

Mybatis-plus

- `SqlSessionFactoryBuilder` 获取并解析 `xml` 配置，传递解析后的 `xml` 配置文件给 `DefaultSqlSessionFactory`，通过 `new DefaultSqlSessionFactory(config)` 创建 `SqlSessionFactory`，一旦创建了 `SqlSessionFactory`，就不再需要它了；
- `SqlSessionFactory` 是核心：每个基于 `MyBatis` 的应用都是以一个 `SqlSessionFactory` 的实例为中心的
- `Properties` 属性优先级：`url/resource` 关联的 `properties` 文件属性 > `property` 标签属性
- 如果设置别名时不指定具体别名，mybatis会使用 `class.getSimpleName()` 作为别名（`alias`）
- `Mybatis` 已经默认对许多类使用了别名（如：`arrayList`、`list`、`collection`、`hashMap`、`map`、`_byte[]` 等），具体查看 `TypeAliasRegistry` 构造方法；
- `objectFactory`：`MyBatis` 每次创建结果对象的新实例时，它都会使用一个对象工厂（`ObjectFactory`）实例来完成；
- 所有的配置信息解析之后，保存在 `BaseBuilder.configuration` 中；
- `environments` 主要是配置不同环境下的数据源；
- `databaseIdProvider` 支持多种数据库版本；
- `Mybatis` 默认提供了许多 `typeHandler`，具体请查看 `org.apache.ibatis.type.TypeHandlerRegistry` 类构造方法；
- `SqlSessionFactory` 的 `getConfiguration` 方法可以即时查看 `mybatis` 配置信息；
- 如果 `sqlSessionFactory.openSession()` 的时候没有设置 `autoCommit`，那么事物是不会自动提交的，需要调用 `sqlSession.commit()`；
- `Mybatis` 自定义插件2步走战略：

MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用。默认情况下，MyBatis 允许使用插件来拦截的方法调用包括：

- **Executor** (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
- **ParameterHandler** (getParameterObject, setParameters)
- **ResultSetHandler** (handleResultSets, handleOutputParameters)
- **StatementHandler** (prepare, parameterize, batch, update, query)

这些类中方法的细节可以通过查看每个方法的签名来发现，或者直接查看 `MyBatis` 的发行包中的源代码。假设你想做的不仅仅是监控方法的调用，那么你应该很好的了解正在重写的方法的行为。因为如果在试图修改或重写已有方法的行为的时候，你很可能在破坏 `MyBatis` 的核心模块。这些都是更低层的类和方法，所以使用插件的时候要特别当心。

第一，通过对 `MyBatis` `org.apache.ibatis.executor.statement.StatementHandler` 中的 `prepare` 方法进行拦截即可：

```
package com.bytebeats.mybatis3.interceptor;

import org.apache.ibatis.executor.statement.StatementHandler;
import org.apache.ibatis.mapping.BoundSql;
import org.apache.ibatis.plugin.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.Connection;
import java.util.Properties;

/**
 * ${DESCRIPTION}
 *
 * @author Ricky Fung
 * @date 2017-02-17 11:52
 */
@Intercepts({ @Signature(type = StatementHandler.class, method = "prepare",
```

```

        args = { Connection.class, Integer.class}) })
public class SQLStatsInterceptor implements Interceptor {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    @Override
    public Object intercept(Invocation invocation) throws Throwable {

        StatementHandler statementHandler
            = (StatementHandler) invocation.getTarget();
        BoundSql boundSql = statementHandler.getBoundSql();
        String sql = boundSql.getSql();
        logger.info("mybatis intercept sql:{}", sql);
        return invocation.proceed();
    }

    @Override
    public Object plugin(Object target) {
        return Plugin.wrap(target, this);
    }

    @Override
    public void setProperties(Properties properties) {
        String dialect = properties.getProperty("dialect");
        logger.info("mybatis intercept dialect:{}", dialect);
    }
}

```

第二，配置：

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <plugins>
        <plugin interceptor="com.bytebeats.mybatis3
            .interceptor.SQLStatsInterceptor">
            <property name="dialect" value="mysql" />
        </plugin>
    </plugins>
</configuration>

```

- `com.baomidou.mybatisplus.plugins.PaginationInterceptor` : `mybatis-plus` 分页插件;
- `Mybatis-plus` 中，哈哈，也许你根本用不到：

JSON 序列化移除 transient 修饰的 Page 无关紧要的返回属性

```

// Jackson 方式
objectMapper.configure(MapperFeature.PROPAGATE_TRANSIENT_MARKER, true);

```

架构

[博客](#) 待总结。。。

[系列博客](#)