```
ds();
        System.out.println(users);
    }

```

注：这个分页方法和BaseMapper提供的分页一样都是内存分页，并非物理分页，因为sql语句中没用limit，和BaseMapper的selectPage方法一样，配置了分页插件后就可以实现真正的物理分页。AR的分页方法与BaseMapper提供的分页方法不同的是，BaseMapper的selectPage方法返回值是查询到的记录的list集合，而AR的selectPage方法返回的是page对象，该page对象封装了查询到的信息，可以通过getRecords方法获取信息。

二、插件的配置：

MP提供了很多好用的插件，而且配置简单，使用方便。接下来一起看看MP的插件如何使用。

1、分页插件：

之前就有说到，BaseMapper的selectPage方法和AR提供的selectPage方法都不是物理分页，需要配置分页插件后才是物理分页，那么现在就来看看如何配置这个插件。

```
<!-- 3、配置mybatisplus的sqlSessionFactory -->
    <bean id="sqlSessionFactory" class=
"com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean">
   <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:mybatis-
config.xml"/>
        <property name="typeAliasesPackage"
value="com.ty.mybatisplus.entity"/>
        <!-- 注入全局配置 -->
        <property name="globalConfig" ref="globalConfiguration"/>
        <!-- 配置插件 -->
        <property name="plugins">
            <list>
```

```
            <!-- 分页插件 -->
            <bean
class="com.baomidou.mybatisplus.plugins.PaginationInterceptor"/>
        </list>
    </property>
</bean>
```

注：在sqlSessionFactory这个bean中，通过配置插件，接下来的所有插件都配置在这个list中。

```
@Test
    public void testPage() {
        //配置了分页插件后，还是和以前一样的使用selectpage方法，
        //但是现在就是真正的物理分页了，sql语句中有limit了
        Page<Employee> page = new Page<>(1, 2);
        List<Employee> employeeList =
                emplopyeeDao.selectPage(page, null);
        System.out.println(employeeList);
        System.out.println("================= 相关的分页信息
==================");
        System.out.println("总条数:" + page.getTotal());
        System.out.println("当前页码:" + page.getCurrent());
        System.out.println("总页数:" + page.getPages());
        System.out.println("每页显示条数:" + page.getSize());
        System.out.println("是否有上一页:" + page.hasPrevious());
        System.out.println("是否有下一页:" + page.hasNext());
        //还可以将查询到的结果set进page对象中
        page.setRecords(employeeList);
    }
```


由图可知，sql语句中已经有了limit，是物理分页了。

也可以通过page调用相关方法获取到相关的分页信息，而且还可以把查询到的结果set回page对象中，方便前端使用。

2、性能分析插件：

在plugin的list中添加如下bean即可开启性能分析插件：

```
 <!-- 输出每条SQL语句及其执行时间，生产环境不建议使用该插件 -->
 <bean
class="com.baomidou.mybatisplus.plugins.PerformanceInterceptor">
        <property name="format" value="true"/><!-- 格式化SQL语句 -->
        <property name="maxTime" value="1000"/><!-- sql执行时间超过
value值就会停止执行，
                                                单位是毫秒 -->
 </bean>
```

注：这个性能分析插件配置了两个属性，第一个是格式化sql语句，设置为true后，sql语句格式就像上面的截图中的一样；第二个属性是sql语句执行的最大时间，超过value值就会报错，这里表示超过1000毫秒就会停止执行sql语句。

3、执行分析插件：

```
<!-- 如果是对全表的删除或更新操作，就会终止该操作 -->
<bean class="com.baomidou.mybatisplus.plugins.SqlExplainInterceptor">
       <property name="stopProceed" value="true"/>
</bean>
```

注：这个插件配置了一个属性，stopProceed设置为true后，如果执行的是删除表中全部内容，那就会抛出异常，终止该操作。该插件主要是防止手抖误删数据。

```
@Test
public void testSqlEx 标记。
```

3、mapper:

```
public interface UserDao extends BaseMapper<User> {
}
```

4、配置逻辑删除：

需要在spring-dao.xml中做如下配置：
首先定义逻辑删除的bean：

```
<!-- 逻辑删除 -->
<bean class="com.baomidou.mybatisplus.mapper.LogicSqlInjector"
id="logicSqlInjector"/>
```

再在全局配置的bean中注入逻辑删除以及逻辑删除值：

```
<!-- 5、mybatisplus的全局策略配置 -->
    <bean id="globalConfiguration"
class="com.baomidou.mybatisplus.entity.GlobalConfiguration">
        <!-- 此处省略其他全局配置 -->
        <!-- 注入自定义全局操作，做逻辑删除时需要先注释掉 -->
        <!--<property name="sqlInjector" ref="mySqlInjector"/>-->
        <!-- 注入逻辑删除，先要把自定义的注释掉 -->
        <property name="sqlInjector" ref="logicSqlInjector"/>
        <!-- 注入逻辑删除值 -->
        <property name="logicDeleteValue" value="-1"/><!-- -1是删除状
态 -->
        <property name="logicNotDeleteValue" value="1"/><!-- 1是未删除
状态 -->
    </bean>
```

注：因为逻辑删除实际上也是一个sqlInjector，所以先要把刚才做自定义全局操作时注入的自定义全局操作注释掉，上面代码中已有详细注释说明。

5、测试：

```
@Test
    public void testLogicDelete(){
        Integer result = userDao.deleteById(1);
        System.out.println(result);
        //User user = userDao.selectById(1);
        //System.out.println(user);
    }
```

注：运行该测试，执行删除操作的时候，真正执行的sql语句是UPDATE tb_user SET logic_flag=-1 WHERE id=?，就是把逻辑删除字段的值设置为-1；当逻辑删除字段的值是-1时再执行查询操作，sql是SELECT ... FROM tb_user WHERE id=? AND logic_flag=1，所以查询结果是null。

五、公共字段自动填充：

我们知道，当我们进行插入或者更新操作时，没有设置值的属性，那么在数据表中要么是为null，要么是保留原来的值。有的时候我们我们没有赋值但是却不想让其为空，比如name属性，我们插入时会默认赋上“林志玲”，更新时会默认赋值上“朱茵”，那么就可以用公共字段自动填充。

1、使用@TableField注解标记填充字段

```
@TableField(fill = FieldFill.INSERT_UPDATE)//插入和更新时填充
 private String name;
```

2、编写公共字段填充处理器类：

```
public class MyMetaObjectHandler extends MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        Object fieldValue = getFieldValByName(”name”,metaObject); //获取需要填充的字段
```

```
        if(fieldValue == null){   //如果该字段没有设置值
            setFieldValByName("name","林志玲",metaObject); //那就将其设置为"林志玲"
        }
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        Object fieldValue = getFieldValByName("name",metaObject);//获取需要填充的字段
        if(fieldValue == null){ //如果该字段没有设置值
            setFieldValByName("name","朱茵",metaObject);  //那就将其设置为"朱茵"
        }
    }
}
```

注：该类继承了MetaObjectHandler类，重写了insertFill和updateFill方法，在这两个方法获取需要填充的字段以及默认填充的值。

3、在spring-dao.xml中配置：

```
<!-- 公共字段填充处理器 -->
<bean class="com.ty.mybatisplus.handler.MyMetaObjectHandler" id="myMetaObjectHandler"/>
<!-- 5、mybatisplus的全局策略配置 -->
    <bean id="globalConfiguration" class="com.baomidou.mybatisplus.entity.GlobalConfiguration">
        <!-- 此处省略其他配置 -->
        <!-- 注入公共字段填充处理器 -->
        <property name="metaObjectHandler" ref="myMetaObjectHandler"/>
    </bean>
```

注：和配置逻辑删除一样，都是先将自定义的类注册成bean，再在全局策略配置中引用这个bean即可。

4、测试：

```
@Test
public void testHandlerInsert() {
        User user = new User();
        user.setGender(1);
        user.setAge(22);
        user.setLogicFlag(1);
        userDao.insert(user);
}
```

```


注：可以看到，虽然我们并没有给name赋值，但是已经自动把“林志玲”传进去了。更新时也一样有效，此处就不将测试代码贴出来了。

原文链接: https://juejin.cn/post/7386461612700712970
```