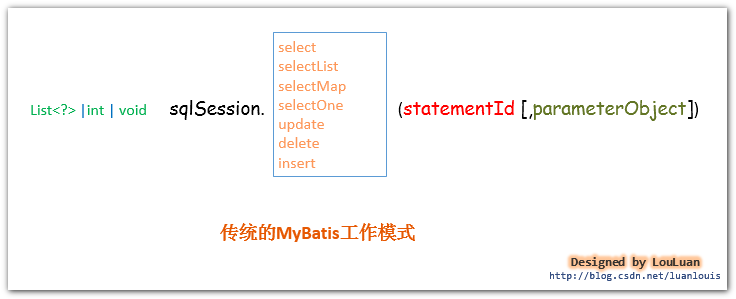
# http://img.blog.csdn.net/20141028232313593?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbHVhbmxvdWlz/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEastMyBatis的框架设计

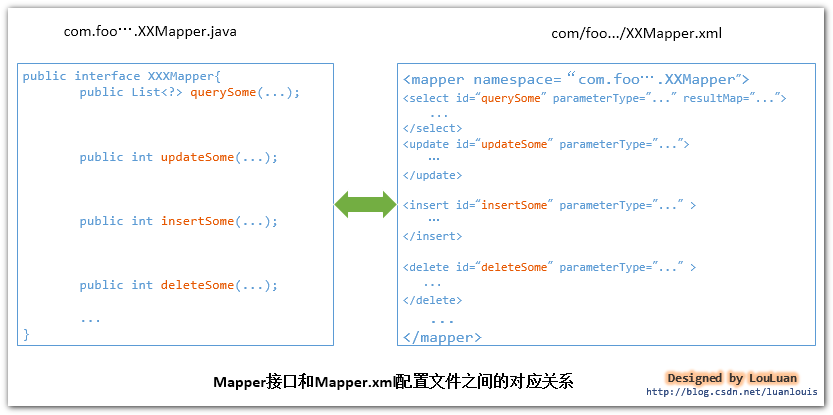
## 1.1 接口层调用方式

* 使用传统的MyBatis提供的API



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

* 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](http://lib.csdn.net/base/17)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"/>

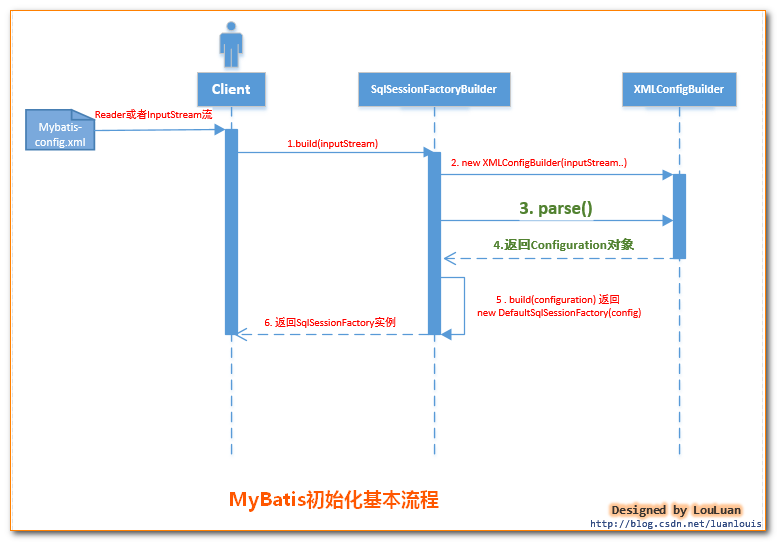
</resultMap>

## 1.3 框架支撑层和引导层

（见框架图）

# 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.queryAllBlogInfo");



* **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

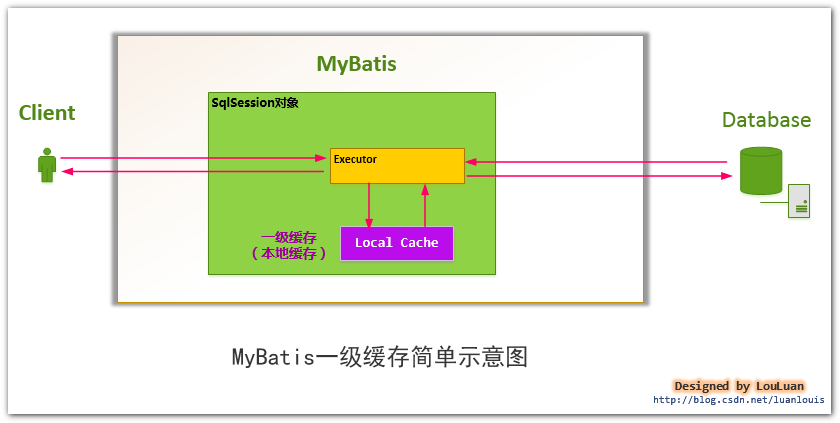
这些配置属性会被映射成一个org.apache.ibatis.session.Configuration 对象，作为配置信息的容器，保存在org.apache.ibatis.session.SqlSessionFactory对象中。

# 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）。**

# MyBatis事务机制

1. <environments default="development">
2. <environment id="development">
3. <transactionManager type="JDBC" />
4. <dataSource type="POOLED">
5. <property name="driver" value="com.mysql.jdbc.Driver" />
6. <property name="url" value="jdbc:mysql://localhost:3306/test" />
7. <property name="username" value="root" />
8. <property name="password" value="BIUBIU" />
9. </dataSource>
10. </environment>
11. </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。

# 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。