Latke 持久层 - 新增 add 方法解读

作者：nobt
原文链接：https://ld246.com/article/1536366694473
来源网站：链滴

说明：这篇文章不探讨Latke框架的IOC/DI部分，Latke框架中的IOC/DI功能跟Spring是很相似的，起来会觉得很好上手，在这里只是说明为什么Latke可以将一个JSON用类似ORM的功能存储到关系数据库。

add源码追溯

以solo新增一篇博客的后台全过程为例

- 博客新增入口

```
@RequestProcessing(value = "/console/article/", method = HTTPRequestMethod.POST)
   public void addArticle(final HttpServletRequest request, final HttpServletResponse response
 final HTTPRequestContext context,
                        final JSONObject requestJSONObject) throws Exception {
      //省略代码
      final String articleId = articleMgmtService.addArticle(requestJSONObject);
      //省略代码
   }
```

- 上面接口调用的下一层方法

```
public String addArticleInternal(final JSONObject article) throws ServiceException {
        //省略代码，将前台传过来的json封装成最终新增所需要的JSON数据，调用新增方法
   articleRepository.add(article);
   //省略代码

   }
```

articleRepository是通过Latke框架的注解@Inject注入进来的，在它的实现类ArticleRepositoryImp中没有add(JSONObject jsonObject)类型的方法，那么它调的肯定是其父类的add(JSONObject jso Object)方法

● 看看ArticleRepository的实现类ArticleRepositoryImpl的结构

```
public class ArticleRepositoryImpl extends AbstractRepository implements ArticleRepository {
    public ArticleRepositoryImpl() {
        super(Article.ARTICLE);
    }
}
```

● 这个无参的构造方法，发现它没干什么事，只是调用了父类的无参构造方法，注意Article.ARTICLE面会用到

```
public AbstractRepository(final String name) {
        try {
            Class<Repository> repositoryClass;

            final Latkes.RuntimeDatabase runtimeDatabase = Latkes.getRuntimeDatabase();
            switch (runtimeDatabase) {
                case MYSQL:
                case H2:
                case MSSQL:
                case ORACLE:
                    repositoryClass = (Class<Repository>) Class.forName("org.b3log.latke.repository.
dbc.JdbcRepository");

                    break;
                case NONE:
                    repositoryClass = (Class<Repository>) Class.forName("org.b3log.latke.repository
NoneRepository");

                    break;
                default:
                    throw new RuntimeException("The runtime database [" + runtimeDatabase + "] is
not support NOW!");
            }

            final Constructor<Repository> constructor = repositoryClass.getConstructor(String.cla
s);

            repository = constructor.newInstance(name);
        } catch (final Exception e) {
            throw new RuntimeException("Can not initialize repository!", e);
        }

        Repositories.addRepository(repository);
        LOGGER.log(Level.INFO, "Constructed repository [name={0}]", name);
    }
```

Latkes.getRuntimeDatabase()是从配置文件local.properties中获取变量名为runtimeDatabase（行数据库类型）的值，从代码中可以看到，支持三种数据库：MYSQL、H2、ORACLE，根据runtime atabase的值通过反射实例化一个JdbcRepository对象，这个对象就是比较底层的一个持久方法了。

● 接着再看看ArticleRepositoryImpl的父类AbstractRepository中的add(JSONObject jsonObject)法

```
@Override
public String add(final JSONObject jsonObject) throws RepositoryException {
        if (!isWritable() && !isInternalCall()) {
            throw new RepositoryException("The repository [name=" + getName() + "] is not writ
ble at present");
        }

        Repositories.check(getName(), jsonObject, Keys.OBJECT_ID);

        return repository.add(jsonObject);
    }
```

前面说了repository对象是根据runtimeDatabase的值通过反射实例化一个JdbcRepository对象，
么调用的就是JdbcRepository类中的add方法

```
@Override
    public String add(final JSONObject jsonObject) throws RepositoryException {
        final JdbcTransaction currentTransaction = TX.get();
        if (null == currentTransaction) {
            throw new RepositoryException("Invoking add() outside a transaction");
        }

        final Connection connection = getConnection();
        final List<Object> paramList = new ArrayList<>();
        final StringBuilder sql = new StringBuilder();
        String ret;

        try {
            if (Latkes.RuntimeDatabase.ORACLE == Latkes.getRuntimeDatabase()) {
                toOracleClobEmpty(jsonObject);
            }
            ret = buildAddSql(jsonObject, paramList, sql);
            JdbcUtil.executeSql(sql.toString(), paramList, connection, false);
            JdbcUtil.fromOracleClobEmpty(jsonObject);
        } catch (final Exception e) {
            LOGGER.log(Level.ERROR, "Add failed", e);

            throw new RepositoryException(e);
        }

        return ret;
    }
```

从方法的命名应该可以很轻易的判断出是什么意思了，就是第一步构造SQL语句，第二步执行SQL语句

```
private String buildAddSql(final JSONObject jsonObject, final List<Object> paramlist, final Str
ngBuilder sql) throws Exception {
        String ret = null;
        if (!jsonObject.has(Keys.OBJECT_ID)) {
            if (!(KEY_GEN instanceof DBKeyGenerator)) {
                ret = (String) KEY_GEN.gen();
                jsonObject.put(Keys.OBJECT_ID, ret);
            }
        } else {
```

```
            ret = jsonObject.getString(Keys.OBJECT_ID);
        }
        setProperties(jsonObject, paramlist, sql);
        return ret;
    }
```

判断传进来的JSON中设置的默认主键是否为空，如果为空就设置一个，如果不为空就取出来用来返，solo项目中的主键名称都是oId,值是时间戳，

调用setProperties(jsonObject, paramlist, sql);方法。

```
private void setProperties(final JSONObject jsonObject, final List<Object> paramlist, final StrigBuilder sql) throws Exception {
        final Iterator<String> keys = jsonObject.keys();

        final StringBuilder insertString = new StringBuilder();
        final StringBuilder wildcardString = new StringBuilder();

        boolean isFirst = true;
        String key;
        Object value;

        while (keys.hasNext()) {
            key = keys.next();

            if (isFirst) {
                insertString.append("(").append(key);
                wildcardString.append("(?");
                isFirst = false;
            } else {
                insertString.append(",").append(key);
                wildcardString.append(",?");
            }

            value = jsonObject.get(key);
            paramlist.add(value);

            if (!keys.hasNext()) {
                insertString.append(")");
                wildcardString.append(")");
            }
        }

        sql.append("insert into ").append(getName()).append(insertString).append(" values ").ap
end(wildcardString);
    }
```

这段代码的意思就是在根据传进来的JSON拼接SQL语句，最后一句中有一个getName()语句，其实接在这里就是对应数据库的表名称，什么时候传进来的，其实是在上面的ArticleRepositoryImpl类中用父类的无参构造方法时传进来的，在父类中通过反射实例化JdbcRepository对象需要传入name属，在类JdbcRepository中会经常用到，可以说只要跟数据库相关的操作可能都要用到表名称。

拼接好SQL语句后，就是执行SQL了，至此一个新增操作的全部就解读完毕了。

总结

通过看源码，就不难明白，为什么他说不需要像Hibernate、MyBatis那类ORM框架非得从JSON转实体类对象，再跟关系型数据库对应了，为什么能前端拼接的JSON能直接调用add方法就能保存到据库了。