

- `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`，那么事物是不是会自动提交的，需要调用 `sqlSession.commit()`；

- `Mybatis` 自定义插件2步走战略：

```

+ > Executor (update, query, flushStatements, commit, rollback,
+   getTransaction, close, isClosed)
+ > ParameterHandler (getParameterObject, setParameters)
+ > ResultHandler (handleResultSets, handleOutputParameters)
+ > StatementHandler (prepare, parameterize, batch, update, query)

```

`Mybatis` 允许我们自定义插件对数据库操作中间环节进行拦截，在拦截中构造一些返回的结果集，即：自定义插件，`Mybatis` 允许我们自定义插件拦截数据库操作



这里是一个插件的接口，这里主要是在 `mybatis-config.xml` 文件中配置 `plugins` 节点，拦截数据库操作：



- `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

```

// 序列化 Page
// PageInterceptor: org.apache.ibatis.plugins.PluginImpl, org.apache.ibatis.plugins.PluginImpl, org.apache.ibatis.plugins.PluginImpl

```

## 架构

博客 待总结。。。