



build passing maven-central v0.0.7.101 license Apache-2.0

Joya 是对Spring Data JPA 扩展, JPA本身功能已经很强大了, 但是复杂查询语句HQL通常都是大量字符串拼接, 不利于维护和阅读, 提供优雅、易读和强大的链式查询语句的Joya应运而生

🔗 项目特性

- 基于Hibernate NativeQuery 进行扩展,支持全字段查询和指定字段查询,支持多种风格灵活易用
- 兼容JPA, 可插拔式集成, 无需修改任何代码, 不影响JPA和Hibernate 原有功能和特性
- 作为 JPA 的扩展和增强, 兼容 Spring Data JPA 原有功能和各种特性
- 拥有使用原生SQL语句的极致体验
- SQL结果可返回指定对象实体,同样支持单个字段返回包装类和String类
- 可扩展性强,兼容其他ORM框架底层工作量小

🔗 使用前提

适用于 使用Java Spring Data JPA 和JDK 1.8 及以上的项目

🔗 项目集成

1.引入依赖

Maven Project

```
<dependency>
  <groupId>com.sondertara</groupId>
  <artifactId>joya</artifactId>
  <version>0.0.7.102</version>
</dependency>
```

Gradle Project

```
implementation 'com.sondertara:joya:0.0.7.102'
```

2.添加配置

以Spring boot 项目为例,注入Bean.

```
@Bean
public JoyaRepositoryFactoryBean joyaRepositoryFactoryBean(){
```

```
return new JoyaRepositoryFactoryBean();
}
```

通过 `JoyaRepositoryFactoryBean` 会注入 `JoyaRepository` 和 `JoyaSpringContext`(一个Spring全局类)

3.application.yml 配置 (可选的)

```
joya:
  # 是否打印sql日志
  sql-view: true
```

使用示例

Joya 主要提供 `NativeSqlQuery` 来处理查询语句,关于使用有如下特殊说明:

`select`,`from`和`where`语句中都可以使用原生sql字符串方式来拼接查询

Joya默认会为表生成别名, 按照表在`select`和`where`部分第一次出现的顺序(`select`), 别名依次为 `t0`,`t1`... 如果要添加自定义的sql, 请使用表别名

对于联表查询到实体, 避免字段冲突。如果有相同的`column`字段,默认会使用第一个表中的值映射到目标实体,也可以为字段指定别名来都映射到目标实体

针对`where`语句中的`subQuery`,如果`where`是AND查询则`subQuery`联接条件为OR,反之`where`联接为OR,`subQuery`则为AND

```
// SQL编写风格

NativeSqlQuery query = NativeSqlQuery.builder()
    .select()
    .from()
    .where()
    .groupBy()
    .having()
    .orderBy()
    .build();
```

JoyaRepository中方法摘要

```
/**
 * 查询list
 * @param sql 原生sql
 * @param clazz 目标实体
 * @param params 参数
 * @param <T> 泛型
 * @return list
 */
```

```
@SuppressWarnings("unchecked")
public<T> List<T> findListBySql(String sql,Class<T> clazz,Object...params);

/**
 * find to list
 * @param nativeSql native query
 * @param resultClass result class
 * @param <T> generic
 * @return list
 */
@SuppressWarnings("unchecked")
public<T> List<T> findListBySql(NativeSqlQuery nativeSql,Class<T> resultClass);

/**
 * 分页查询
 *
 * @param resultClass 查询结果接收class
 * @param queryParam 查询数据
 * @param targetClass 查询数据库表
 * @param <T> 泛型
 * @return 分页数据
 */
public<T> PageResult<T> queryPage(PageQueryParam queryParam,Class<T>
resultClass,Class<?>...targetClass)

/**
 * 分页查询
 *
 * @param resultClass 查询结果接收class
 * @param queryParam 查询数据
 * @param <T> 泛型
 * @param joinPart join语句
 * @return 分页数据
 */
public<T> PageResult<T> queryPage(PageQueryParam queryParam,Class<T>
resultClass,UnaryOperator<JoinCriterion> joinPart)

/**
 * 通过sql 分页查询
 *
 * @param nativeSql 查询语句
 * @param resultClass 结果映射class
 * @param pageNo page
 * @param pageSize pageSize
 * @param <T> result
 * @return result
 */
@SuppressWarnings("unchecked")
public<T> PageResult<T> queryPage(NativeSqlQuery nativeSql,Class<T>
resultClass,Integer pageNo,Integer pageSize)
```

1.单表查询

- 查询全部字段

```
public class Test {  
    /**  
     * 等同于select * from xxx , 默认表中的字段全部列出  
     */  
    @Test  
    public void testSelectAll() {  
        // SELECT t0.id,t0.user_name,t0.user_email,t0.user_phone,t0.age FROM user  
        AS t0 WHERE t0.user_name = ?1  
  
        //使用function函数和 entity对于的class类  
        NativeSqlQuery query = NativeSqlQuery.builder()  
            .select()  
            .from(UserPo.class)  
            .where(w -> w.eq(UserPo::getUserName, "张三"))  
            .build();  
  
        //使用字符串格式  
        NativeSqlQuery query = NativeSqlQuery.builder()  
            .select()  
            .from("user AS t0")  
            .where(w -> w  
                // .eq("t0.user_name", "张三")) 下划线格式  
                .eq("t0.userName", "张三")) //驼峰格式  
            .build();  
  
        // 映射到UserDTO  
        List<UserDTO> list = joyaRepository.findListBySql(query, UserDTO.class);  
    }  
}
```

- 指定字段

```
public class Test {  
    @Test  
    public void testSelectSome() {  
  
        // SELECT t0.id, t0.user_name FROM user AS t0 WHERE t0.user_name = ?1  
  
        // 使用function函数接口 支持选择1-12个字段  
        NativeSqlQuery query = NativeSqlQuery.builder()  
            .select(UserPo::getId, UserPo::getUserName)  
            .from(UserPo.class)
```

```

        .where(w -> w.eq(UserPo::getUserName, "张三"))
        .build();

//使用UnaryOperator 函数接口
NativeSqlQuery query1 = NativeSqlQuery.builder()
    .select((UnaryOperator<SelectCriterion>) s -> {
        s.add(UserPo::getUserName).add(UserPo::getId);
        return s;
    })
    .from(UserPo.class)
    .where(w -> w.eq(UserPo::getUserName, "张三"))
    .build();

//字符串格式指定字段
NativeSqlQuery query3 = NativeSqlQuery.builder()
    .select("t0.id", "t0.userName")
    //.select("t0.id", "t0.user_name")
    .from(UserPo.class)
    .where(w -> w.eq(UserPo::getUserName, "张三"))
    .build();

List<UserDTO> list = joyaRepository.findListBySql(query, UserDTO.class);

// 查询单个字段映射到包装类或String
//SELECT t0.user_name FROM user AS t0 WHERE t0.user_name = ?1
NativeSqlQuery query3 = NativeSqlQuery.builder()
    .select(UserPo::getUserName)
    .from(UserPo.class)
    .where(w -> w.eq(UserPo::getUserName, "张三"))
    .build();

List<String> list1 = joyaRepository.findListBySql(query3, String.class);
    }
}

```

2.联表查询

```

public class Test {
    @Test
    public void testJoin() {
        // 联接字段在where 语句中
        //SELECT
        t0.id,t0.user_name,t0.user_email,t0.user_phone,t0.age,t1.user_id,t1.account_expire
        d_time,t1.password_expired_time,t1.ext_data FROM user AS t0, user_extend AS t1
        WHERE t0.id = t1.user_id AND t0.user_name = ?1
        NativeSqlQuery query = NativeSqlQuery.builder()
            .select()
            .from(UserPo.class, UserExtendPo.class)
            .where(w -> w
                .eq(UserPo::getId, UserExtendPo::getUserId)

```

```

        .eq(UserPo::getUserName, "张三"))
        .build();

    // 联接字段在from语句中 支持inner join,left join和 right join三种方式,目前为了提升效率,限制最大支持三张表联表查询

    //SELECT
    t0.id,t0.user_name,t0.user_email,t0.user_phone,t0.age,t1.user_id,t1.account_expired_time,t1.password_expired_time,t1.ext_data FROM user AS t0 JOIN user_extend AS t1 ON t0.id = t1.user_id WHERE t0.user_name = ?1
    NativeSqlQuery query1 = NativeSqlQuery.builder()
        .select()
        .from(j -> j.join(UserPo::getId, UserExtendPo::getUserId))
        .where(w -> w
            .eq(UserPo::getUserName, "张三"))
        .build();

    //对于冲突字段可以指定别名,如user表和user_extend表同时有 updateTime字段,可以通过指定别名来避免字段值覆盖
    // SELECT
    t0.id,t0.user_name,t0.user_email,t0.user_phone,t0.update_time,t1.update_time AS modifyTime,t1.account_expired_time,t1.password_expired_time,t1.ext_data FROM user AS t0 JOIN user_extend AS t1 ON t0.id = t1.user_id WHERE t0.user_name = ?1
    NativeSqlQuery query3 = NativeSqlQuery.builder()
        .select()//查询全部字段
        .specificS("t1.updateTime AS modifyTime") //将user_extend中重名的updateTime 指定为modifyTime
        .from(j -> j.join(UserPo::getId, UserExtendPo::getUserId))
        .where(w -> w
            .eq(UserPo::getUserName, "张三"))
        .build();
    }
}

```

3.特殊定制化查询

在Joya中,where 查询语句支持常用查询,也可以通过specificw方法来添加特殊查询语句

```

.where(w -> w
    .eq() // =
    .gt() // >
    .lt() // <
    .gte() // >=
    .lte() // <=
    .isNull() // is null
    .isNotNull() // is not null
    .in() // in
    .notIn() // not in
    .endsWith() // like '%a'
    .contains() // like '%a%'

```

```
.startsWith() // like 'a%'
.specificS() //指定特殊的查询语句
```

where 查询语句默认是AND条件联接,可以选择OR条件联接

```
//SELECT
t0.id,t0.user_name,t0.user_email,t0.user_phone,t0.age,t1.user_id,t1.salt,t1.account_expired_time,t1.password_expired_time,t1.ext_data FROM user AS t0, user_extend AS t1 WHERE t0.user_name = ?1 OR t0.user_name = ?2 OR ( t0.age >= ?2 AND t0.user_phone LIKE ?2 )
NativeSqlQuery query=NativeSqlQuery.builder()
    .select()
    .from(UserPo.class,UserExtendPo.class)
    .where(w->w
        .eq(UserPo::getUserName,"张三")
        .eq(UserPo::getUserName,"李四")

        .subQuery(q-
>q.gte(UserPo::getAge,18).startsWith(UserPo::getUserPhone,"1385")),true)
    .build();
```

4.分页查询

在Joya中为分页查询封装了PageQueryParam 提供给restFul接口使用

```
/**
 * 分页查询参数支持常用封装
 */
public class PageQueryParam extends JoyaQuery implements Serializable {
    /**
     * 分页大小
     */
    private Integer pageSize = 10;
    /**
     * 页数 默认从0开始
     */
    private Integer page = 0;
    /**
     * 连接类型 默认and
     */
    private LinkType linkType = LinkType.AND;
    /**
     * 排序字段
     */
    private List<OrderParam> orderList = Lists.newArrayList();

    /**
     * 搜索参数
     */
}
```

```
private List<SearchParam> params = Lists.newArrayList();

/**
 * group by
 */
private String groupBy;

public enum LinkType {
    /**
     *
     */
    AND,
    OR;
}
}
```

如有如下查询场景

分页关联查询user表和user_extend表,查询条件为userName like '张三%'并且 age>18,按照user表中的id降序

那么接口请求的PageQueryParam则为

```
{
  "orderList": [
    {
      "fieldName": "t0.id",
      "orderType": "DESC"
    }
  ],
  "page": 0,
  "pageSize": 10,
  "params": [
    {
      "fieldName": "t0.userName",
      "fieldValue": "张三",
      "operator": "LIKE_R"
    },
    {
      "fieldName": "t0.age",
      "fieldValue": 18,
      "operator": "GT"
    }
  ]
}
```

使用Joya查询方法有如下两种:

- 分页查询(指定join字段)


```
PageQueryParam pageQueryParam=JSON.parseObject(jsonString,PageQueryParam.class);

PageResult<UserDTO>
pageResult=joyaRepository.queryPage(pageQueryParam,UserDTO.class,j-
>j.join(UserPo::getId,UserExtendPo::getUserId));
```

- 分页查询(指定where语句中的联接字段)

```
PageQueryParam pageQueryParam1=JSON.parseObject("",PageQueryParam.class);
//附加where
pageQueryParam1.setSpecificW("t0.id=t1.userId");

//指定要查询的表, 顺序要注意, 因为别名按照先后顺序来生成
PageResult<UserDTO>
pageResult=joyaRepository.queryPage(pageQueryParam,UserDTO.class,UserPo.class,User
ExtendPo.class);
```

参与贡献

fork本项目,添加features或bugfix,提交Pull Requests

开源许可证

Joya 遵守 [Apache License 2.0](#) 许可证。