1、Mybatis 动态 sql 是做什么的？都有哪些动态 sql？简述一下动态 sql 的执行原理？

Q1）动态 sql 就是（在进行 sql 操作的时候）动态的根据属性值（所匹配的条件）来拼接数据库执行的 sql 语句，也就是多次查询或变更操作，根据传入的属性值不同，动态拼接出不同的可执行 sql。包含判断为空、循环、拼接等情况；

Q2）常见动态 sql 标签：

1） if 和 where 标签和 include 标签

```
<!--动态Sql : where / if-->
<select id="findUserById"  resultType="com.lagou.pojo.User">
  select <include refid="userInfo"/> from user
  <where>
    <if test="id != null and id != 0">
      AND id = #{id}
    </if>
    <if test="name != null and name != ''">
      AND name = #{name}
    </if>
  </where>
</select>
```

2） choose、when、otherwise 标签

```
<!--动态Sql: choose、when、otherwise 标签-->
<select id="findUserById" resultType="com.lagou.pojo.User">
  select * from user
  <where>
    <choose>
      <when test="name != null and name != ''">
        AND name = #{name}
      </when>
      <otherwise>
        AND id = #{id}
      </otherwise>
    </choose>
  </where>
</select>
```

3） foreach 标签

```xml
<!--动态sql: foreach标签, 批量插入-->
<insert id="insertBatch" useGeneratedKeys="true" keyProperty="id">
  insert into user (id, name)
  values
  <foreach collection="list" item="user" separator="," >
    (#{user.id}, #{user.name})
  </foreach>
</insert>

<!--动态sql: foreach标签, in查询-->
<select id="dynamicSqlSelectList" resultType="com.lagou.pojo.User">
  SELECT * from user WHERE id in
  <foreach collection="list" item="id" open="(" close=")" separator="," >
    #{id}
  </foreach>
</select>
```

4） set 标签

```xml
<!--动态sql: set 标签-->
<update id="updateSet" parameterType="com.lagou.pojo.User">
  update user
  <set>
    <if test="name != null and name != ''">
      name = #{name},
    </if>
  </set>
  where id = #{id}
</update>
```

5） trim 标签

```xml
<!--动态sql: trim 标签-->
<select id="findUser" resultType="com.lagou.pojo.User">
  select * from user
  <trim prefix="where" suffix="order by id" prefixOverrides="and | or" suffixOverrides=",">
    <if test="name != null and name != ''">
      AND name = #{name}
    </if>
    <if test="id != null">
      AND id = #{id}
    </if>
  </trim>
</select>
```

6） map 参数

```xml
<!--动态sql: foreach标签, map参数查询-->
<select id="findByMap" resultType="com.lagou.pojo.User">
  select * from user WHERE
  <foreach collection="map" index="key" item="value"  separator="=">
    ${key} = #{value}
  </foreach>
</select>
```

Q3) 动态 sql 执行原理

- 首先在解析 xml 配置文件的时候，会有一个 SqlSource sqlSource = langDriver.createSqlSource(configuration, context, parameterTypeClass) 的操作；
- createSqlSource 底层使用了 XMLScriptBuilder 来对 xml 中的标签进行解析；
- XMLScriptBuilder 调用了 parseScriptNode()的方法；
- 在 parseScriptNode()的方法中有一个 parseDynamicTags()方法，会对 nodeHandlers 里的标签根据不同的 handler 来处理不同的标签；
- 然后把 DynamicContext 结果放回 SqlSource 中；
- DynamicSqlSource 获取 BoundSql；
- 在 Executor 执行的时候，调用 DynamicSqlSource 的解析方法，并返回封装好的 BoundSql 对象。

2、Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

mybatis 仅支持关联对象 association 和关联集合对象 collection 的延迟加载。MyBatis 根据对关联对象查询的 select 语句的执行时机，分为三种类型：直接加载、侵入式加载与深度延迟加载。

- 直接加载：执行完对主加载对象的 select 语句，马上执行对关联对象的 select 查询。
- 侵入式延迟：执行对主加载对象的查询时，不会执行对关联对象的查询。但当要访问主加载对象的详情时，就会马上执行关联对象的 select 查询。即对关联对象的查询执行，侵入到了主加载对象的详情访问中。也可以这样理解：将关联对象的详情侵入到了主加载对象的详情中，即将关联对象的详情作为主加载对象的详情的一部分出现了。
- 深度延迟：执行对主加载对象的查询时，不会执行对关联对象的查询。访问主加载对象的详情时也不会执行关联对象的 select 查询。只有当真正访问关联对象的详情时，才会执行对关联对象的 select 查询。

  原理：简单的说是，使用 CGLIB 生成代理对象,对象方法调用时执行查询语句。

3、Mybatis 都有哪些 Executor 执行器？它们之间的区别是什么？

BaseExcutor	SimpleExecutor	每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。	Executor的这些特点，都严格限制在SqlSession生命周期范围内
	ReuseExecutor	执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map内，供下一次使用。简言之，就是重复使用Statement对象。	
	BatchExecutor	执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理。与JDBC批处理相同。	
CachingExcutor	二级缓存执行器		全局

4、 简述下 Mybatis 的一级、二级缓存（分别从存储结构、范围、失效场景。三个方面来作答）？

	存储结构	范围	失效场景	缓存顺序
一级缓存	hashMap	SqlSession范围	1.SqlSession执行了DML操作（增删改），并且提交到数据库，MyBatis则会清空SqlSession中的一级缓存 2.当一个SqlSession结束后该SqlSession中的一级缓存也就不存在了 3. 手动清空一级缓存	
二级缓存	hashMap	mapper的同一个namespace	（1）LRU最近最少使用策略，一处做长时间不被使用的对象。 （2）FIFO先进先出策略，按对象进入缓存的顺序来移除它们。 （3）SOFT软引用策略，移除基于垃圾回收器状态和软引用规则的对象。 （4）WEAK弱引用策略，更积极地移除基于垃圾收集器状态和弱引用规则的对象	缓存的顺序：先看二级缓存再看一级缓存，如果都没有在查询数据

5、 简述 Mybatis 的插件运行原理，以及如何编写一个插件？

Q1)运行原理：

* 在四大对象创建的时候
 * 1、每个创建出来的对象不是直接返回的，而是
 * interceptorChain.pluginAll(parameterHandler);
 * 2、获取到所有的 Interceptor（拦截器）（插件需要实现的接口）；
 * 调用 interceptor.plugin(target);返回 target 包装后的对象
 * 3、插件机制，我们可以使用插件为目标对象创建一个代理对象；AOP（面向切面）
 * 我们的插件可以为四大对象创建出代理对象；
 * 代理对象就可以拦截到四大对象的每一个执行；

Q2) 编写插件：

1、 创建插件类实现 interceptor 接口并且使用注解标注拦截对象与方法

```
@Intercepts({
        @Signature(type = Executor.class, method = "query",
                args = {MappedStatement.class, Object.class, RowBounds.class, ResultHandler.class})})
public class TestInterceptor implements Interceptor {
```

```java
@Override
public Object intercept(Invocation invocation) throws Throwable {
    //被代理对象
    Object target = invocation.getTarget();
    //代理方法
    Method method = invocation.getMethod();
    //方法参数
    Object[] args = invocation.getArgs();
    // do something ..... 方法拦截前执行代码块
    //执行原来方法
    Object result = invocation.proceed();
    // do something .......方法拦截后执行代码块
    return result;
}
```

```java
@Override
public Object plugin(Object target) {
    System.out.println("MySecondPlugin为目标对象" + target + "创建代理对象");
    //this表示当前拦截器，target表示目标对象，wrap方法利用mybatis封装的方法为目标对象创建代理对象
    Object wrap = Plugin.wrap(target, this);
    return wrap;
}
```

```java
@Override
public void setProperties(Properties properties) {
    System.out.println(properties);
}
```

2、 在配置文件中写入 plugins 标签

```xml
<plugins>
        <plugin interceptor="city.albert.TestInterceptor">
            <property name="name" value="name"/>
        </plugin>
    </plugins>
```