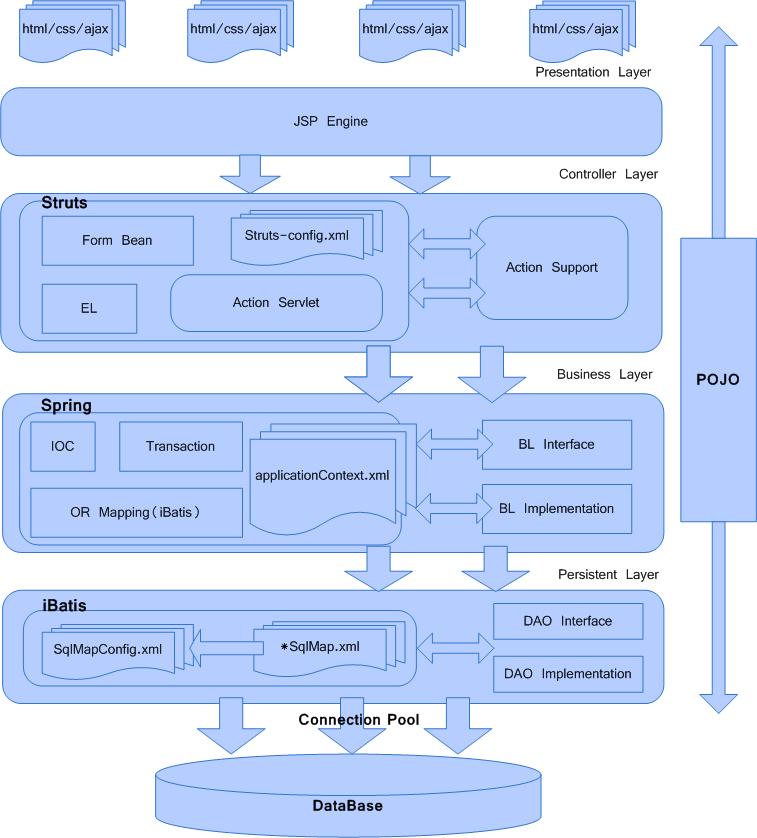
# 一、前言

上次讲了Struts结合Spring并使用Spring的JdbcTemplate来搭建工程框架后我们面临着jar库无法管理，工程发布不方便，jar包在工程内太占空间，jar包冲突，管理，甚至漏包都问题。于是我们在讲“万能框架spring(二)”前，传授了一篇番外篇，即讲利用maven来管理我们的jar库。

从今天开始我们将结合“万能框架spring(一)”与番外篇maven来更进一步丰富我们的****ssx****框架，那么今天讲的是使用iBatis3结合SS来构建我们的****ssi****框架，我们把这个框架命名为beta吧。

# 二、SSI**框架**



还记得我们在第十八天中讲到的我们的框架的架构图吗？****上面这张是我们今天的架构图****，除了Struts，Spring层，我们需要变换的是DAO层即把原来的SQL这部分换成iBatis，我们在次使用的是iBatis版本3。

由于我们在第十八天中已经说了这样的一个框架的好处其中就有：

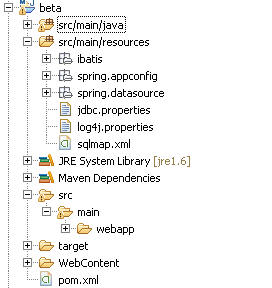
**层中相关技术的替换不影响到其它层面**

所以对于我们来说我们需要改动的代码只有****datasource.xml****与****dao****层的2个接口两个类，那我们就一起来看看这个基于全注解的****SSi****框架是怎么样搭起来的吧。

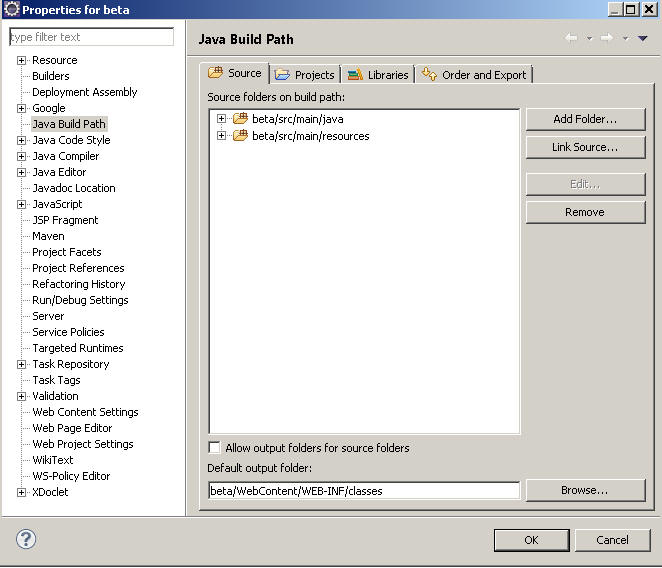
# 三、搭建SSI**框架**

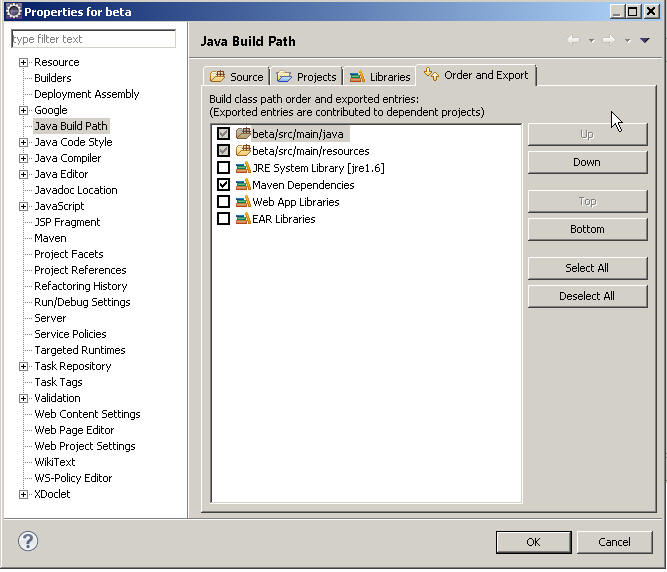
## 3.1**建立工程**

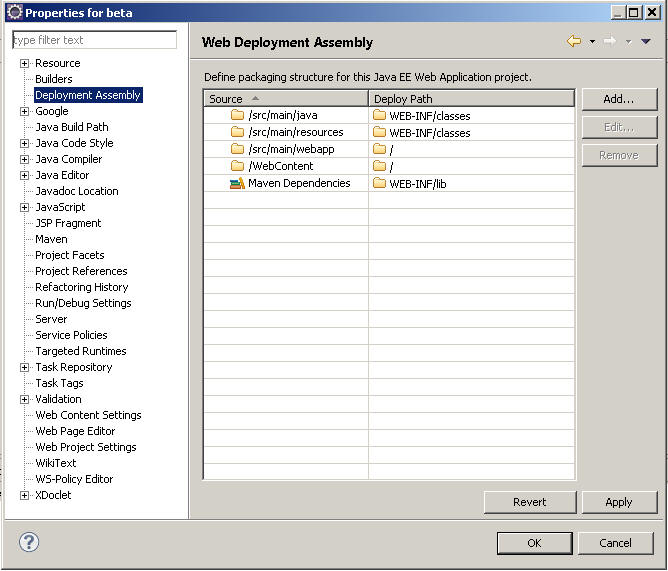
我们还是使用maven来建立我们的工程



建完后照着翻外篇《第十九天》中的“****四、如何让Maven构建的工程在eclipse里跑起来****”对工程进行设置。







## 3.2 **增加iBatis3的jar相关包**

打开pom.xml

## 第一步

找到“slf4j”，将它在pom中的描述改成如下内容：

|  |
| --- |
| <dependency>                          <groupId>org.slf4j</groupId>                          <artifactId>slf4j-api</artifactId>                          <version>****1.5.10****</version>  </dependency> |

## 第二步

增加两个jar包

|  |
| --- |
| <dependency>                          <groupId>org.slf4j</groupId>                          <artifactId>slf4j-log4j12</artifactId>                          <version>1.5.10</version>  </dependency>  <dependency>                          <groupId>org.apache.ibatis</groupId>                          <artifactId>ibatis-core</artifactId>                          <version>3.0</version>  </dependency> |

## 3.3 **开始配置ibatis与spring结合**

打开****/src/main/resources/spring/datasource****下的datasource.xml，增加如下几行

|  |
| --- |
| <bean id="iBatisSessionFactory" class="org.sky.ssi.ibatis.IBatis3SQLSessionFactoryBean" scope="singleton">                                                  <property name="configLocation" value="sqlmap.xml"></property>                                                  <property name="dataSource" ref="dataSource"></property>  </bean>  <bean id="iBatisDAOSupport" class="org.sky.ssi.ibatis.IBatisDAOSupport">  </bean>  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">                          <property name="dataSource" ref="dataSource" />  </bean> |

此处，我们需要4个类，它们是：

### org.sky.ssi.ibatis.IBatis3SQLSessionFactoryBean**类**

|  |
| --- |
| package org.sky.ssi.ibatis;    import java.io.IOException;  import java.io.Reader;  import javax.sql.DataSource;  import org.apache.ibatis.builder.xml.XMLConfigBuilder;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.mapping.Environment;  import org.apache.ibatis.session.Configuration;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.defaults.DefaultSqlSessionFactory;  import org.springframework.beans.factory.FactoryBean;  import org.springframework.beans.factory.InitializingBean;  import org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy;    /\*\*   \*   \* IBatis3SQLSessionFactoryBean is responsible for integrating iBatis 3 <p>   \* with spring 3. Since all environment configurations have been moved to <p>   \* spring, this class takes the responsibility to get environment information<p>   \*  from spring configuration to generate SqlSessionFactory.   \* @author lifetragedy   \*   \*/  public class IBatis3SQLSessionFactoryBean implements FactoryBean<SqlSessionFactory>, InitializingBean{                          private String configLocation;      private DataSource dataSource;      private SqlSessionFactory sqlSessionFactory;      private boolean useTransactionAwareDataSource = true;      private String environmentId = "development";                          public String getConfigLocation() {                                                  return configLocation;                          }                          public void setConfigLocation(String configLocation) {                                                  this.configLocation = configLocation;                          }                          public DataSource getDataSource() {                                                  return dataSource;                          }                          public void setDataSource(DataSource dataSource) {                                                  this.dataSource = dataSource;                          }                          public SqlSessionFactory getSqlSessionFactory() {                                                  return sqlSessionFactory;                          }                          public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {                                                  this.sqlSessionFactory = sqlSessionFactory;                          }                          public boolean isUseTransactionAwareDataSource() {                                                  return useTransactionAwareDataSource;                          }                          public void setUseTransactionAwareDataSource(                                                                          boolean useTransactionAwareDataSource) {                                                  this.useTransactionAwareDataSource = useTransactionAwareDataSource;                          }                          public String getEnvironmentId() {                                                  return environmentId;                          }                          public void setEnvironmentId(String environmentId) {                                                  this.environmentId = environmentId;                          }                            public SqlSessionFactory getObject() throws Exception {          return this.sqlSessionFactory;      }        public Class<SqlSessionFactory> getObjectType() {          return  SqlSessionFactory.class;      }        public boolean isSingleton() {          return true;      }        public void afterPropertiesSet() throws Exception {          this.sqlSessionFactory = this.buildSqlSessionFactory(configLocation);      }        protected SqlSessionFactory buildSqlSessionFactory(String configLocation)      throws IOException {                          if (configLocation == null) {                                                  throw new IllegalArgumentException(                                                  "configLocation entry is required");                          }                          DataSource dataSourceToUse = this.dataSource;                          if (this.useTransactionAwareDataSource                                                                          && !(this.dataSource instanceof TransactionAwareDataSourceProxy)) {                                                  dataSourceToUse = new TransactionAwareDataSourceProxy(                                                                                                  this.dataSource);                          }                            Environment environment = new Environment(environmentId,                                                                          new IBatisTransactionFactory(dataSourceToUse), dataSourceToUse);                            Reader reader = Resources.getResourceAsReader(configLocation);                          XMLConfigBuilder parser = new XMLConfigBuilder(reader, null, null);                          Configuration config = parser.parse();                          config.setEnvironment(environment);                            return new DefaultSqlSessionFactory(config);      }    } |

### org.sky.ssi.ibatis.IBatisDAOSupport

|  |
| --- |
| package org.sky.ssi.ibatis;    import javax.annotation.Resource;    import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.log4j.Logger;      /\*\*   \* Base class for all DAO class. The subclass extends this class to get   \* <p>   \* DAO implementation proxy.   \*   \* @author lifetragedy   \*   \* @param <T>   \*/  public class IBatisDAOSupport<T> {                            protected Logger log = Logger.getLogger(this.getClass());                            @Resource                          private SqlSessionFactory ibatisSessionFactory;                            private T mapper;                            public SqlSessionFactory getSessionFactory() {                                                  return ibatisSessionFactory;                          }                            protected SqlSession getSqlSession() {                                                  return ibatisSessionFactory.openSession();                          }                            public T getMapper(Class<T> clazz) {                                                  mapper = getSqlSession().getMapper(clazz);                                                  return mapper;                          }                            public T getMapper(Class<T> clazz, SqlSession session) {                                                  mapper = session.getMapper(clazz);                                                  return mapper;                          }                            /\*\*                           \* close SqlSession                           \*/                          protected void closeSqlSession(SqlSession sqlSession) throws Exception {                                                  try {                                                                          if (sqlSession != null) {                                                                                                  sqlSession.close();                                                                                                  sqlSession = null;                                                                          }                                                  } catch (Exception e) {                                                  }                          }  } |

### org.sky.ssi.ibatis.IBatisTransaction

|  |
| --- |
| package org.sky.ssi.ibatis;  import java.sql.Connection;  import java.sql.SQLException;  import javax.sql.DataSource;  import org.apache.ibatis.transaction.Transaction;  import org.springframework.jdbc.datasource.DataSourceUtils;  public class IBatisTransaction implements Transaction{                          private DataSource dataSource;                          private Connection connection;                          public IBatisTransaction(DataSource dataSource, Connection con, boolean autoCommit){                                                  this.dataSource = dataSource;                                                  this.connection = con;                          }                            public Connection getConnection(){                                                  return connection;                          }        public void commit()          throws SQLException{                        }        public void rollback()          throws SQLException{                        }        public void close()          throws SQLException{                          if(dataSource != null && connection != null){                                                  DataSourceUtils.releaseConnection(connection, dataSource);                          }      }  } |

### org.sky.ssi.ibatis.IBatisTransactionFactory

|  |
| --- |
| package org.sky.ssi.ibatis;    import java.sql.Connection;  import java.util.Properties;    import javax.sql.DataSource;    import org.apache.ibatis.transaction.Transaction;  import org.apache.ibatis.transaction.TransactionFactory;    public class IBatisTransactionFactory implements TransactionFactory{                             private DataSource dataSource;                             public IBatisTransactionFactory(DataSource dataSource){                                                   this.dataSource = dataSource;                           }                             public void setProperties(Properties properties){      }                             public Transaction newTransaction(Connection connection, boolean flag){                                                   return new IBatisTransaction(dataSource,connection,flag);                           }    } |

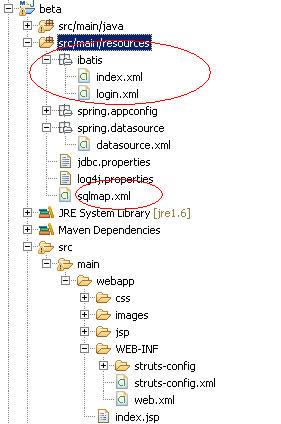
此三个类的作用就是在datasource.xml文件中描述的，把spring与datasource.xml中的datasource和transaction连接起来，此处尤其是“IBatis3SQLSessionFactoryBean”的写法，它通过spring中的“注入”特性，把iBatis的配置注入进spring并委托spring的context来管理iBatis（此属网上没有的资料，全部为本人在历年工程中的经验总结，并且已经在至少3个项目中进行了集成使用与相关测试）。

## 建立iBatis**配置文件**

我们先在****/src/main/resources****目录下建立一个叫sqlmap.xml的文件，内容如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE configuration PUBLIC "-//ibatis.apache.org//DTD Config 3.0//EN" "http://ibatis.apache.org/dtd/ibatis-3-config.dtd">  <configuration>                          <mappers>                                                  <mapper resource="ibatis/index.xml" />                                                  <mapper resource="ibatis/login.xml" />                          </mappers>  </configuration> |

然后我们在****/src/main/resources****目录下建立index.xml与login.xml这2个xml文件。



看到这儿，有人会问了：为什么不把这两个xml文件也建立在spring目录下？

原因很简单：

在datasource.xml文件内我们已经通过

|  |
| --- |
| <bean id="iBatisSessionFactory" class="org.sky.ssi.ibatis.IBatis3SQLSessionFactoryBean" scope="singleton">                                  <property name="configLocation" value="****sqlmap.xml****"></property>                                  <property name="dataSource" ref="dataSource"></property>  </bean> |

这样的方式把iBatis委托给了spring，iBatis的核心就是这个sqlmap.xml文件了，而在这个sqlmap.xml文件已经引用了login.xml与index.xml文件了。

而我们的web.xml文件里有这么一句：

|  |
| --- |
| <context-param>                                  <param-name>contextConfigLocation</param-name>                                  <param-value>****/WEB-INF/classes/spring/\*\*/\*.xml****</param-value>  </context-param> |

因此如果我们再把ibatis/index.xml与ibatis/login.xml再建立到src/main/resources/spring目录下，spring于是会在容器启动时试图加载这两个xml文件，然后一看这两个xml文件不是什么spring的bean，直接抛错，对吧？

****其们等一会再来看login.xml文件与index.xml文件，我们先来搞懂iBatis调用原理.****

## 3.4 iBatis**调用原理**

1）iBatis就是一个dao层，它又被称为sqlmapping，它的sql是书写在一个.xml文件内的，在该xml文件内会将相关的sql绑定到相关的dao类的方法。

2）在调用结束时我们需要在finally块中关闭相关的sql调用。

我们来看一个例子。

### login.xml**文件**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper  PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"  "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">  <mapper namespace="org.sky.ssi.dao.LoginDAO">                          <select id="validLogin" resultType="int" parameterType="****java.util.Map****">                                                  <![CDATA[                                                  SELECT count(1) from t\_login ****where login\_id= #{loginId} and login\_pwd=#{loginPwd}****                                                  ]]>                          </select>  </mapper> |

该DAO指向了一个接口org.sky.ssi.dao.LoginDAO，该dao接受一个sql，并且接受一个Map类型的参数。

那么我们来看该DAO

### LoginDao.java

|  |
| --- |
| package org.sky.ssi.dao;    import java.util.Map;    public interface LoginDAO {                  public int validLogin(****Map<String, Object> paraMap****) throws Exception;  } |

### LoginImpl.java

|  |
| --- |
| package org.sky.ssi.dao.impl;    import java.util.Map;    import org.apache.ibatis.session.SqlSession;  import org.sky.ssi.dao.LoginDAO;  import org.sky.ssi.ibatis.IBatisDAOSupport;  import org.springframework.stereotype.Repository;    @Repository  public class LoginDAOImpl extends IBatisDAOSupport<LoginDAO> implements LoginDAO {                    public int validLogin(Map<String, Object> paraMap) throws Exception {  ****SqlSession session = this.getSqlSession();****                                  try {                                                  return this.getMapper(LoginDAO.class, session).validLogin(paraMap);                                  } catch (Exception e) {                                                  log.error(e.getMessage(), e);                                                  throw new Exception(e);                                  } ****finally {****  ****this.closeSqlSession(session);****  ****}****                  }    } |

很简单吧，一切逻辑都在xml文件内。

一定记得不要忘了在finally块中关闭相关的sql调用啊，要不然将来工程出了OOM的错误不要怪我啊.

## 3.5 index**模块**

### Index.xml**文件**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper  PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"  "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">  <mapper namespace="org.sky.ssi.dao.StudentDAO">                          <select id="getAllStudent" resultType="org.sky.ssi.dbo.StudentDBO">                                                  <![CDATA[                                                  SELECT student\_no studentNo, student\_name studentName from t\_student                                                  ]]>                          </select>                            <update id="addStudent" parameterType="java.util.Map">                                                  insert into t\_student(student\_no, student\_name)values(seq\_student\_no.nextval,#{stdName})                          </update>                            <update id="delStudent" parameterType="java.util.Map">                                                  delete from t\_student where student\_no=#{stdNo}                          </update>  </mapper> |

它指向了StudentDAO这个接口

### StudentDAO.java

|  |
| --- |
| package org.sky.ssi.dao;    import org.sky.ssi.dbo.StudentDBO;  import org.sky.ssi.student.form.\*;  import java.util.\*;    public interface StudentDAO {                            public List<StudentDBO> getAllStudent() throws Exception;                            public void addStudent(Map<String, Object> paraMap) throws Exception;                            public void delStudent(Map<String, Object> paraMap) throws Exception;  } |

### StudentDAOImpl.java

|  |
| --- |
| package org.sky.ssi.dao.impl;    import java.util.List;  import java.util.Map;    import org.apache.commons.logging.Log;  import org.apache.commons.logging.LogFactory;  import org.apache.ibatis.session.SqlSession;  import org.sky.ssi.dao.StudentDAO;  import org.sky.ssi.ibatis.IBatisDAOSupport;  import org.sky.ssi.dbo.StudentDBO;    import org.springframework.stereotype.Repository;    @Repository  public class StudentDAOImpl extends IBatisDAOSupport<StudentDAO> implements StudentDAO {                            @Override                          public List<StudentDBO> getAllStudent() throws Exception {                                                  SqlSession session = this.getSqlSession();                                                  try {                                                                          return this.getMapper(StudentDAO.class, session).getAllStudent();                                                  } catch (Exception e) {                                                                          throw new Exception(e);                                                  } finally {                                                                          this.closeSqlSession(session);                                                  }                          }                            public void addStudent(Map<String, Object> paraMap) throws Exception {                                                  SqlSession session = this.getSqlSession();                                                  try {                                                                          this.getMapper(StudentDAO.class, session).addStudent(paraMap);                                                  } catch (Exception e) {                                                                          throw new Exception(e);                                                  } finally {                                                                          this.closeSqlSession(session);                                                  }                          }                            public void delStudent(Map<String, Object> paraMap) throws Exception {                                                  SqlSession session = this.getSqlSession();                                                  try {                                                                          this.getMapper(StudentDAO.class, session).delStudent(paraMap);                                                  } catch (Exception e) {                                                                          throw new Exception(e);                                                  } finally {                                                                          this.closeSqlSession(session);                                                  }                          }  } |

## 3.6 Service**接口微微有些改变**

为了演示给大家看 iBatis接受多个参数的例子因此我们把原来的如：****login(String loginId, String loginPwd)****这样的方法改成了****public int validLogin(Map<String, Object> paraMap) throws Exception;****这样的结构，请大家注意。

# 四、beta**工程中的增加功能**

## 4.1 **增加了一个filter**

在我们的web.xml文件中

|  |
| --- |
| <filter>                          <filter-name>LoginFilter</filter-name>                          <filter-class>org.sky.ssi.filter.LoginFilter</filter-class>                          <init-param>                                                  <param-name>****exclude****</param-name>                                                  <param-value****>/jsp/login/login.jsp,****  ****/login.do****                                                  </param-value>                          </init-param>  </filter>  <filter-mapping>                          <filter-name>LoginFilter</filter-name>                          <url-pattern>\*.jsp</url-pattern>  </filter-mapping>  <filter-mapping>                          <filter-name>LoginFilter</filter-name>                          <url-pattern>/servlet/\*</url-pattern>  </filter-mapping>  <filter-mapping>                          <filter-name>LoginFilter</filter-name>                          <url-pattern>\*.do</url-pattern>  </filter-mapping> |

有了这个filter我们就不用在我们的web工程中每一个action、每 个jsp里进行“****用户是否登录****”的判断了，它会自动根据配置除去“****exclude****”中的相关web resource，全部走这个“****是否登录****”的判断。

****注意此处这个exclude是笔者自己写的，为什么要exclude？****

**如果你不exclude，试想一个用户在login.jsp中填入相关的登录信息后点一下login按钮跳转到了login.do，而这两个web resource由于没有被“排除”出“需要登录校验”，因此每次你一调用login.jsp, login.do这个filter就都会强制要求你再跳转到login.jsp，那么我们一个用户从login.jsp登录完后再跳回login.jsp再跳回，再跳回，如此重复，进入死循环。**

## 4.2 **增加了一个自动记录异常的日志功能**

在我们的applicationContext.xml文件中

|  |
| --- |
| <bean          id="methodLoggerAdvisor"          class="org.sky.ssi.util.LoggerAdvice" >      </bean>      <aop:config>          <aop:aspect              id="originalBeanAspect"              ref="methodLoggerAdvisor" >              <aop:pointcut                  id="loggerPointCut"                  expression="execution(\* org.sky.ssi.service.impl.\*.\*(..))" />              <aop:around                  method="aroundAdvice"                  pointcut-ref="loggerPointCut" />          </aop:aspect>      </aop:config> |

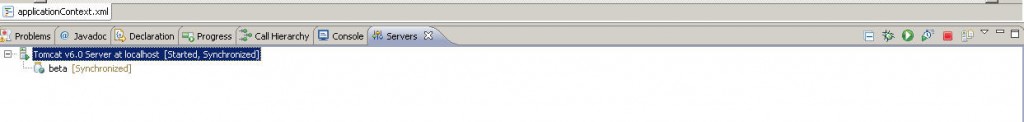
这样，我们的dao层、service层、有错就只管往外throw，框架一方面在接到相关的exception会进行数据库事务的自动回滚外，还会自动把service层抛出的exception记录在log文件中。

# 五、测试我们的工程

确认我们的****StudentServiceImpl****中删除学生的****delStudent****方法内容如下：

|  |
| --- |
| ****public** **void**** delStudent(String[] stdNo) ****throws**** Exception {  ****for**** (String s : stdNo) {                       Map<String, Object> paraMap = ****new**** HashMap<String, Object>();                       paraMap.put("stdNo", s);                       studentDAO.delStudent(paraMap);                       throw new Exception("force system to throw a exception");                }  } |

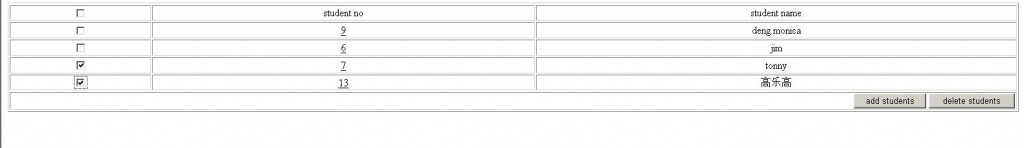
我们把beta工程添加入我们在eclipse中配好的j2eeserver中去并启动起来。



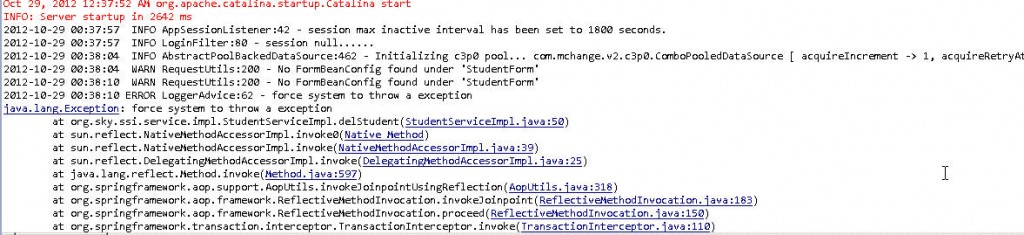
在IE中输入：[http://localhost:8080/beta/index.do](http://localhost:8080/beta/index.do" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)。 系统直接跳到login界面



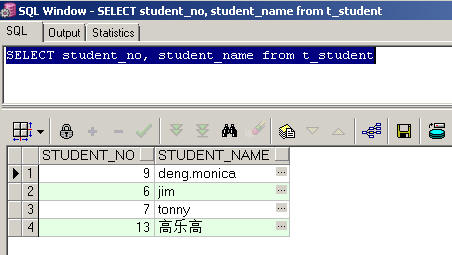
我们输入相关的用户名写密码。



我们选中“13号学生高乐高”与“9号学生”，点“deletestudent”按钮。



后台抛错了，查看数据库内的数据



数据还在，说明我们的iBatis的事务已经在spring中启作用了.

再次更改****StudentServiceImpl.java****类中的****delStudent****方法，把“****throw new Exception("force system to throw a exception****");”注释掉，再来运行



我们再次选 中9号和13号学生，点deletestudent按钮，删除成功，****这个够13的人终于被删了****，呵呵。