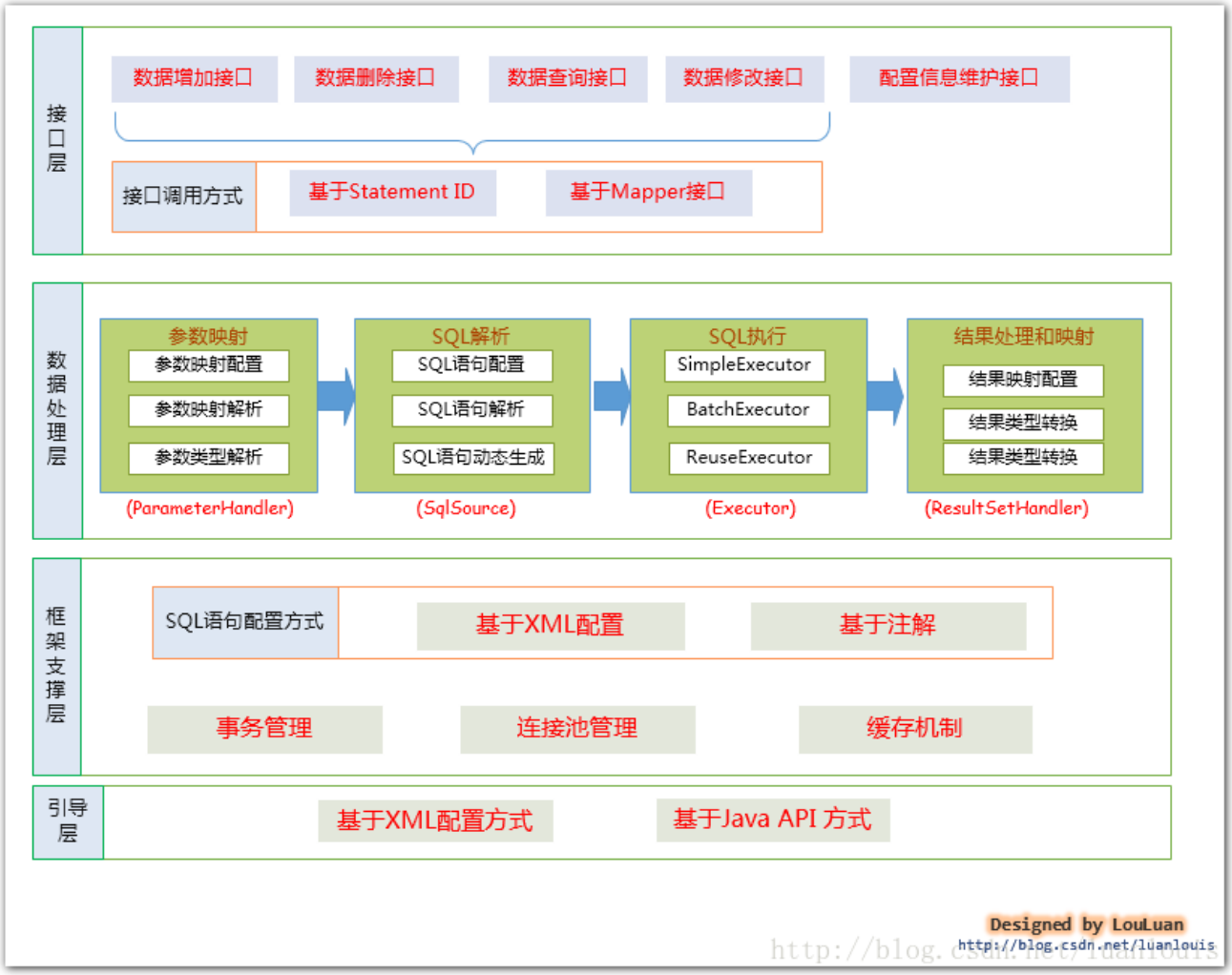
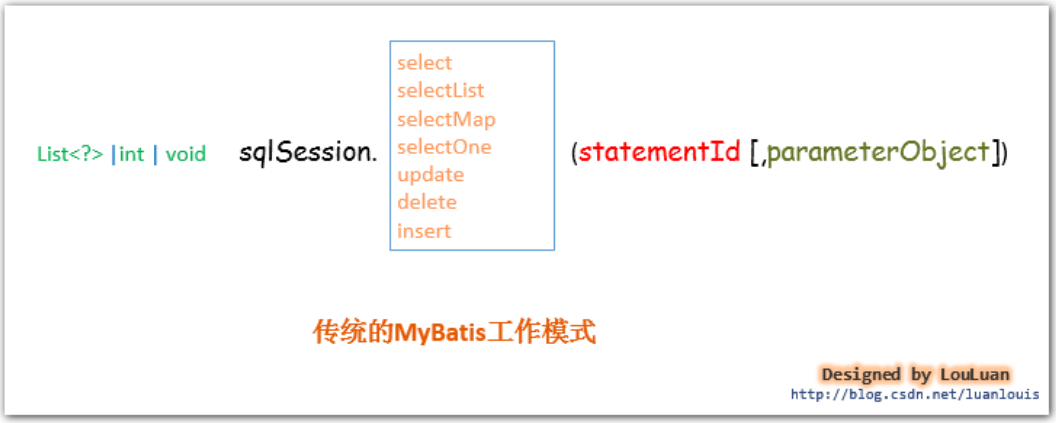
1.MyBatis 的框架设计



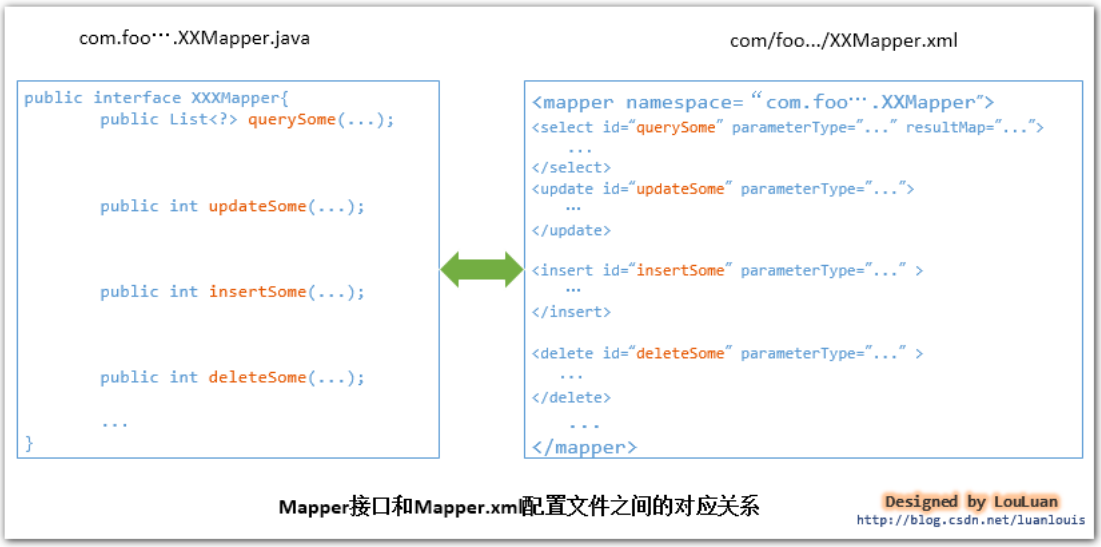
1.1 接口层调用方式

- 使用传统的 MyBatis 提供的 API



```
1. SqlSession sqlSession = sqlSessionFactory.openSession();
2. List list = sqlSession.selectList("com.foo.bean.BlogMapper.queryAllBlogI
   nfo");
```

- 使用 Mapper 接口




```
1. SqlSession sqlSession = sqlSessionFactory.openSession();
2. BlogMapper mapper = sqlSession.getMapper(BlogMapper.class)
3. List list = mapper.selectList();
```

1.2 数据处理层功能

- 通过传入参数构建动态 SQL 语句，使得 MyBatis 有很强的灵活性和扩展性。
- SQL 语句的执行以及封装查询结果集自动转换成 List<E>，并且支持嵌套查询语句的查询，还有一种是嵌套结果集的查询。
- 支持参数映射（Java 数据类型和 **jdbc** 数据类型之间的转换）

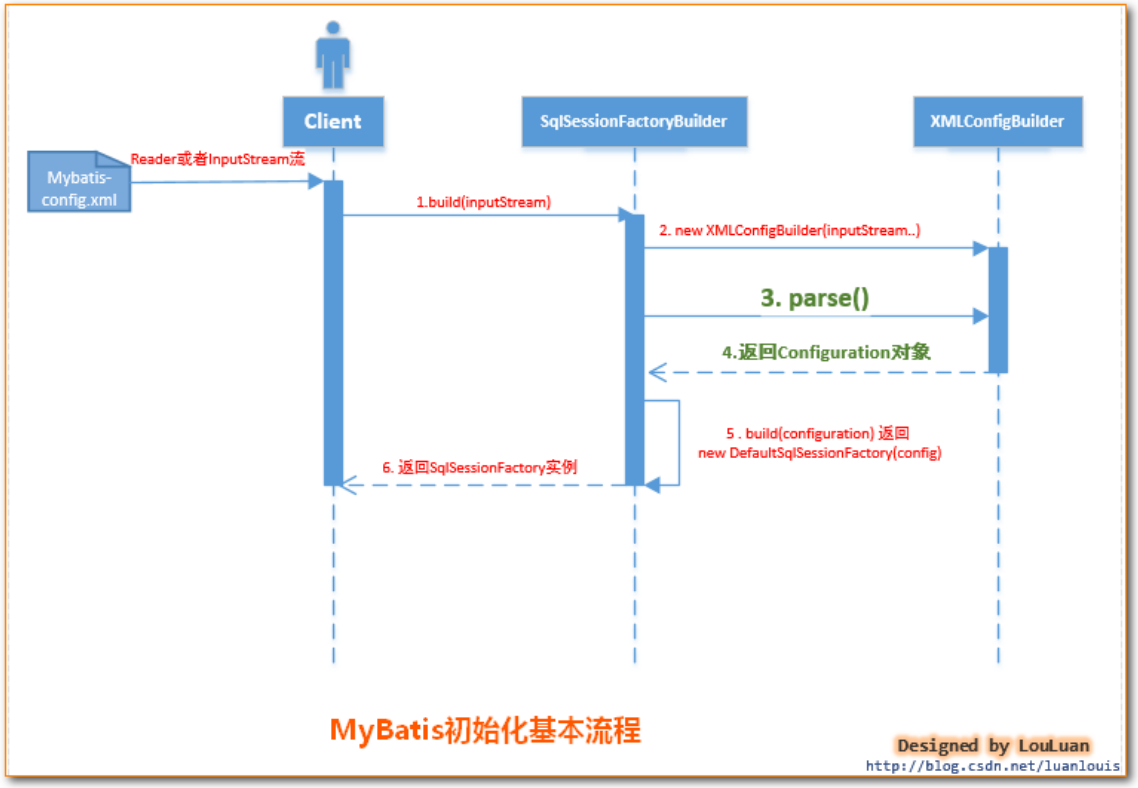
```
<resultMap id="userResultMap" type="org.sang.bean.User">
        <result typeHandler="org.sang.db.MyDateTypeHandler"
                column="reg_time"
                javaType="java.util.Date"
                jdbcType="VARCHAR"
                property="regTime"/>
</resultMap>
```

1.3 框架支撑层和引导层

（见框架图）

2. MyBatis 初始化

```
1. String resource = "mybatis-config.xml";
2. InputStream inputStream = Resources.getResourceAsStream(resource);
3. SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build
   (inputStream);
4. SqlSession sqlSession = sqlSessionFactory.openSession();
5. List list = sqlSession.selectList("com.foo.bean.BlogMapper.queryAllBlogInf
   o");
```



- mybatis-config.xml 相关配置属性
  - × configuration 配置
    - ×properties 属性
    - ×settings 设置
    - ×typeAliases 类型命名
    - ×typeHandlers 类型处理器
    - ×objectFactory 对象工厂

×plugins 插件
×environments 环境
×environment 环境变量
×transactionManager 事务管理器
×dataSource 数据源
×mapper 映射器

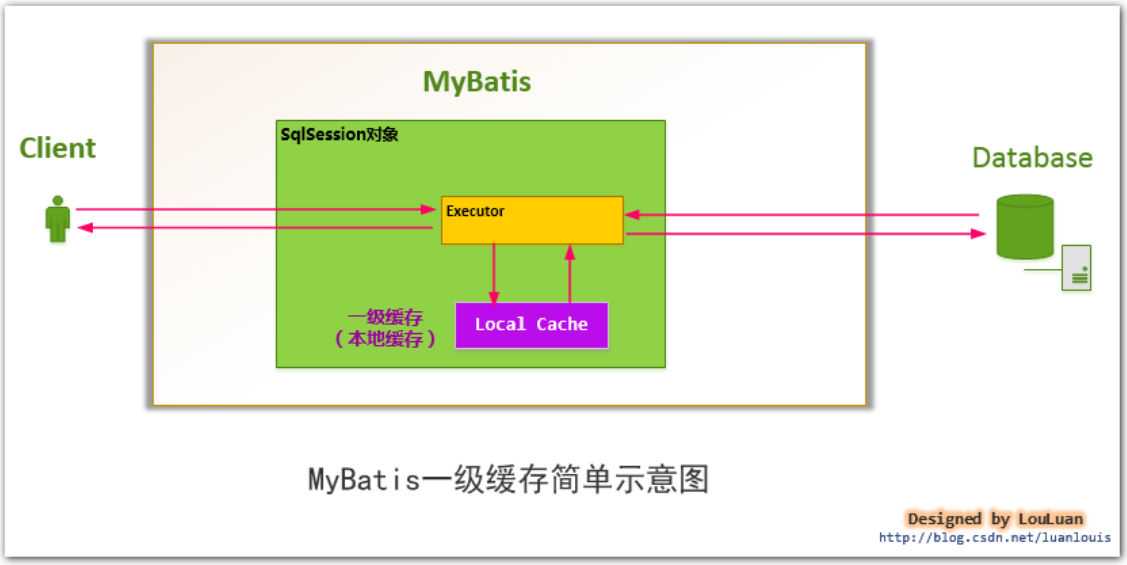
这些配置属性会被映射成一个 org.apache.ibatis.session.Configuration 对象，作为配置信息的容器，保存在 org.apache.ibatis.session.SqlSessionFactory 对象中。

3. MyBatis 缓存

MyBatis 提供了一级缓存、二级缓存这两个缓存机制，能够很好地处理和维护缓存，以提高系统的性能。

3.1 一级缓存

每当我们使用 MyBatis 开启一次和数据库的会话，MyBatis 会创建出一个 SqlSession 对象表示一次数据库会话。Mybatis 一级缓存是**会话级别的缓存**，也就是说，一级缓存会保存在 **SqlSession 对象**中。



➢ **一级缓存生命周期**

1. 如果 SqlSession 调用了 close()方法，会释放掉一级缓存缓存对象，一级缓存将不可用；
2. 如果 SqlSession 调用了 clearCache()，会清空缓存对象中的数据，但是该对象仍可使用；
3. SqlSession 中执行了任何一个 update 操作(update()、delete()、insert()) ，都会清空缓存对象的数据，但是该对象可以继续使用；

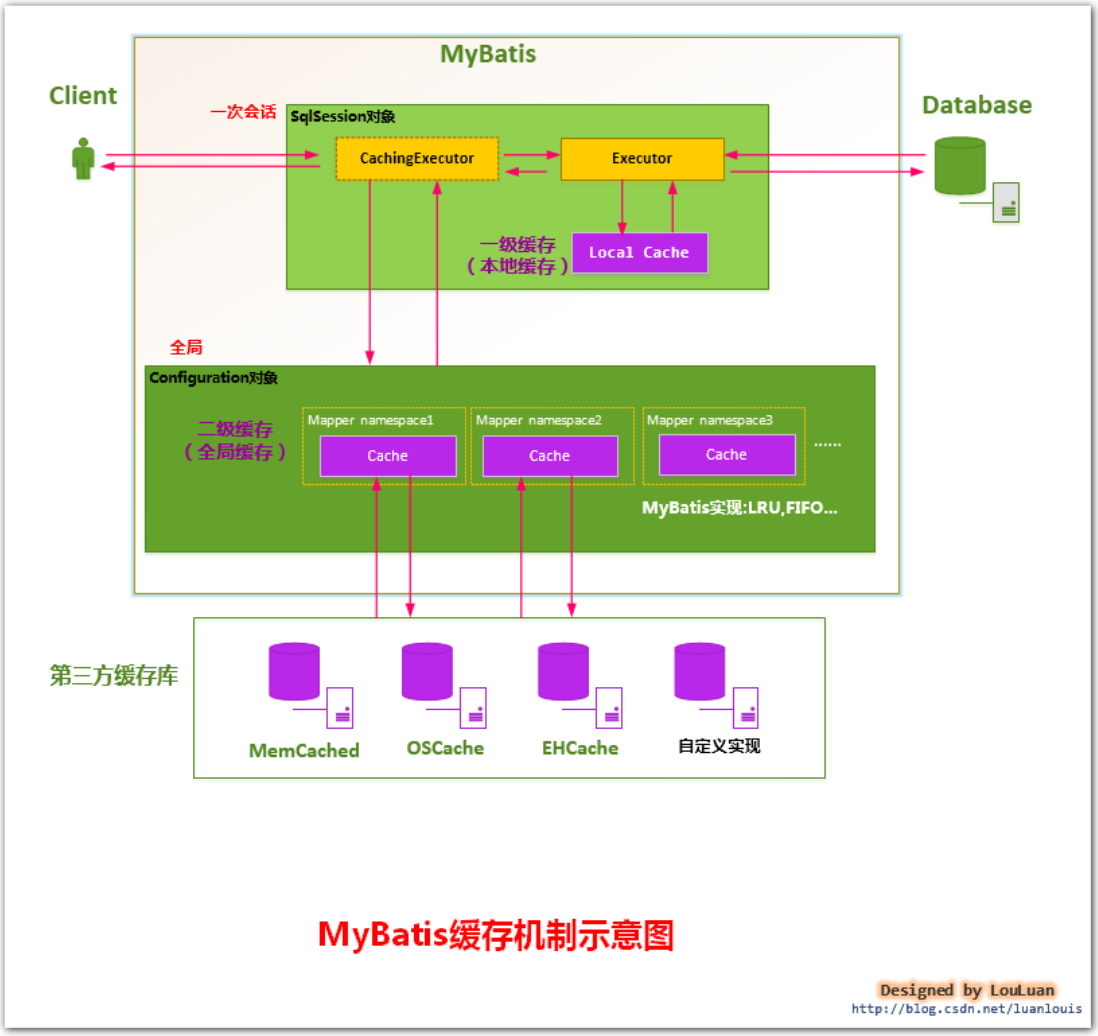
➢ key 一致性判断

满足 **statementId + rowBounds +传递给 JDBC 的 SQL +传递给 JDBC 的参数值** 一样。

一级缓存是一个粗粒度的缓存，没有更新缓存、缓存过期和容量限制的概念。一级缓存默认打开，可以通过配置 mapper 中 statmentId 的属性 flushCache 来关闭。

3.2 二级缓存

MyBatis 的二级缓存是 Application 级别的缓存。



MyBatis 的二级缓存机制的关键就是对这个 Executor 对象做文章。如果用户配置了"cacheEnabled=true"(默认即为 true)，那么 MyBatis 在为 SqlSession 对象创建 Executor 对象时，会对 Executor 对象加上一个装饰者：CachingExecutor，这时 SqlSession 使用 CachingExecutor 对象来完成操作请求。CachingExecutor 对于查询请求，会先判断该查询请求在 Application 级别的二级缓存中是否有缓存结果，如果有查询结果，则直接返回缓存结果；如果缓存中没有，再交给真正的 Executor 对象来完成查询操作，之后 CachingExecutor 会将真正 Executor 返回的查询结果放置到缓存中，然后在返回给用户。

➢ 2.MyBatis 二级缓存的划分

MyBatis 并不是简单地对整个 **Application** 就只有一个 **Cache** 缓存对象，它将缓存划分的更细，即是 **Mapper** 级别的，即每一个 Mapper 都可以拥有一个 **Cache** 对象，配置方式：

a.为每一个 Mapper 分配一个 Cache 缓存对象（使用<cache>节点配置）；

b.多个 Mapper 共用一个 Cache 缓存对象（使用<cache-ref>节点配置）；

c.在<select>节点中配置 useCache="true"，Mapper 才会对此 Select 的查询支持缓存特性。

d.自定义缓存，用户是需要实现 org.apache.ibatis.cache.Cache 接口，然后将 Cache 实现类配置在<cache type="">节点的 type 属性，也可集成第三方缓存（如 Memecached）。

4. MyBatis 事务机制

```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC" />
        <dataSource type="POOLED">
            <property name="driver" value="com.mysql.jdbc.Driver" />
            <property name="url" value="jdbc:mysql://localhost:3306/test" />
            <property name="username" value="root" />
            <property name="password" value="BIUBIU" />
        </dataSource>
    </environment>
</environments>
```

MyBatis 的事务管理分为两种形式：

1、使用 JDBC 的事务管理机制：即利用 java.sql.Connection 对象完成对事务的提交（commit()）、回滚（rollback()）、关闭（close()）等。

2、使用 MANAGED 的事务管理机制：这种机制 MyBatis 自身不会去实现事务管理，而是让程序的容器如（Spring，JBOSS，Weblogic）来实现对事务的管理。

➢ 事务对 UPDATE 的影响

- 如果开启 MyBatis 事务管理，则需要手动进行事务提交，否则事务会回滚到原状态
  1. 如果不执行 sqlSession.commit()操作，直接执行 sqlSession.close()，则会在 close()中进行事务回滚。
  2. 如果不执行 sqlSession.commit()操作也不手动关闭 sqlSession，在程序结束时关闭数据库连接时会进行事务回滚；

➢ 事务对 SELECT 的影响

- 事务对 select 操作的影响主要体现在对缓存的影响上，主要包括一级缓存和二级缓存
  1. 一级缓存是 Session 级别的，因此事务的提交回滚对 MyBatis 的一级缓存没有影响;
  2. 如果从二级缓存中未命中缓存，则需要从数据库中查取，再将查询结果放入二级缓存中；查询结果首先放入二级缓存临时缓存（TransactionalCache，sqlsession 级别，存在 CacheExecutor）中，只有执行了 commit()事务提交才正式转移到正式缓存中；也就是说只有执行了 commit()方法的缓存才被下次查询使用，不然仍会执行数据库查询任务并覆盖上次的临时缓存。

5. MyBatis-Spring 使用

MyBatis-Spring ：将 MyBatis 代码无缝地整合到 Spring 中。 使用这个类库中的类, Spring 将会加载必要的 MyBatis 工厂类和 session 类。 这个类库也提供一个简单的方式来注入 MyBatis 数据映射器和 SqlSession 到业务层的 bean 中。 而且它也会处理事务, 翻译 MyBatis 的异常到 Spring 的 DataAccessException 异常。

- **以 TSM 新架构为例**

```
<!-- MyBatis 映射文件 -->
<bean id="sqlSessionFactoryForOnl" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSourceForOnl" />
    <!-- 自动扫描 entity 目录, 省掉 Configuration.xml 里的手工配置 -->
    <property name="mapperLocations" value="classpath:sqlmap/onl/*.xml" />
</bean>

<!--MyBatis 通过接口来加载配置文件 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.up.tsm.svr.data.dao.onl" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactoryForOnl" />
</bean>

<!-- 配置事务管理器 -->
<bean id="transactionManagerForOnl"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSourceForOnl" />
</bean>

<!-- 注解方式配置事物 -->
<tx:annotation-driven transaction-manager="transactionManagerForOnl" />
```

MapperScannerConfigurer 将会创建 MapperFactoryBean,之后自动装配。