用的 JDBC URL 信息确定底层连接的数据库类型，并创建一个 Dialect 对象。我们可以再来看 PageInterceptor 的注解信息，会发现 PageInterceptor 会拦截 Executor 中带有 RowBounds 参数的两个查询方法。拦截到目标方法之后，PageInterceptor.intercept() 方法会通过 Dialect 对象完成分页操作，核心代码如下：

```
List resultList;
// 判断是否需要进行分页
if (!dialect.skip(ms, parameter, rowBounds)) {
    // 是否需要查询总记录数，这可以帮助我们显示总页数
    if (dialect.beforeCount(ms, parameter, rowBounds)) {
        // 查询总记录数
        Long count = count(executor, ms, parameter, rowBounds, null, boundSql);
        // 处理查询总记录数，返回true时继续分页查询，false时直接返回，会返回false的原因很
        if (!dialect.afterCount(count, parameter, rowBounds)) {
            return dialect.afterPage(new ArrayList(), parameter, rowBounds);
        }
    }
    // 执行分页查询
    resultList = ExecutorUtil.pageQuery(dialect, executor,
            ms, parameter, rowBounds, resultHandler, boundSql, cacheKey);
} else {
    // 如果不需要，直接交给Executor执行查询，返回结果
```

```
        resultList = executor.query(ms, parameter, rowBounds, resultHandler, cacheKey,
    }
    // 在afterPage()方法中会完成总页数的计算等后置操作
    return dialect.afterPage(resultList, parameter, rowBounds);
```

通过对 PageInterceptor 的分析我们看到，**核心的分页逻辑都是在 Dialect 中完成的**，PageHelper 针对每个数据库都提供了一个 Dialect 接口实现。下图展示了 MySQL 数据库对应的 Dialect 接口实现：



MySqlDialect 的继承关系图

在上图中，PageHelper 是一个通用的 Dialect 实现，会将上述分页操作委托给当前线程绑定的 Dialect 实现进行处理，这主要是靠其中的 autoDialect 字段（PageAutoDialect 类型）实现的。AbstractDialect 中只提供了一个生成“查询总记录数”SQL 语句（即 select count(*) 语句）的功能。

AbstractRowBoundsDialect 这条继承线是针对 RowBounds 进行分页的 Dialect 实现，其中会根据 RowBounds 实现 Dialect 接口，例如，在 MySqlRowBoundsDialect 中的 getPageSql() 方法实现中会改写 SQL 语句，添加 limit 子句，其中的 offset、limit 参数均来自传入的 RowBounds 参数。

如果没有用 RowBounds 参数进行分页，而是在传入的 SQL 语句绑定实参（即 Executor.query() 方法的第二个参数 parameter）中指定 pageNum、pageSize 等分页信息，则会走 AbstractHelperDialect 这条继承线。在 PageObjectUtil 这个工具类中，会从绑定实参中解析出分页信息并封装成 Page 对象，然后传递给 AbstractHelperDialect 完成分页操作。例如，在 MySqlDialect 实现中的 getPageSql() 方法和 processPageParameter() 方法，都会从 Page 参数中获取分页信息，这两个方法的具体实现就留给你自己分析了。

到此为止，PageHelper 分页插件中的分页功能就介绍完了，除了基本的分页功能，PageHelper 还提供了分页使用的缓存等相关能力，这里就不再展开详细分析了，你若感兴趣的话可以下载其源码进行深入分析。

MyBatis-Plus

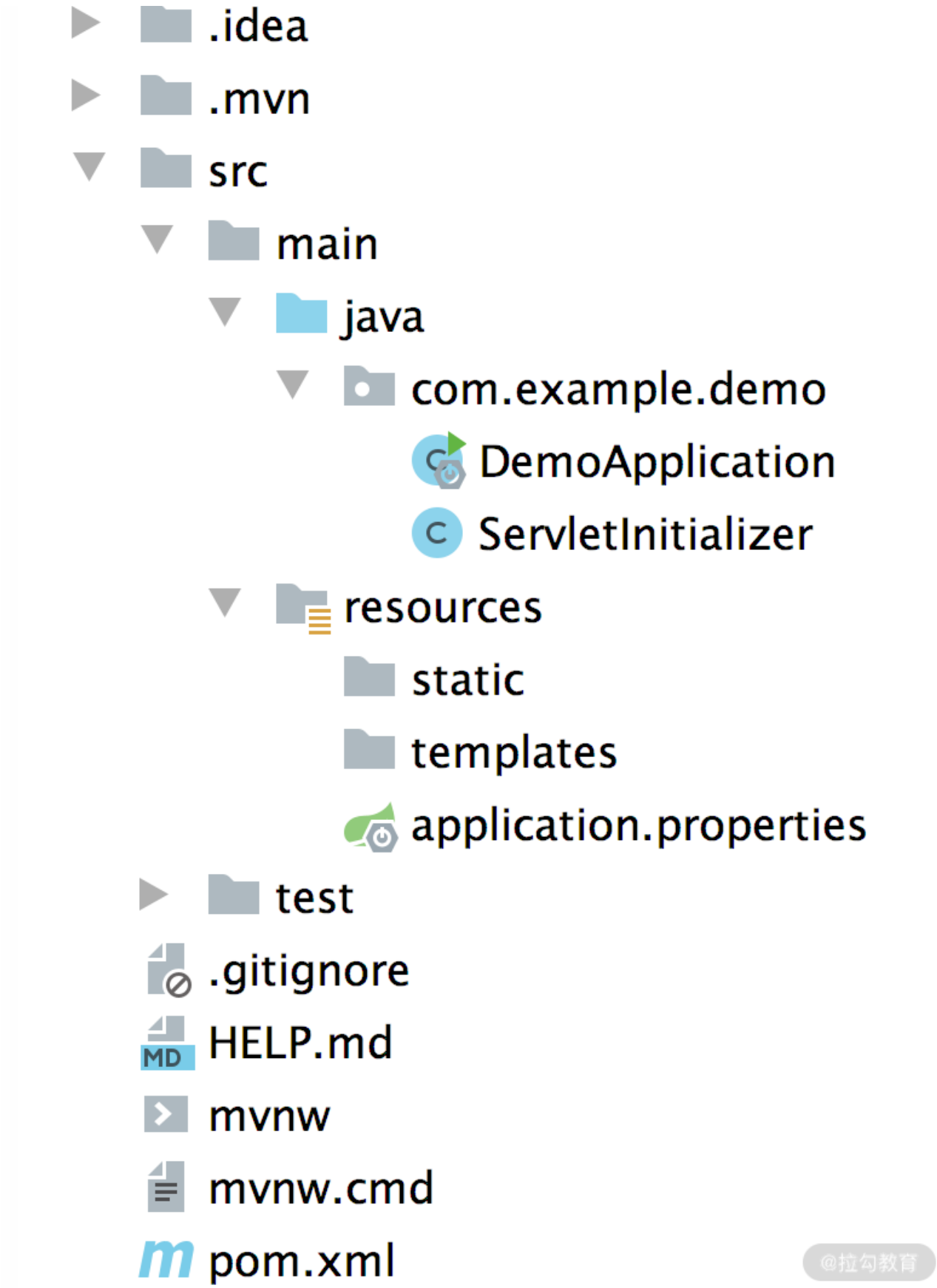
MyBatis-Plus 是国人开发的一款 MyBatis 增强工具，通过其名字就能看出，**它并没有改变 MyBatis 本身的功能，而是在 MyBatis 的基础上提供了很多增强功能，使我们的开发更加简洁高效**。也正是由于其“只做增强不做改变”的特性，让我们可以在使用 MyBatis 的项目中无感知地引入 MyBatis-Plus。

MyBatis-Plus 对 MyBatis 的很多方面进行了增强，例如：

- 内置了通用的 Mapper 和通用的 Service，只需要添加少量配置即可实现 DAO 层和 Service 层；
- 内置了一个分布式唯一 ID 生成器，可以提供分布式环境下的 ID 生成策略；
- 通过 Maven 插件可以集成生成代码能力，可以快速生成 Mapper、Service 以及 Controller 层的代码，同时支持模块引擎的生成；
- 内置了分页插件，可以实现和 PageHelper 类似的“物理分页”，而且分页插件支持多种数据库；
- 内置了一款性能分析插件，通过该插件我们可以获取一条 SQL 语句的执行时间，可以更快地帮助我们发现慢查询。

既然 MyBatis-Plus 在 MyBatis 之上提供了这么多的扩展，那么我们就来快速上手体验一下 MyBatis-Plus。这里我们依旧选用 MySQL 数据库，复用上面介绍 MyBatis-Generator 示例时用到的 test 库和 t_customer 表。

首先，新建一个 Spring Boot 项目，这里我们可以使用 Spring 官网提供的项目生成器快速生成，导入 IDEA 之后会发现 Spring Boot 的配置和启动类都已经生成好了，如下图所示：



Spring Boot 示例项目的结构图

接下来我们打开 pom.xml 文件，看到其中已经自动添加了 Spring Boot 的全部依赖，此时

只需要添加 mysql-connector-java 依赖以及 MyBatis-Plus 依赖即可（目前 MyBatis-Plus 最新版本是 3.4.2）：

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.2</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

再接下来，我们修改 application.properties 文件，添加数据库的相关配置：

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEnc
spring.datasource.username=root
spring.datasource.password=xxx
```

然后，我们开始编写 Customer 类和 CustomerMapper 接口，这两个类非常简单，Customer 类中需要定义 t_customer 表中各列对应的属性，如下所示：

```
@TableName(value = "t_customer") // 通过@TableName注解，指定Customer与 t_customer表的
public class Customer {
    private Integer id;
    private String name;
    private String password;
    private Long account;
    // 省略上述字段的getter/setter方法，以及toString()方法
}
```

CustomerMapper 接口的定义更加简单，只需要继承 BaseMapper 即可，具体定义如下：

```
public interface CustomerMapper extends BaseMapper<Customer> {
  // 无须提供任何方法定义，而是从BaseMapper继承
}
```

最后，我们修改一下这个 Spring Boot 项目的启动类 DemoApplication，在其中添加 @MapperScan 注解指定 Mapper 接口所在的包，该注解会自动进行扫描，DemoApplication 的具体实现如下：

```
@SpringBootApplication
@MapperScan("com.example.demo.mapper")
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

完成上述示例的编写之后，我们可以添加一个测试用例来查询 t_customer 表中的数据，具体实现如下：

```
@RunWith(SpringRunner.class)
@SpringBootTest
class DemoApplicationTests {
    @Autowired
    private CustomerMapper customerMapper;
    @Test
    public void testSelect() {
        Customer customer = new Customer();
        customer.setId(1);
        customer.setName("Bob");
        customer.setPassword("pwd");
```

```
        customer.setAccount(10097L);
        int insert = customerMapper.insert(customer);
        System.out.println("affect row num:" + insert);
        List<Customer> userList = customerMapper.selectList(null);
        userList.forEach(System.out::println);
    }
}
```

执行该单元测试之后，得到如下输出：

```
affect row num:1
Customer{id=1, name='Bob', password='pwd', account=10097}
```

MyBatis-Plus 的基础使用示例就介绍到这里了。另外，MyBatis-Plus官方文档中还提供了很多核心功能的说明和介绍，同时 MyBatis-Plus 还提供了示例 GitHub 仓库，其中包含了非常多的 MyBatis-Plus 示例代码和使用技巧，非常值得你参考。

总结

在这一讲我们重点介绍了 MyBatis 相关的辅助工具以及在 MyBatis 之上衍生出来的扩展框架。

- 首先，分析了 MyBatis-Generator 工具，它可以根据我们已有的数据表快速生成 MyBatis 中的 Domain 类、Mapper 接口以及 Mapper.xml 文件。
- 然后，介绍了 MyBatis 分页插件—— PageHelper，PageHelper 可以让我们直接使用 RowBounds API 实现“内存分页”，同时也可以帮助我们实现对不同数据库产品的分页功能。
- 最后，还讲解了 MyBatis-Plus 框架，MyBatis-Plus 内置了默认的 DAO 和 Service 实现以及分页功能，可以大幅度提高开发效率，你也可以结合我展示的示例来帮助你快速上手 MyBatis-Plus 框架。

上一页 下一页