大厂Java面试题：MyBatis是如何进行分页的？分页插件的实现原理是什么？
==========

大家好，我是[王有志](http://cxyroad.com/”https://www.yuque.com/wangyouzhi-u3woi/cr7d5y/uw8c5iyvpgnqpzmg”)。

今天给大家带来的是一道来自**京东的关于 MyBatis 实现分页功能的面试题**：MyBatis是如何进行分页的？分页插件的实现原理是什么？

通常，分页的方式可以分为两种：

* 逻辑（内存）分页
* 物理分页

逻辑（内存）分页指的是数据库返回全部符合条件的数据，然后再通过程序代码对数据结果进行分页处理；物理分页指的是通过 SQL 语句查询，由数据库返回分页后的查询结果。

逻辑（内存）分页和物理分页各有优缺点，物理分页需要频繁的访问数据库，对数据库的负担较重，逻辑（内存）分页在数据量较大时也会对应用程序的性能造成较大的影响。

MyBatis 中实现逻辑（内存）分页

在 MyBatis 中实现逻辑（内存）分页，需要借助 MyBatis 提供的 RowBounds 对象。我们举个例子，首先定义 Mapper 接口：

```
List<UserDO>  logicalPagination(RowBounds rowBounds);
```

接着我们来写 MyBatis 映射器中的 SQL 语句：

```
<select id="logicalPagination" resultType="com.wyz.entity.UserDO">
select * from user
</select>
```

可以看到，虽然我们在 Java 的接口中定义了入参 RowBounds，但是在 MyBatis 映射器中并没有使用它。

最后我们来写单元测试代码：

```
public void testLogicalPagination() {
  Reader mysqlReader = Resources.getResourceAsReader("mybatis-config.xml");
  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(mysqlReader);
  SqlSession sqlSession = sqlSessionFactory.openSession();
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

  RowBounds rowBounds = new RowBounds(0, 3);
  List<UserDO> users = userMapper.logicalPagination(rowBounds);
  log.info(JSON.toJSONString(users));
  sqlSession.close();
}
```

执行单元测试可以看到，虽然我们在 MyBatis 映射器中编写的 SQL 语句没有做任何限制，但实际上我们查询的结果只返回了 3 条数据，这就在 MyBatis 中实现逻辑（内存）分页的方式。

MyBatis 实现逻辑（内存）分页的原理

MyBatis 实现逻辑分页的原理并不复杂，简单来说，**在执行查询语句前先创建 ResultSetHandler 对象，并持有 RowBounds 参数，在查询结果返回后，使用 ResultSetHandler 对象处理查询结果时，进行逻辑分页**。

首先是构建 ResultSetHandler 对象的流程，在执行查询前，MyBatis 会创建 ResultSetHandler 对象，整体调用流程如下：

在执行查询后，MyBatis 会调用 ResultSetHandler 对象进行结果集的处理，其中就包含对逻辑分析的处理，部分源码如下：

```
private void handleRowValuesForSimpleResultMap(ResultSetWrapper rsw, ResultMap resultMap, ResultHandler<?> resultHandler, RowBounds rowBounds, ResultMapping parentMapping) throws SQLException {
  DefaultResultContext<Object> resultContext = new DefaultResultContext<>();
  ResultSet resultSet = rsw.getResultSet();
  skipRows(resultSet, rowBounds);
  while (shouldProcessMoreRows(resultContext, rowBounds) && !resultSet.isClosed() && resultSet.next()) {
    ResultMap discriminatedResultMap = resolveDiscriminatedResultMap(resultSet, resultMap, null);
    Object rowValue = getRowValue(rsw, discriminatedResultMap, null);
    storeObject(resultHandler, resultContext, rowValue, parentMapping, resultSet);
  }
}
```

首先来看第 4 行中调用的`DefaultResultSetHandler#skipRows`方法：

```
private void skipRows(ResultSet rs, RowBounds rowBounds) throws SQLException {
  if (rs.getType() != ResultSet.TYPE_FORWARD_ONLY) {
    if (rowBounds.getOffset() != RowBounds.NO_ROW_OFFSET) {
      rs.absolute(rowBounds.getOffset());
    }
  } else {
    for (int i = 0; i < rowBounds.getOffset(); i++) {
      if (!rs.next()) {
        break;
      }
    }
  }
}
```

```

因为没有设置 ResultSet 的类型，因此我们不必 if 语句中的内容，直接来看 else 语句中的内容，逻辑非常清晰，**根据传入的 RowBounds 对象的偏移量（即 offset）来移动 ResultSet 对象的游标位置**，来保证逻辑（内存）分页时数据的起始位置。

接着来看第 5 行中调用的 `DefaultResultSetHandler#shouldProcessMoreRows`方法：

```
private boolean shouldProcessMoreRows(ResultContext<?> context, RowBounds rowBounds) {
  return !context.isStopped() && context.getResultCount() < rowBounds.getLimit();
}
```

该方法也并不复杂，是用来控制查询数据总量的，当 ResultContext 对象中的数据量小于 RowBounds 中最大数据量时，才会进入 while 循环，以此来保证查询到的数据不会超出我们指定的范围。

而 ResultContext 对象 resultCount 字段的变化，是在 while 循环中调用 `DefaultResultSetHandler#storeObject`方法时改变的，这点就留给大家自行探索了。

**MyBatis 中实现物理分页**
---

在 MyBatis 中实现物理分页的常见方式有 3 种：

* 使用数据库提供的功能，如 MySQL 中的 limit，Oracle 中的 rownum；
* 通过自定义插件（拦截器）实现分页功能；
* 使用 MyBatis 分页插件实现，如 PageHelper。

### **使用数据库的功能实现分页**
```

我们以 MySQL 数据库为例展示一个完整的分页功能。

首先定义分页对象 Page，并定义 3 个字段，源码如下：

```
public class Page {

  /**
   * 当前页码
   */
  private Integer currentPage;

  /**
   * 每页条数
   */
  private Integer pageSize;

  /**
   * 总条数
   */
  private Integer totalSize;
}
```

接着我们来写 Mapper 中的接口，此时要定义两个接口，第一个是通过 Page 对象查询数据的接口，第二个是查询全部数据数量的接口，源码如下：

```
List<UserDO> selectUsers(@Param("page")Page page);

Long selectUsersCount();
```

最后我们来写 MyBatis 的映射器：

```
<select id="selectUsers" parameterType="com.wyz.entity.Page"
resultType="com.wyz.entity.UserDO">
  select * from user
```

```
  <if test="page != null">
    <bind name="start" value="((page.currentPage) - 1) *
page.pageSize"/>
    limit #{start}, #{page.pageSize}
  </if>
</select>

<select id="selectUsersCount" resultType="long">
  select count(*) from user
</select>
```

这里需要注意，我们与前端约定的页码是从 1 开始的，因此在 MyBatis 映射器中处理页码时需要减 1，不过即便如此，你也需要做好参数校验。另外，这里我们添加查询数据量的接口方法selectUsersCount是为了将数据量提供给前端，用于前端展示使用。

通过自定义插件（拦截器）**实现分页**

上面的方式虽然能够实现分页的需求，但问题是如果每一个需要分页的查询都要添加这些内容的话，那么我们需要花费一些精力来维护这些 SQL 语句，那么有没有一劳永逸的方法?

还记得我们在 [MyBatis核心配置讲解（下）](http://cxyroad.com/"https://mp.weixin.qq.com/s/smCR2OjUvTfGC5h1FnkOeA")中提到的 MyBatis 插件吗？MyBatis 中为每个关键场景都提供了插件的执行时机：

* StatementHandler，SQL 语句处理器；
* ParameterHandler，参数处理器；
* Executor，MyBatis 执行器；
* ResultSetHandler，结果集处理器。

如果想要实现物理分页，我们可以选择在 StatementHandler 和 Executor 阶段让插件介入，通过修改原始 SQL 来实现物理分页的功能。

首先我们来修改selectUsers方法对应的 MyBatis 映射器中的 SQL 语句，我们删除与分页相关的片段，源码如下：

```
```

```
<select id="selectUsers" resultType="com.wyz.entity.UserDO">
  select * from user
</select>
```

注意，这里我们没有删除接口的中 Page 参数，因为后面我们还要用到。

接着我们来定义自己的分页插件，这里我选择在 `StatementHandler#prepare`的阶段，修改原始 SQL，使其具备分页的能力，源码如下：

```
@Intercepts(
  {
    @Signature(
      type = StatementHandler.class,
      method = "prepare",
      args = {Connection.class, Integer.class}
    )
  })
public class MyPageInterceptor implements Interceptor {

  @Override
  @SuppressWarnings("unchecked")
  public Object intercept(Invocation invocation) throws Throwable {
    StatementHandler statementHandler = (StatementHandler)
invocation.getTarget();
    MetaObject metaObject =
SystemMetaObject.forObject(statementHandler);

    // 获取参数
    ParameterHandler parameterHandler = (ParameterHandler)
metaObject.getValue("delegate.parameterHandler");
    Map<String, Object> params = (Map<String, Object>)
parameterHandler.getParameterObject();
    Page page = (Page)params.get("page");

    // 获取原始SQL
    BoundSql boundSql = statementHandler.getBoundSql();
    String sql = boundSql.getSql();

    // 修改原始SQL
    sql = sql + " limit " + page.getCurrentPage() + "," +
page.getPageSize();
```

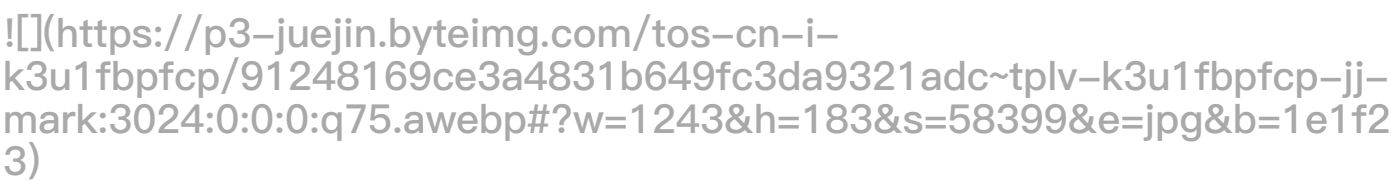
```
    metaObject.setValue("delegate.boundSql.sql", sql);
    return invocation.proceed();
  }

  // 省略部分方法
}
```

因为是在`StatementHandler#prepare`阶段让插件介入，此时 MyBatis 还没有生成 PreparedStatement 对象，此时我们只需要修改原始 SQL 语句即可。

接着我们在 mybatis-config.xml 配置我们自定义的插件：

```
<configuration>
  <plugins>
    <plugin interceptor="com.wyz.customize.plugin.MyPageInterceptor"/>
  </plugins>
</configuration>
```

最后执行单元测试，可以看到结果如我们预期的那样，实现了分页功能。



上面的自定义分页插件只实现了修改原始查询 SQL 语句的能力，依旧需要我们自行实现查询总数的接口，不过，我们也可以在插件中自动生成查询总数的方法。

Tips：上面的自定义插件只是为了展示，功能很不完善，健壮性也很差，不能在生产环境中使用。

使用分页插件来实现分页

MyBatis 中最常用的分页插件就是 PageHelper 了，它的用法非常简单。

首先是引入 PageHelper 插件：

```
<dependencies>
  <dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>6.1.0</version>
  </dependency>
</dependencies>
```

接着在 mybatis-config.xml 中配置 PageHelper 的插件：

```
<plugins>
  <plugin interceptor="com.github.pagehelper.PageInterceptor"/>
</plugins>
```

最后，我们来使用 PageHelper 来写一个单元测试：

```
public void testPageHelper() {
  PageHelper.startPage(0, 3);

  List<UserDO> users = userMapper.selectAll();

  PageInfo<UserDO> pageInfo = new PageInfo<>(users);
  long count = pageInfo.getTotal();

  sqlSession.close();
}
```

与我们自定义的分页插件不同的是，PageHelper 并不需要我们传入分页参数，而是通过`PageHelper#startPage`设置分页相关参数即可。PageHelper 是

通过 ThreadLocal 变量来保证同一个线程中的 PageInterceptor 能够获取到分页参数的。

核心原理上，PageHelper 与我们自定义实现的分页插件并没有太大差别，都是通过为 SQL 语句添加“limit”来实现的分页功能，只不过 PageHelper 选择的处理阶段为`Executor#query`，PageInterceptor 类的声明如下：

```
@Intercepts(
  {
    @Signature(type = Executor.class, method = ”query”, args = {MappedStatement.class, Object.class, RowBounds.class, ResultHandler.class}),
    @Signature(type = Executor.class, method = ”query”, args = {MappedStatement.class, Object.class, RowBounds.class, ResultHandler.class, CacheKey.class, BoundSql.class}),
  }
)
public class PageInterceptor implements Interceptor
```

因为 PageHelper 的整体逻辑并不复杂，核心原理也与我们之前自定义实现的分页插件相同，所以 PageHelper 的源码就留给大家自行分析了。

Tips：当然了， PageHelper 的功能更加完成，代码健壮性更好。

好了，今天的内容就到这里了，如果本文对你有帮助的话，希望多多点赞支持，如果文章中出现任何错误，还请批评指正。**最后欢迎大家分享硬核 Java 技术的金融摸鱼侠**[王有志](http://cxyroad.com/”https://www.yuque.com/wangyouzhi-u3woi/wvkm9u/uw8c5iyvpgnqpzmg?singleDoc”)，我们下次再见！

原文链接: https://juejin.cn/post/7365741765729632306