게시판(MyBatis)

MyBatis는 무엇인가?

1. MyBatis는 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다. MyBatis는 JDBC코드와 수동으로 셋팅하는 파라미터와 결과 매핑을 제거한다. MyBatis는 데이터베이스 레코드에 원시타입과 Map인터페이스 그리고 자바 POJO를 설정하고 매핑하기 위해 XML과 애노테이션을 사용할 수 있다

2. 모든 MyBatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다. SqlSessionFactory 인스턴스는 SqlSessionFactoryBuilder 를 사용하여 만들수 있다. SqlSessionFactoryBuilder 는 XML 설정파일에서 SqlSessionFactory 인스턴스를 빌드할 수 있다.

XML에서 SqlSessionFactory 빌드하기

```
<?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>
   <environments default="development">   <environment id="development">
   <transactionManager type="JDBC"/>
   <dataSource type="POOLED">
      <property name="driver" value="${driver}"/>
      <property name="url" value="${url}"/>
      <property name="username" value="${username}"/>
      <property name="password" value="${password}"/>
   </dataSource>
</environment>  </environments>
<mappers><mapper resource="org/mybatis/example/BlogMapper.xml"/></mappers>
</configuration>
```

XML을 사용하지 않고 SqlSessionFactory 빌드하기

1. XML보다 자바를 사용해서 직접 설정하길 원한다면, XML파일과 같은 모든 설정을 제공하는 Configuration 클래스를 사용하면 된다.

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory();
Environment environment = new Environment("development", transactionFactory, dataSource);
Configuration configuration = new Configuration(environment);
configuration.addMapper(BlogMapper.class);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);
```

이 설정에서 추가로 해야 할 일은 Mapper 클래스를 추가하는 것이다. Mapper 클래스는 SQL 매핑 애노테이션을 가진 자바 클래스이다. 어쨌든 자바 애노테이션의 몇가지 제약과 몇가지 설정 방법의 목잡함에도 불구하고, XML매핑은 세부적인 매핑을 위해 언제든 필요하다.

SqlSessionFactory 에서 SqlSession 만들기

1. SqlSessionFactory 이름에서 보듯이, SqlSession 인스턴스를 만들수 있다.
SqlSession 은 데이터베이스에 대해 SQL 명령어를 실행하기 위해 필요한 모든 메서드를 가지고 있다. 그래서 SqlSession 인스턴스를 통해 직접 SQL구문을 실행할 수 있다.

```
SqlSession session = sqlMapper.openSession();
try {
     Blog blog =
     (Blog) session.selectOne( "org.mybatis.example.BlogMapper.selectBlog", 101);
} finally {
    session.close();
 }
```

주어진 SQL구문의 파라미터와 리턴값을 설명하는 인터페이스(예를 들면, BlogMapper.class)를 사용하여, 문자열 처리 오류나 타입 캐스팅 오류 없이 좀더 타입에 안전하고 깔끔하게 실행할 수 있다.

```
SqlSession session = sqlSessionFactory.openSession();
    try {
        BlogMapper mapper = session.getMapper(BlogMapper.class);
        Blog blog = mapper.selectBlog(101);
    } finally {
            session.close();      }
```

SqlSessionFactory

한번 만든뒤, SqlSessionFactory는 애플리케이션을 실행하는 동안 존재해야만 한다. 그래서 삭제하거나 재생성할 필요가 없다. 애플리케이션이 실행되는 동안 여러 차례 SqlSessionFactory 를 다시 빌드하지 않는 것이 가장 좋은 형태이다. 가장 간단한 방법은 싱글턴 패턴이나 static 싱글턴 패턴을 사용하는 것이다.

SqlSession

각각의 쓰레드는 자체적으로 SqlSession 인스턴스를 가져야 한다. SqlSession 인스턴스는 공유되지 않고 쓰레드에 안전하지도 않다. 그러므로 가장 좋은 스코프는 요청 또는 메서드 스코프이다. SqlSession 을 static 필드나 클래스의 인스턴스 필드로 지정해서는 안된다. HTTP 요청을 받을때마다 만들고, 응답을 리턴할때마다 SqlSession을 닫을 수 있다. SqlSession을 닫는 것은 중요하다. 언제나 finally 블록에서 닫아야만 한다.

```
try {   // do work
} finally {  session.close();  }
```

Mapper 인스턴스

Mapper는 매핑된 구문을 바인딩 하기 위해 만들어야 할 인터페이스이다. mapper인터페이스의 인스턴스는 SqlSession에서 생성한다.

```
SqlSession session = sqlSessionFactory.openSession();
try {  BlogMapper mapper = session.getMapper(BlogMapper.class);
        // do work
} finally {  session.close();  }
```

매퍼 설정 XML

MyBatis XML설정파일은 다양한 셋팅과 프로퍼티를 가진다.

- configuration
  - properties
  - settings
  - typeAliases
  - typeHandlers
  - objectFactory
  - plugins,
  - environments
    - environment
      - transactionManager
      - dataSource
  - mappers

properties

이 설정은 외부에 옮길 수 있다. 자바 프로퍼티 파일 인스턴스에 설정할 수도 있고, properties 요소의 하위 요소에 둘수도 있다. 예를 들면:

```
<properties resource="org/mybatis/example/config.properties">
    <property name="username" value="dev_user"/>
    <property name="password" value="F2Fa3!33TYyg"/>
</properties>
```

속성들은 파일 도처에 둘수도 있다. 예를 들면:

```
<dataSource type="POOLED">
     <property name="driver" value="${driver}"/>
     <property name="url" value="${url}"/>
     <property name="username" value="${username}"/>
     <property name="password" value="${password}"/>
</dataSource>
```

속성은 SqlSessionBuilder.build() 메서드에 전달될 수 있다.

```
 SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, props);
// ... or ...
  SqlSessionFactory factory =
        sqlSessionFactoryBuilder.build(reader, environment, props);
```

속성이 한개 이상 존재한다면, MyBatis는 일정한 순서로 로드한다.:

1. properties 요소에 명시된 속성
2. properties요소의 클래스패스 자원이나 url속성
3. 메서드 파라미터로 전달된 속성을 읽는다

settings

셋팅	설명	사용가능한 값들	디폴트
cacheEnabled	설정에서 각 mapper 에 설정된 캐시를 전역적으로 사용할지 말지에 대한 여부	true | false	true
lazyLoadingEnabled	늦은 로딩을 사용할지에 대한 여부. 사용하지 않는다면 모두 즉시 로딩할 것이다.	true | false	true
aggressiveLazyLoading	활성화 상태로 두게 되면 늦은(lazy) 로딩 프로퍼티를 가진 객체는 호출에 따라 로드될 것이다. 반면에 개별 프로퍼티는 요청할때 로드된다.	true | false	true
multipleResultSetsEnabled	한개의 구문에서 여러개의 ResultSet 을 허용할지의 여부(드라이버가 해당 기능을 지원해야 함)	true | false	true
useColumnLabel	칼럼명 대신에 칼럼라벨을 사용. 드라이버마다 조금 다르게 작동한다. 문서와 간단한 테스트를 통해 실제 기대하는 것처럼 작동하는지 확인해야 한다.	true | false	true
useGeneratedKeys	생성키에 대한 JDBC 지원을 허용. 지원하는 드라이버가 필요하다. true 로 설정하면 생성키를 강제로 생성한다. 일부 드라이버(예를들면, Derby)에서는 이 설정을 무시한다.	true | false	False
autoMappingBehavior	MyBatis 가 칼럼을 필드/프로퍼티에 자동으로 매핑할지와 방법에 대해 명시. PARTIAL 은 간단한 자동매핑만 할뿐, 내포된 결과에 대해서는 처리하지 않는다. FULL 은 처리가능한 모든 자동매핑을 처리한다.	NONE, PARTIAL, FULL	PARTIAL
defaultExecutorType	디폴트 실행자(executor) 설정. SIMPLE 실행자는 특별히 하는 것이 없다. REUSE 실행자는 PreparedStatement 를 재사용한다. BATCH 실행자는 구문을 재사용하고 수정을 배치처리한다.	SIMPLE REUSE BATCH	SIMPLE
defaultStatementTimeout	데이터베이스로의 응답을 얼마나 오래 기다릴지를 판단하는 타임아웃을 셋팅	양수	셋팅되지 않음(null)

```
<settings>
<setting name="cacheEnabled" value="true"/>
<setting name="lazyLoadingEnabled" value="true"/>
<setting name="multipleResultSetsEnabled" value="true"/>
<setting name="useColumnLabel" value="true"/>
<setting name="useGeneratedKeys" value="false"/>
<setting name="enhancementEnabled" value="false"/>
<setting name="defaultExecutorType" value="SIMPLE"/>
<setting name="defaultStatementTimeout" value="25000"/>
</settings>
```

typeAliases

타입 별칭은 자바 타입에 대한 좀더 짧은 이름이다. 오직 XML 설정에서만 사용되며, 타이핑을 줄이기 위해 존재한다. 예를들면:

```
<typeAliases>
<typeAlias alias="Author" type="domain.blog.Author"/>
<typeAlias alias="Blog" type="domain.blog.Blog"/>
<typeAlias alias="Comment" type="domain.blog.Comment"/>
<typeAlias alias="Post" type="domain.blog.Post"/>
<typeAlias alias="Section" type="domain.blog.Section"/>
<typeAlias alias="Tag" type="domain.blog.Tag"/>
</typeAliases>
```

이 설정에서, “Blog” 는 도처에서 “domain.blog.Blog” 대신 사용될 수 있다.

select

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
    SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

이 구문의 이름은 selectPerson 이고 int 타입의 파라미터를 가진다. 그리고 결과 데이터는 HashMap에 저장된다. 파라미터 표기법을 보자.

#{id}

이 표기법은 MyBatis 에게 PreparedStatement 파라미터를 만들도록 지시한다. JDBC를 사용할 때 PreparedStatement에는 "?" 형태로 파라미터가 전달된다.

결과적으로 위 설정은 아래와 같이 작동하게 되는 셈이다.

```
String selectPerson = "SELECT * FROM PERSON WHERE ID=?";
PreparedStatement ps = conn.prepareStatement(selectPerson);
ps.setInt(1,id);
```

```
<select  id="selectPerson"          parameterType="int"
 parameterMap="deprecated"    resultType="hashmap"
 resultMap="personResultMap"  flushCache="false"
 useCache="true"                    timeout="10000"
 fetchSize="256" statementType="PREPARED"
 resultSetType="FORWARD_ONLY"  >
```

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
parameterMap	외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType 을 대신 사용하라.
resultType	이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭. collection 이 경우, collection 타입 자체가 아닌 collection 이 포함된 타입이 될 수 있다. resultType 이나 resultMap 을 사용하라.
resultMap	외부 resultMap 의 참조명. 결과맵은 MyBatis 의 가장 강력한 기능이다. resultType 이나 resultMap 을 사용하라.
flushCache	이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지워질것이다(flush). 디폴트는 false 이다.
useCache	이 값을 true 로 셋팅하면, 구문의 결과가 캐시될 것이다. 디폴트는 true 이다.
timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.
fetchSize	지정된 수만큼의 결과를 리턴하도록 하는 드라이버 힌트 형태의 값이다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.
statementType	STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.
resultSetType	FORWARD_ONLY|SCROLL_SENSITIVE|SCROLL_INSENSITIVE 중 하나를 선택할 수 있다. 디폴트는 셋팅하지 않는 것이고 드라이버에 다라 다소 지원되지 않을 수 있다.

insert, update, delete

```
<insert  id="insertAuthor" parameterType="domain.blog.Author"
  flushCache="true" statementType="PREPARED" keyProperty=""
  keyColumn="" useGeneratedKeys="" timeout="20000">
<update id="insertAuthor" parameterType="domain.blog.Author"
 flushCache="true" statementType="PREPARED" timeout="20000">
<delete id="insertAuthor" parameterType="domain.blog.Author"
 flushCache="true" statementType="PREPARED" timeout="20000">
```

속성	설명
Id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
parameterMap	외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType 을 대신 사용하라.
flushCache	이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지워질것이다(flush). 디폴트는 false 이다.
Timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.
statementType	STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.
useGeneratedKeys	(입력(insert)에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어, MySQL 또는 SQL Server 와 같은 RDBMS 의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메서드를 사용하도록 설정하다. 디폴트는 false 이다.
keyProperty	(입력(insert)에만 적용) getGeneratedKeys 메서드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 셋팅할 프로퍼티를 지정. 디폴트는 셋팅하지 않는 것이다.
keyColumn	(입력(insert)에만 적용) 생성키를 가진 테이블의 칼럼명을 셋팅. 키 칼럼이 테이블이 첫번째 칼럼이 아닌 데이터베이스(PostgreSQL 처럼)에서만 필요하다.

Insert, update, delete 구문의 예제이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
insert into Author (id,username,password,email,bio)
values (#{id},#{username},#{password},#{email},#{bio})
</insert>
<update id="updateAuthor" parameterType="domain.blog.Author">
    update Author set username = #{username}, password = #{password},
    email = #{email}, bio = #{bio}  where id = #{id}
</update>
<delete id="deleteAuthor" parameterType="int">
   delete from Author where id = #{id}
</delete>
```

insert 는 key 생성과 같은 기능을 위해 몇가지 추가 속성과 하위 요소를 가진다. 먼저, 사용하는 데이터베이스가 자동생성키(예를 들면, MySQL과 SQL 서버)를 지원한다면, useGeneratedKeys="true" 로 설정하고 대상 프로퍼티에 keyProperty 를 셋팅할 수 있다. 예를 들어, Author 테이블이 id칼럼에 자동생성키를 적용했다고 하면, 구문은 아래와 같은 형태일 것이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author"
  useGeneratedKeys="true" keyProperty="id">
    insert into Author (username,password,email,bio)
    values (#{username},#{password},#{email},#{bio})
</insert>
```

iBatis 는 자동생성키 칼럼을 지원하지 않는 오라클 등은 다른 방법 또한 제공한다.

```
<insert id="InsertOrganization" parameterClass="Organization" resultClass="int">
  <selectKey property="Id" type="pre" resultClass="int">
    select seq_orgs.nextval as value from dual </selectKey>
      INSERT INTO Organizations (Org_Id, Org_Code, Org_Name) VALUES (#Id#,
        #Code#, #Name#)
  </insert>
```

위 예제에서, selectKey 구문이 먼저 실행되고, Author id프로퍼티에 셋팅된다. 그리고 나서 insert 구문이 실행된다. 이건 복잡한 자바코드 없이도 데이터베이스에 자동생성키의 행위와 비슷한 효과를 가지도록 해준다.

myBatis

```
<selectKey keyProperty="id" resultType="int" order="BEFORE"
statementType="PREPARED">
```

속성	설명
keyProperty	selectKey 구문의 결과가 셋팅될 대상 프로퍼티
resultType	결과의 타입. MyBatis 는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을것이다. MyBatis 는 String 을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.
order	BEFORE 또는 AFTER 를 셋팅할 수 있다. BEFORE 로 셋팅하면, 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER 로 셋팅하면, insert 구문을 실행한 뒤 selectKey 구문을 실행한다. Oracle 과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출 형태로 처리한다.
statementType	위 내용과 같다. MyBatis 는 Statement, PreparedStatement 그리고 CallableStatement 을 매핑하기 위해 STATEMENT, PREPARED 그리고 CALLABLE 구문타입을 지원한다.

pom.xml

1. Mybatis설정 추가

```
<dependency>
    <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
       <version>3.2.2</version>
</dependency>
<dependency>
      <groupId>org.mybatis</groupId>
          <artifactId>mybatis-spring</artifactId>
           <version>1.2.0</version>
</dependency>
```

Configuration.xml

```
<?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>
  <properties resource="dbconnect.properties"/>
    <typeAliases>
       <typeAlias alias="BoardBean" type="board.model.BoardBean" />
    </typeAliases>
    <environments default="development">
       <environment id="development">
          <transactionManager type="JDBC"  />
          <dataSource type="POOLED">
             <property name="driver" value="${driver}" />
             <property name="url"
                value="${url}" />
             <property name="username" value="${username}" />
             <property name="password" value="${password}" />
          </dataSource>
       </environment>
    </environments>
    <mappers>
       <mapper resource="mybatis/Board.xml" />
    </mappers>
</configuration>
```

Board.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="Board">
  <!-- Use type aliases to avoid typing the full classname every time. -->
  <resultMap id="BoardResult" type="Board">
    <result property="id" column="id"/>
    <result property="title" column="title"/>
    <result property="writer" column="writer"/>
    <result property="content" column="content"/>
    <result property="hit" column="hit"/>
    <result property="regDate" column="regDate"/>
  </resultMap>
  <!-- Select with no parameters using the result map for Account class. -->
  <select id="listAll" parameterType="Board" resultMap="BoardResult">
    select * from board1
    <where>
       <if test="search=='title'">title like '%'||#{keyword}||'%'</if>
       <if test="search=='writer'"> writer like '%'||#{keyword}||'%' </if>
       <if test="search=='content'"> content like '%'||#{keyword}||'%' </if>
    </where>
    order by id desc
  </select>
```

```
<select id="total" parameterType="Board" resultType="int">
   select count(*) from board1
   <where><if test="search=='title'"> title like '%'||#{keyword}||'%'</if>
   <if test="search=='writer'"> writer like '%'||#{keyword}||'%' </if>
   <if test="search=='content'"> content like '%'||#{keyword}||'%' </if>
   </where>
 </select>
 <insert id="insert" parameterType="Board">
 <selectKey order="BEFORE" keyProperty="id" resultType="int">
 select nvl(max(id),0)+1 id from board1
 </selectKey>
   insert into board1 values ( #{id}, #{title}, #{writer}, #{content},0,sysdate )
 </insert>
  <select id="select" parameterType="int" resultType="Board">
   select * from board1  where id = #{id}
 </select>
 <update id="hitUpdate" parameterType="int">
   update board1 set hit = hit + 1 where id = #{id}
 </update>
 <update id="update" parameterType="Board">
   update board1 set title=#{title},writer=#{writer}, content=#{content}  where id=#{id}
 </update>
 <delete id="delete" parameterType="int">
   delete from board1 where id=#{id}
 </delete>
</mapper>
```

BoardDAO.java

```
package dao;
import java.util.List;
import model.Board;
public interface BoardDao {
        int insert(Board board);
        Board select(int id);
        void hitUpdate(int id);
        int update(Board board);
        int delete(int id);
        int count(Board board);
        List<Board> list(Board board);
}
```

BoardDaoImpl.java

```
package dao;
import java.io.IOException;     import java.io.Reader;  import java.sql.SQLException;
import java.util.List;  import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Repository;    import model.Board;
@Repository
public class BoardDaoImpl implements BoardDao {
        private SqlSession getSession() throws IOException {
                String src = "mybatis/configuration.xml";
                Reader reader = Resources.getResourceAsReader(src);
                SqlSessionFactory ssf =  new SqlSessionFactoryBuilder().build(reader);
                SqlSession session = ssf.openSession(true);
                return session;
        }
        public List<Board> list(Board board) {
                List<Board> list = null; SqlSession session = null;
                try {       session = getSession();
                            list = session.selectList("listAll",board);
                } catch (Exception e) {  System.out.println(e.getMessage());
                } finally { session.close(); }
                return list;
        }
```

```
public int insert(Board board) {
        int result = 0;   SqlSession session = null;
        try {       session = getSession();
                    result = session.insert("insert",board);
        } catch (Exception e) {  System.out.println(e.getMessage());
        } finally { session.close(); }
        return result;
}
public Board select(int id) {
Board board = null;   SqlSession session = null;
        try {       session = getSession();
                    board = (Board)session.selectOne("select",id);
        } catch (Exception e) {  System.out.println(e.getMessage());
        } finally { session.close(); }
        return board;
}
public int count(Board board) {
        int total = 0;   SqlSession session = null;
        try {
                    session = getSession();
                    total = session.selectOne("total",board);
        } catch (Exception e) {
                    System.out.println(e.getMessage());
        } finally { session.close(); }
        return total;
}
```

```
	public void hitUpdate(int id) {
		SqlSession session = null;
		try {	session = getSession();
			session.update("hitUpdate",id);
		} catch (Exception e) {	System.out.println(e.getMessage());
		} finally { session.close(); }
	}
	public int update(Board board) {
		int result = 0;  SqlSession session = null;
		try {
			session = getSession();
			result = session.update("update",board);
		} catch (Exception e) {  System.out.println(e.getMessage());
		} finally { session.close(); }
		return result;
	}
	public int delete(int id) {
		int result = 0;   SqlSession session = null;
		try {
		session = getSession();
		result = session.delete("delete",id);
		} catch (Exception e) {  System.out.println(e.getMessage());
		} finally { session.close(); }
		return result;
	}
}
```