# ScalikeJDBC 基于 SQL 的简介 DB 访问

1. **什么是 ScalikeJDBC**

ScalikeJDBC 是一款给 Scala 开发者使用的简洁 DB 访问类库，它是基于 SQL 的，使用者只需要关注 SQL 逻辑的编写，所有的数据库操作都交给 ScalikeJDBC。这个类库内置包含了JDBC API，并且给用户提供了简单易用并且非常灵活的 API。并且，QueryDSL(通用查询查询框架)使你的代码类型安全的并且可重复使用。我们可以在生产环境大胆地使用这款 DB 访问类库。

# 项目中使用 ScalikeJDBC

## 添加依赖

<!-- scalikejdbc\_2.11 -->

<dependency>

<groupId>org.scalikejdbc</groupId>

<artifactId>scalikejdbc\_2.11</artifactId>

<version>2.5.0</version>

</dependency>

<!-- scalikejdbc-config\_2.11 -->

<dependency>

<groupId>org.scalikejdbc</groupId>

<artifactId>scalikejdbc-config\_2.11</artifactId>

<version>2.5.0</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.38</version>

</dependency>

* 1. **数据库CURD**
     1. **配置数据库信息**

# MySQL example db.default.driver="com.mysql.jdbc.Driver"

db.default.url="jdbc:mysql://localhost/bbs?characterEncoding= utf-8"

db.default.user="root"

db.default.password="123456"

或者

# db.sheep.driver="com.mysql.jdbc.Driver" # db.sheep.url="jdbc:mysql://localhost/bbs" # db.sheep.user="root"

# db.sheep.password="123456"

更多配置请参考官网.

## 加载数据配置信息

// 加载数据库配置信息

// 默认加载 db.default.\* DBs.setup()

// 加载 db.sheep 配置信息

DBs.setup('sheep)

* + 1. **查询数据库并封装数据**

// 加载数据库配置信息

// 默认加载 db.default.\* DBs.setup()

// 查询数据并返回单个列, 并将列数据封装到集合中

**val** list: List[String] = DB readOnly { **implicit** session => sql"select content from post".map(rs =>

rs.string("content")).list().apply()

}

**for** (s <- list ) { println(s)

}

// 用户实体

**case class** Users(id: String, name: String, nickName: String)

*/\*\**

*\* 查询数据库,并将数据封装成一个对象并返回一个集合*

*\*/*

// 初始化数据库链接

DBs.setup('sheep)

**val** userses: List[Users] = NamedDB('sheep) readOnly

{ **implicit** session =>

sql"SELECT \* from users".map(rs => Users(rs.string("id"), rs.string("name"), rs.string("nickname"))).list().apply()

}

**for** (usr <- userses ) { println(usr.nickName)

}

* + 1. **插入数据**
       1. **AutoCommit**

*/\*\**

*\* 插入数据, 使用autoCommit*

*\*/*

**val** insertResult: Int = DB.autoCommit { **implicit** session => SQL("insert into users(name, nickname)

values(?,?)").bind("test01", "test01").update().apply()

}

println(insertResult)

* + - 1. **插入返回主键标识**

*/\*\**

*\* 插入数据, 并返回主键*

*\*/*

**val** id: Long = DB.localTx(**implicit** session => { sql"INSERT INTO users(name, nickname, sex) VALUES

(?,?,?)".bind("测试", "000",

1).updateAndReturnGeneratedKey("nickname").apply()

})

println(id)

* + - 1. **事务插入**

*/\*\**

*\* 使用事务插入数据库*

*\*/*

**val** tx: Int = DB.localTx { **implicit** session => sql"INSERT INTO users(name, nickname, sex) VALUES

(?,?,?)".bind("犊子", "000", 1).update().apply()

// var s = 1 / 0

SQL("INSERT INTO users(name, nickname, sex)

values(?,?,?)").bind("王八犊子", "xxx", 0).update().apply()

}

println(s"tx = **$**{tx}")

**2.2.5. 更新数据**

*/\*\**

*\* 更新数据*

*\*/*

DB.localTx{ **implicit** session => sql"UPDATE users SET pwd

= ?".bind("88999").update().apply()

}