知识点列表

编号	名称	描述	级别
1	AOP 注解配置	掌握在 XML 中使用 AOP 注解同在切面组件中使用 Aop 注解	**
2	Spring 对数据库访问技术的支持	掌握 Spring 整合 JDBC 的案例 ,掌握 Spring 整合 Hibernate 的案例	**

注：　"*"理解级别　"**"掌握级别　"***"应用级别

目录

1. AOP 注解配置 **

1) 在 xml 配置中启用 AoP 注解配置
   - ✓ <aop:aspectj-autoproxy/>
2) 在切面组件中使用 Aop 注解
   - ✓ @Aspect
   - ✓ @Pointcut
   - ✓ @Before、@After、@AfterReturing、@AfterThrowing、@Around

【案例 1】AOP 注解方式 **

1) 新建工程 spring3

2) 导入 jar 包

3) XML 形式

a. 新建 UserDAO

```
package tarena.dao;

public interface UserDAO {
	public void save();
	public void update();
}
```

b. 新建 JDBCUserDAO

```
package tarena.dao;

public class JDBCUserDAO implements UserDAO {

	public void save() {
		System.out.println("DAO##采用JDBC技术保存用户信息！");

	}

	public void update() {
		System.out.println("DAO##采用JDBC技术更新用户信息！");

	}
}
```

c.　新建 UserService

```
package tarena.service;

public interface UserService {
	public void regist();
	public void update();
}
```

d.　新建 UserServiceImpl

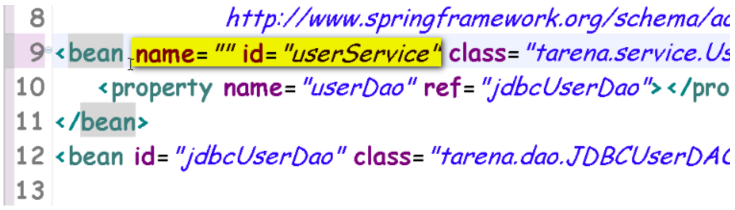
```
package tarena.service;

import tarena.dao.UserDAO;

public class UserServiceImpl implements UserService{
	private UserDAO userDao;

	public void setUserDao(UserDAO userDao) {
		this.userDao = userDao;
	}

	public void regist() {
		System.out.println("Service##用户注册处理");
		userDao.save();
	}

	public void update() {
		System.out.println("Service##用户修改个人信息处理");
		userDao.update();
	}
}
```

e.　新建 schema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

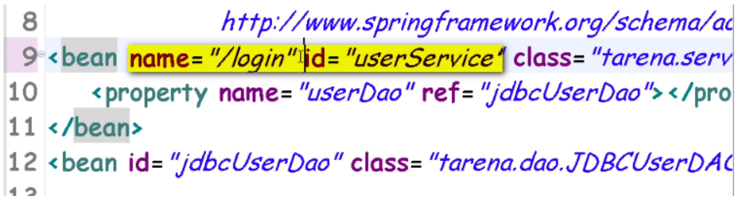
```
          xmlns:context="http://www.springframework.org/schema/context"
          xmlns:aop="http://www.springframework.org/schema/aop"
          xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

     <bean id="userService" class="tarena.service.UserServiceImpl">
          <property name="userDao" ref="jdbcUserDao"></property>
     </bean>
     <bean id="jdbcUserDao" class="tarena.dao.JDBCUserDAO"></bean>

</beans>
```

name 属性和 id 属性作用是一样的



```
                    http://www.springframework.org/schema/ac
<bean name="" id="userService" class="tarena.service.Us
     <property name="userDao" ref="jdbcUserDao"></pro
</bean>
<bean id="jdbcUserDao" class="tarena.dao.JDBCUserDA(
```

如果出现类似“/login”这样的特殊字符，就必须使用 name 属性，id 是不可以的，简言之，name 属性比 id 属性强大之处是可以使用特殊字符。



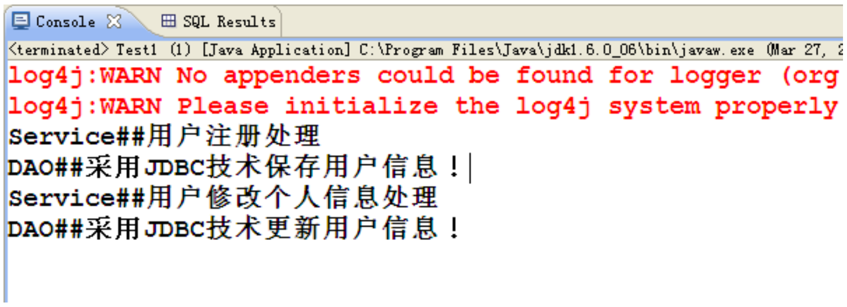
```
                    http://www.springframework.org/schema/ac
<bean name="/login" id="userService" class="tarena.serv
     <property name="userDao" ref="jdbcUserDao"></pro
</bean>
<bean id="jdbcUserDao" class="tarena.dao.JDBCUserDA(
```

f.   新建 Test1

```
package tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import tarena.service.UserService;
```

```
public class Test1 {

	public static void main(String[] args) {
		ApplicationContext ac =
			new ClassPathXmlApplicationContext("schema.xml");
		UserService userService = (UserService)ac.getBean("userService");
		userService.regist();
		userService.update();
	}
}
```

g. 运行 Test1



```
Console ☒   SQL Results
<terminated> Test1 (1) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Service##用户注册处理
DAO##采用JDBC技术保存用户信息！
Service##用户修改个人信息处理
DAO##采用JDBC技术更新用户信息！
```

测试成功

接下来追加日志记录功能

h. 拷贝 aop.OptLogger

```
package tarena.aop;

import org.aspectj.lang.ProceedingJoinPoint;
import tarena.util.PropertiesUtil;

/**
 * 切面组件，记录操作日志
 * @author tarena
 *
 */
public class OptLogger {
	public Object logger(ProceedingJoinPoint pjp)
	throws Throwable{
		Object obj = pjp.proceed();//执行目标对象的功能

		String methodName = pjp.getSignature().getName();
		String clazzName = pjp.getTarget().getClass().getName();

		PropertiesUtil.getInstance("opt.properties");

		String key = clazzName+"."+methodName;

		System.out.println(
				"执行了"+PropertiesUtil.getProperty(key));
		return obj;
	}
}
```

i.	拷贝 util.PropertiesUtil

```
package tarena.util;

import java.io.IOException;
import java.util.Properties;

public class PropertiesUtil {

	static Properties props = new Properties();

	private PropertiesUtil(){}

	public static Properties getInstance(String path)
	throws IOException{
		props.load(PropertiesUtil.class.getClassLoader()
					.getResourceAsStream(path));
		return props;
	}

	public static String getProperty(String key){
		String val = "";
```

```
                if(props != null){
                        String prop = props.getProperty(key);
                        if(prop != null){
                                val = prop;
                        }
                }
                return val;
        }
}
```

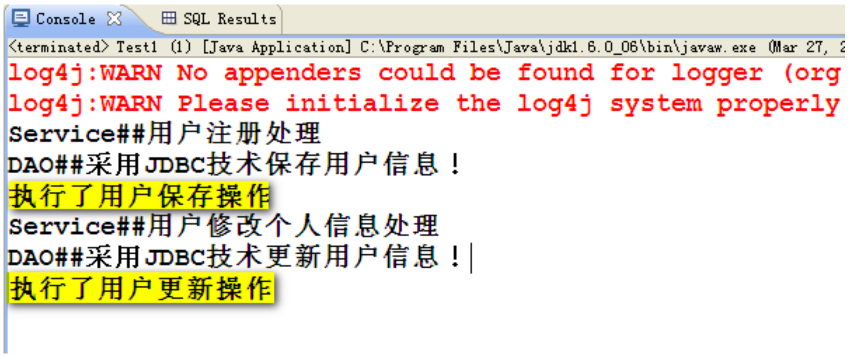
j. 新建 opt.properties

```
tarena.service.UserServiceImpl.update=\u7528\u6237\u66F4\u65B0\u64CD\u4F5C
tarena.service.UserServiceImpl.regist=\u7528\u6237\u4FDD\u5B58\u64CD\u4F5C
```

k. 修改 schema.xml

```
        <bean id="userService" class="tarena.service.UserServiceImpl">
                <property name="userDao" ref="jdbcUserDao"></property>
        </bean>
        <bean id="jdbcUserDao" class="tarena.dao.JDBCUserDAO"></bean>

        <bean id="optLogger" class="tarena.aop.OptLogger"></bean>

        <aop:config>
                <aop:pointcut expression="within(tarena.service..*)"
                id="servicePointcut"/>

                <aop:aspect id="optLoggerAspect" ref="optLogger">
                        <aop:around method="logger"
                        pointcut-ref="servicePointcut"/>
                </aop:aspect>
        </aop:config>

</beans>
```

l. 运行 Test



```
<terminated> Test1 (1) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Service##用户注册处理
DAO##采用JDBC技术保存用户信息！
执行了用户保存操作
Service##用户修改个人信息处理
DAO##采用JDBC技术更新用户信息！
执行了用户更新操作
```

4) 注解方式

a. 新建 annotation.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:context="http://www.springframework.org/schema/context"
           xmlns:aop="http://www.springframework.org/schema/aop"
           xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

     <context:component-scan base-package="tarena">
     </context:component-scan>
</beans>
```

b. 修改 UserServiceImpl

增加注解

```java
package tarena.service;

import org.springframework.stereotype.Service;
import tarena.dao.UserDAO;

@Service("userService")
public class UserServiceImpl implements UserService{
	private UserDAO userDao;

	public void setUserDao(UserDAO userDao) {
		this.userDao = userDao;}


	public void regist() {
		System.out.println("Service##用户注册处理");
		userDao.save();
	}

	public void update() {
		System.out.println("Service##用户修改个人信息处理");
		userDao.update();
	}

}
```

c.	修改 JDBCUserDAO

增加注解

```
package tarena.dao;

import org.springframework.stereotype.Repository;

@Repository("jdbcUserDao")
public class JDBCUserDAO implements UserDAO {

	public void save() {
		System.out.println("DAO##采用JDBC技术保存用户信息！");

	}

	public void update() {
		System.out.println("DAO##采用JDBC技术更新用户信息！");

	}
}
```

d. 修改 OptLogger

增加注解，使用@Component 即可

```
package tarena.aop;

/**
 * 切面组件，记录操作日志
 * @author tarena
 *
 */
@Component("optLogger")
public class OptLogger {
	public Object logger(ProceedingJoinPoint pjp)
	throws Throwable{
		Object obj = pjp.proceed();//执行目标对象的功能

		String methodName = pjp.getSignature().getName();
		String clazzName = pjp.getTarget().getClass().getName();

		PropertiesUtil.getInstance("opt.properties");

		String key = clazzName+"."+methodName;

		System.out.println(
				"执行了"+PropertiesUtil.getProperty(key));
		return obj;
	}
}
```

如上，共追加了 3 个注解，使用注解方式添加 bean 组件

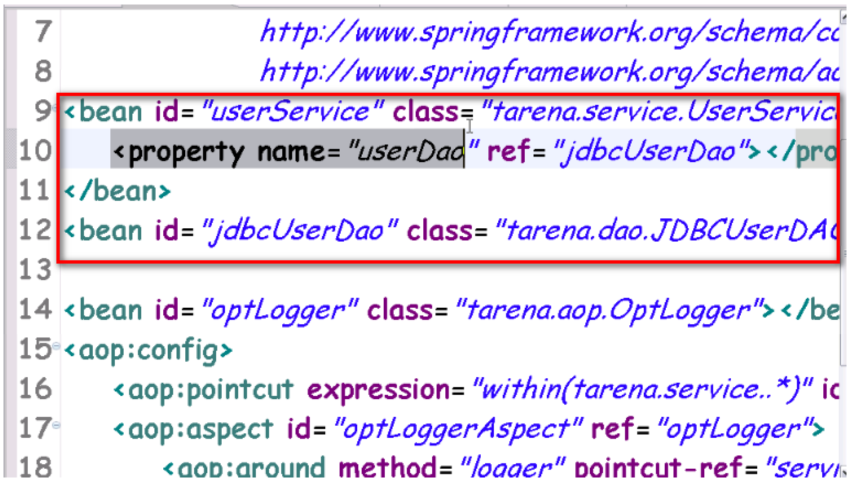
接下来，将 DAO 注入给 Service，有两种方式@Resource 和@Autowired，我们这里使用@Resource

e. 修改 UserServiceImpl

```
package tarena.service;
```

```
import javax.annotation.Resource;
import org.springframework.stereotype.Service;
import tarena.dao.UserDAO;

@Service("userService")
public class UserServiceImpl implements UserService{
	private UserDAO userDao;

	@Resource(name="jdbcUserDao")
	public void setUserDao(UserDAO userDao) {
		this.userDao = userDao;}


	public void regist() {
		System.out.println("Service##用户注册处理");
		userDao.save();
	}

	public void update() {
		System.out.println("Service##用户修改个人信息处理");
		userDao.update();
	}
}
```

如上步骤 a--步骤 e 的操作，相当于完成了创建<bean>和为<bean>添加<property>的功能，如下所示：



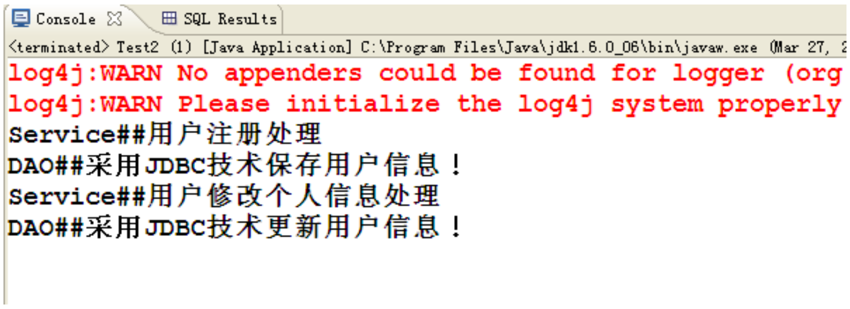
```
                    http://www.springframework.org/schema/cc
                    http://www.springframework.org/schema/ac
<bean id="userService" class="tarena.service.UserServic
    <property name="userDao" ref="jdbcUserDao"></pro
</bean>
<bean id="jdbcUserDao" class="tarena.dao.JDBCUserDA

<bean id="optLogger" class="tarena.aop.OptLogger"></be
<aop:config>
    <aop:pointcut expression="within(tarena.service..*)" ic
    <aop:aspect id="optLoggerAspect" ref="optLogger">
        <aop:around method="logger" pointcut-ref="servi
```

步骤 a-步骤 e，我们已经完成了 IoC，先测试一下

f.　新建 Test2

```
package tarena.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import tarena.service.UserService;

public class Test2 {

    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext("annotation.xml");
        UserService userService =
            (UserService)ac.getBean("userService");
        userService.regist();
        userService.update();
    }
}
```

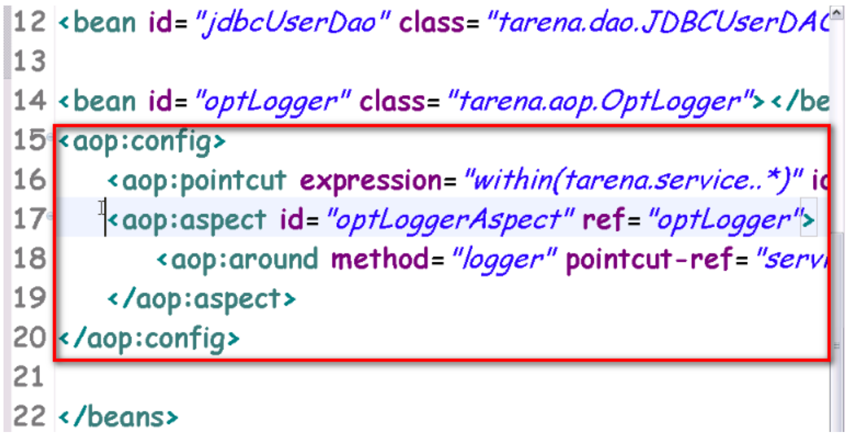
g.　运行 Test2



```
<terminated> Test2 (1) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Service##用户注册处理
DAO##采用JDBC技术保存用户信息！
Service##用户修改个人信息处理
DAO##采用JDBC技术更新用户信息！
```

测试成功，IoC 功能完成

接下来添加注解实现的 AOP 功能，替代如下代码


```
<bean id="jdbcUserDao" class="tarena.dao.JDBCUserDAO
 
<bean id="optLogger" class="tarena.aop.OptLogger"></be
<aop:config>
    <aop:pointcut expression="within(tarena.service..*)" id
    <aop:aspect id="optLoggerAspect" ref="optLogger">
        <aop:around method="logger" pointcut-ref="servi
    </aop:aspect>
</aop:config>
 
</beans>
```

h.　修改 OptLogger

@Aspect 表示该组件是 AOP 组件

@Pointcut("within(tarena.service..*)")　用于指定切入点表达式

@Around("servicePointcut()")　　为指定的目标方法设置环绕通知处理

```
package tarena.aop;

/**
 * 切面组件，记录操作日志
 * @author tarena
 *
 */
@Component("optLogger")
@Aspect
public class OptLogger {

	@Pointcut("within(tarena.service..*)")
	public void servicePointcut(){}

	//环绕通知处理
	@Around("servicePointcut()")
	public Object logger(ProceedingJoinPoint pjp) throws Throwable{
		System.out.println("-------");
		Object obj = pjp.proceed();//执行目标对象的功能
		String methodName = pjp.getSignature().getName();
		String clazzName = pjp.getTarget().getClass().getName();
		PropertiesUtil.getInstance("opt.properties");
```

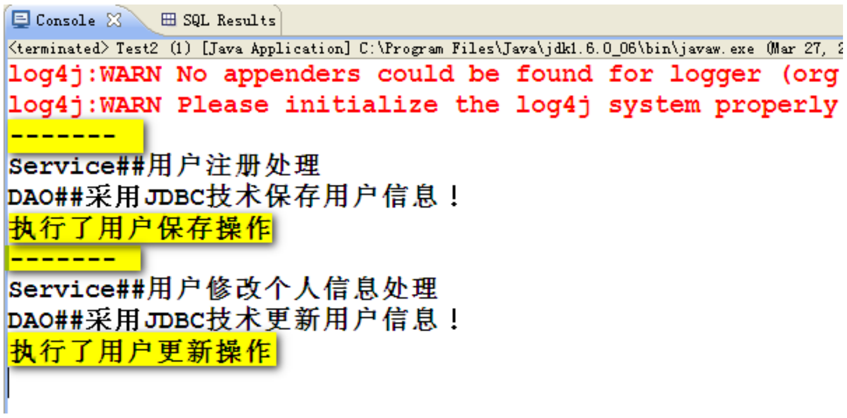
```
            String key = clazzName+"."+methodName;
            System.out.println("执行了"+PropertiesUtil.getProperty(key));
            return obj;
        }
}
```

注意：我们只能在类、方法、属性前加注解

i.　修改 annotation.xml

```
        <context:component-scan base-package="tarena">
        </context:component-scan>
        <aop:aspectj-autoproxy/>
</beans>
```

j.　运行 Test2



```
<terminated> Test2 (1) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
--------
Service##用户注册处理
DAO##采用JDBC技术保存用户信息！
执行了用户保存操作
--------
Service##用户修改个人信息处理
DAO##采用JDBC技术更新用户信息！
执行了用户更新操作
```

注意：因为 OptLogger 中设置为@Around("servicePointcut()")，所以介货是个环绕的

（案例结束）

2.　Spring 对数据库访问技术的支持　**

1)　对 DAO 提供了以下支持

- ✓　一致的异常处理 DataAccessException
- ✓　一致的 DAO 抽象类 DaoSupport、Template

2)　整合 JDBC

a.　使用的 API

➢　JdbcDaoSupport

用于提供编写 DAO 组件的支持

- JdbcTemplate
  用于完成增删改查操作
- update()
  增删改操作
- query()、queryForObject()、queryForInt 等
  查询操作
- execute()
  其他语句，例如建表、修改表结构语句
- 其他操作（了解）
  批处理、返回自动增长主键值

b. XML 配置
首先定义连接池<bean id="dataSource"/>之后将 dataSource 注入给所有 DAO 组件

3) 整合 Hibernate

a. 使用的 API

- HibernateDaoSupport
  提供编写 DAO 组件的支持
- HibernateTemplate
  提供了增删改查操作
- save()：保存
- update()：更新
- delete()：删除
- find()：查询
- 如果需要分页查询，可以使用 HibernateDaoSupport 提供的 this.getSession()方法获取 Session 对象。

b. XML 配置

- 首先配置连接池 dataSource
- 其次配置 SessionFactory
- 最后将 SessionFactory 注入给所有 DAO 组件

【案例 2】Spring 整合 JDBC **

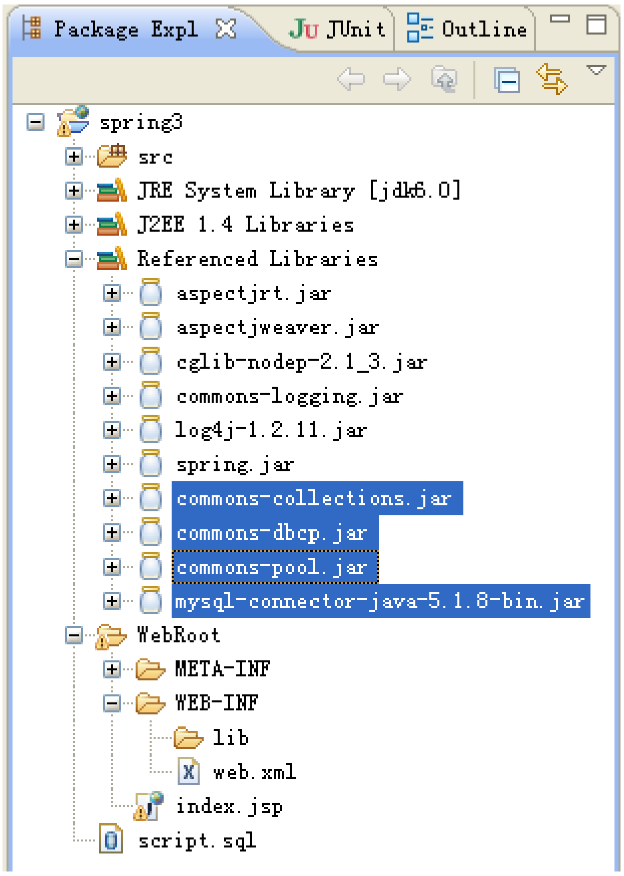
1) 使用 spring3 工程

请下载 spring3.zip（含 Jar 包）

2) 导入 Jar 包

✓ 数据库 Jar 包
✓ commons-dbcp.jar 连接池

- ✓ commons-collections.jar　　dbcp 需要的 commons 包
- ✓ commons-pool.jar　　dbcp 需要的 commons 包



```
spring3
  src
  JRE System Library [jdk6.0]
  J2EE 1.4 Libraries
  Referenced Libraries
    aspectjrt.jar
    aspectjweaver.jar
    cglib-nodep-2.1_3.jar
    commons-logging.jar
    log4j-1.2.11.jar
    spring.jar
    commons-collections.jar
    commons-dbcp.jar
    commons-pool.jar
    mysql-connector-java-5.1.8-bin.jar
  WebRoot
    META-INF
    WEB-INF
      lib
      web.xml
    index.jsp
  script.sql
```

连接池的概念

连接池组件是干什么的？

连接池组件中管理的单元就是我们之前使用的 Connection 对象，在连接池中可以管理 Connection 对象的创建和销毁，除此之外，连接池还可以控制和管理 Connection 对象的数量。

连接池组件的优势：可以提高程序的稳定性；可以灵活的控制访问的连接数量

3)　新建数据库表

```
DROP TABLE IF EXISTS d_user;
CREATE TABLE d_user (
  id int(12) NOT NULL auto_increment,
  email varchar(50) NOT NULL,
  nickname varchar(50) default NULL,
  password varchar(50) NOT NULL,
   user_integral int(12) NOT NULL default '0',
   is_email_verify char(3),
   email_verify_code varchar(50) default NULL,
   last_login_time bigint default NULL,
   last_login_ip varchar(15) default NULL,
   PRIMARY KEY   (id),
   UNIQUE KEY email (email)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4) 在 schema.xml 中配置连接池

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:context="http://www.springframework.org/schema/context"
           xmlns:aop="http://www.springframework.org/schema/aop"
           xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

      <bean id="dataSource" destroy-method="close"
                class="org.apache.commons.dbcp.BasicDataSource">
          <property name="driverClassName"
               value="com.mysql.jdbc.Driver"></property>
          <property name="url"
               value="jdbc:mysql://localhost:3306/test"></property>
          <property name="username" value="root"></property>
          <property name="password" value="root"></property>
          <property name="maxActive" value="10"></property>
          <property name="initialSize" value="2"></property>
          <property name="minIdle" value="2"></property>
          <property name="maxIdle" value="3"></property>
     </bean>

     <bean id="userService" class="tarena.service.UserServiceImpl">
          <property name="userDao" ref="jdbcUserDao"></property>
```

```
    </bean>
    <bean id="jdbcUserDao" class="tarena.dao.JDBCUserDAO"></bean>

    <bean id="optLogger" class="tarena.aop.OptLogger"></bean>

    <aop:config>
        <aop:pointcut expression="within(tarena.service..*)"
        id="servicePointcut"/>
        <aop:aspect id="optLoggerAspect" ref="optLogger">
            <aop:around method="logger"
            pointcut-ref="servicePointcut"/>
        </aop:aspect>
    </aop:config>

</beans>
```

- ✓ destroy-method="close"

  指定销毁 dataSource 的方法，不是必须的，如果指定，能回收及时些
- ✓ <property name="initialSize" value="2">

  表示连接池创建后，初始时有 2 个 Connection
- ✓ <property name="maxActive" value="10">

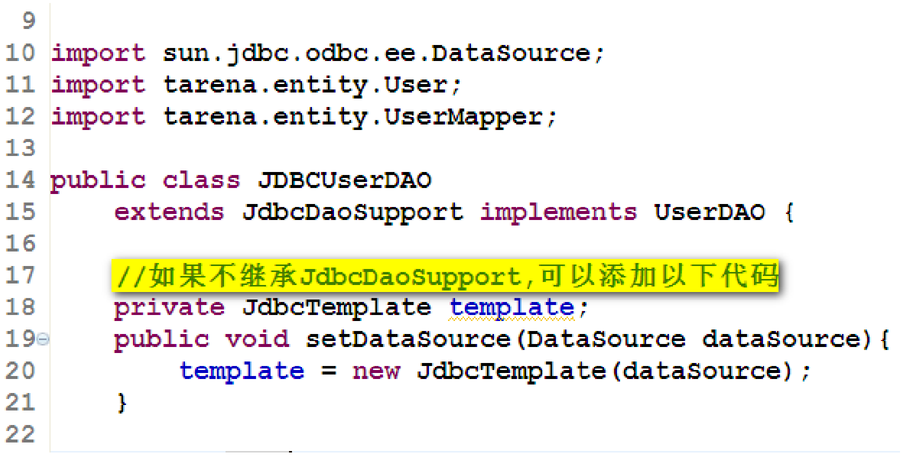
  在连接池中最大创建 10 个 Connection
- ✓ <property name="minIdle" value="2">

  表示最小的空闲数量，用于控制空闲的 Connection 的数量，表示最小空闲数量不能低于 2 个
- ✓ <property name="maxIdle" value="3">

  表示最大的空闲数量，表示最大空闲数量不能超过 3 个

如何使用 dataSource？采用注入的方式（推荐使用 set 方式注入）

其一，我们可以这样使用 set 方式注入（底层就是这么实现的）



```java

import sun.jdbc.odbc.ee.DataSource;
import tarena.entity.User;
import tarena.entity.UserMapper;

public class JDBCUserDAO
    extends JdbcDaoSupport implements UserDAO {

    //如果不继承JdbcDaoSupport,可以添加以下代码
    private JdbcTemplate template;
    public void setDataSource(DataSource dataSource){
        template = new JdbcTemplate(dataSource);
    }

```

其二，Spring 框架的好处就在于为我们提供了一些工具类，JdbcDaoSupport 可以帮助我们完成 dataSource 的注入。

5) 修改 JdbcUserDAO

将连接池注入到 JDBCUserDAO 中，只要这样继承 JdbcDaoSupport 即可。

```java
package tarena.dao;

import org.springframework.jdbc.core.support.JdbcDaoSupport;

public class JDBCUserDAO
    extends JdbcDaoSupport implements UserDAO {

    public void save() {
        System.out.println("DAO##采用JDBC技术保存用户信息！");
    }

    public void update() {
        System.out.println("DAO##采用JDBC技术更新用户信息！");
    }
}
```

6) 配置 schema.xml

```xml
    <bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver"></property>
        <property name="url"
```

```
                value="jdbc:mysql://localhost:3306/test"></property>
            <property name="username" value="root"></property>
            <property name="password" value="root"></property>
            <property name="maxActive" value="10"></property>
            <property name="initialSize" value="2"></property>
            <property name="minIdle" value="2"></property>
            <property name="maxIdle" value="3"></property>
        </bean>

        <bean id="userService" class="tarena.service.UserServiceImpl">
            <property name="userDao" ref="jdbcUserDao"></property>
        </bean>

        <bean id="jdbcUserDao" class="tarena.dao.JDBCUserDAO">
            <!--注意：只能写dataSource，别的名字不行-->
            <property name="dataSource" ref="dataSource"></property>
        </bean>

        <bean id="optLogger" class="tarena.aop.OptLogger"></bean>

        <aop:config>
            <aop:pointcut expression="within(tarena.service..*)"
            id="servicePointcut"/>

            <aop:aspect id="optLoggerAspect" ref="optLogger">
                <aop:around method="logger"
                pointcut-ref="servicePointcut"/>
            </aop:aspect>
        </aop:config>
```

7) 新建 User

```
package tarena.entity;

public class User implements java.io.Serializable {

    // Fields
    private Integer id;
    private String email = "";
    private String nickname = "";
    private String password = "";
    private Integer userIntegral = 0;
    private boolean emailVerify = false;
    private String emailVerifyCode = "";
    private long lastLoginTime = 0L;
    private String lastLoginIp = "";

    // Constructors
    /** default constructor */
    public User() {}

    /** minimal constructor */
    public User(String email, String password,
                Integer userIntegral) {
          this.email = email;
          this.password = password;
          this.userIntegral = userIntegral;
    }

    public boolean isEmailVerify() {
          return emailVerify;}
    public void setEmailVerify(boolean emailVerify) {
          this.emailVerify = emailVerify;}
    public Integer getId() {
          return this.id;}
    public void setId(Integer id) {
          this.id = id;}
    public String getEmail() {
          return this.email;}
    public void setEmail(String email) {
          this.email = email;}
    public String getNickname() {
          return this.nickname;}
    public void setNickname(String nickname) {
          this.nickname = nickname;}
    public String getPassword() {
          return this.password;}
```

```
        public void setPassword(String password) {
                this.password = password;}
        public Integer getUserIntegral() {
                return this.userIntegral;}
        public void setUserIntegral(Integer userIntegral) {
                this.userIntegral = userIntegral;}
        public String getEmailVerifyCode() {
                return this.emailVerifyCode;}
        public void setEmailVerifyCode(String emailVerifyCode) {
                this.emailVerifyCode = emailVerifyCode;}
        public long getLastLoginTime() {
                return this.lastLoginTime;}
        public void setLastLoginTime(long lastLoginTime) {
                this.lastLoginTime = lastLoginTime;}
        public String getLastLoginIp() {
                return this.lastLoginIp;}
        public void setLastLoginIp(String lastLoginIp) {
                this.lastLoginIp = lastLoginIp;}
}
```

8) 修改 UserDao

```
package tarena.dao;

import java.util.List;
import tarena.entity.User;

public interface UserDAO {
        public void save(User user);
        public void update(User user);
        public void deleteById(int id);
}
```

如果 JdbcUserDAO 继承了 org.springframework.jdbc.core.support.JdbcDaoSupport，那么 Spring 框架会提供一些便利的方法，可以直接调用

9) 修改 JdbcUserDao

加入插入和删除 User 的方法

```
package tarena.dao;

```

```java
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import tarena.entity.User;

public class JDBCUserDAO
    extends JdbcDaoSupport implements UserDAO {

    public void save(User user) {
        System.out.println("采用JDBC技术保存用户信息！");
        String sql = "insert into d_user " +
                "(email,nickname,password," +
                "user_integral,is_email_verify," +
                "email_verify_code,last_login_time,last_login_ip) " +
                "values (?,?,?,?,?,?,?,?)";

        this.getJdbcTemplate()
            .update(sql,
                    new Object[]{user.getEmail(),
                                user.getNickname(),
                                user.getPassword(),
                                user.getUserIntegral(),
                                user.isEmailVerify()?"Y":"N",
                                user.getEmailVerifyCode(),
                                user.getLastLoginTime(),
                                user.getLastLoginIp()});
    }

    public void update(User user) {
        System.out.println("采用JDBC技术更新用户信息！");
        String sql = "update d_user set email=?," +
                "nickname=?," +
                "password=?," +
                "user_integral=?," +
                "is_email_verify=?," +
                "email_verify_code=?," +
                "last_login_time=?," +
                "last_login_ip=? " +
                "where id=?";
```

```
          this.getJdbcTemplate()
               .update(sql,
                         new Object[]{user.getEmail(),
                                        user.getNickname(),
                                        user.getPassword(),
                                        user.getUserIntegral(),
                                        user.isEmailVerify()?"Y":"N",
                                        user.getEmailVerifyCode(),
                                        user.getLastLoginTime(),
                                        user.getLastLoginIp(),
                                        user.getId()});
     }

     public void deleteById(int id) {
          String sql = "delete from d_user where id=?";
          this.getJdbcTemplate()
               .update(sql,new Object[]{id});
     }
}
```

✓ user.isEmailVerify()?"Y":"N"　利用三目运算符设置为想要类型的字符串。

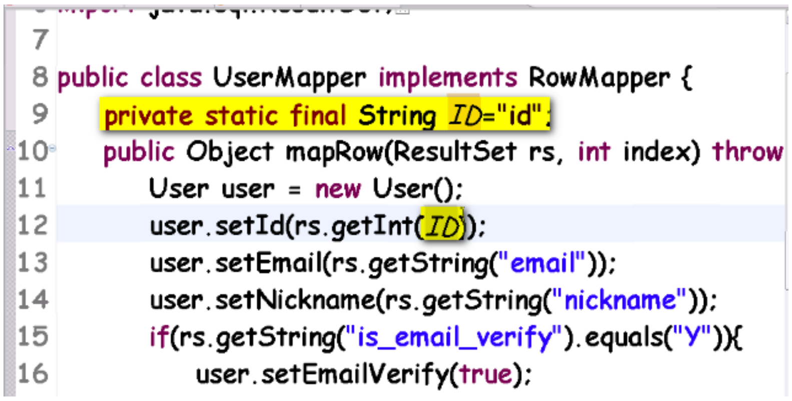
如果我们还想加入一些查询方法，比如 findAll()，怎么做？
我们需要定义一个映射类 UserMapper，完成结果集中的字段值与 User 属性之间的映射关系

10) 新建 UserMapper

UserMapper 实现 RowMapper 接口，覆盖方法 mapRow 完成从结果集中解析出结果做操作

```
package tarena.entity;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class UserMapper implements RowMapper {

     public Object mapRow(ResultSet rs, int index)
     throws SQLException {

        User user = new User();

        user.setId(rs.getInt("id"));

        user.setEmail(rs.getString("email"));

        user.setNickname(rs.getString("nickname"));

        if(rs.getString("is_email_verify").equals("Y")){
            user.setEmailVerify(true);
        }else{
            user.setEmailVerify(false);
        }

        user.setEmailVerifyCode(
                rs.getString("email_verify_code"));

        user.setLastLoginTime(
                rs.getLong("last_login_time"));

        return user;
    }
}
```

其一，代码是简便写法，一般在企业开发中，这些都定义为 final 的，我们这样写

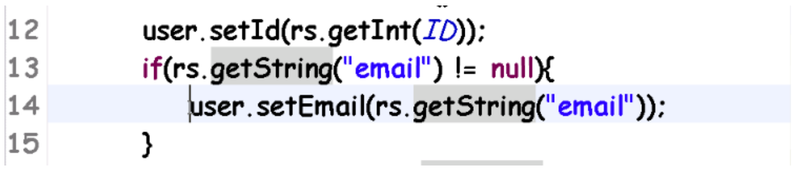


```
public class UserMapper implements RowMapper {
    private static final String ID="id";
    public Object mapRow(ResultSet rs, int index) throw
        User user = new User();
        user.setId(rs.getInt(ID));
        user.setEmail(rs.getString("email"));
        user.setNickname(rs.getString("nickname"));
        if(rs.getString("is_email_verify").equals("Y")){
            user.setEmailVerify(true);
```

其二，一般情况下，我们也不这样写

user.setEmail(rs.getString("xxx"))

为了防止 rs.getString(XXX)取出 null 值后直接放入 user，我们加一个 if 判断



```
        user.setId(rs.getInt(ID));
        if(rs.getString("email") != null){
            user.setEmail(rs.getString("email"));
        }
```

11) 修改 UserDao

```
package tarena.dao;

import java.util.List;

import tarena.entity.User;

public interface UserDAO {
	public void save(User user);
	public void update(User user);
	public void deleteById(int id);
	public User findById(int id);
	public List<User> findAll();
	public int   count();
}
```

12) 修改 JdbcUserDao

再添加 3 个方法

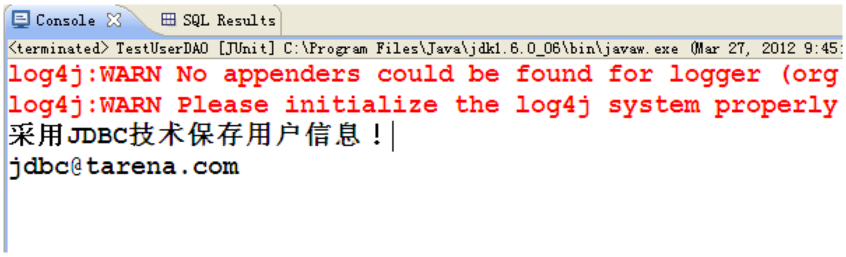
代码片段

```
	public User findById(int id) {
		String sql = "select * from d_user where id=?";
		return (User)this.getJdbcTemplate()
						.queryForObject(
							sql,
							new Object[]{id},
							new UserMapper());
	}

	public List<User> findAll() {
		String sql = "select * from d_user";
		List list = this.getJdbcTemplate().query(sql, new UserMapper());
		return list;
	}

	public int count() {
		String sql = "select count(*) from d_user";
		return this.getJdbcTemplate().queryForInt(sql);
	}
```

13) 新建 TestUserDAO

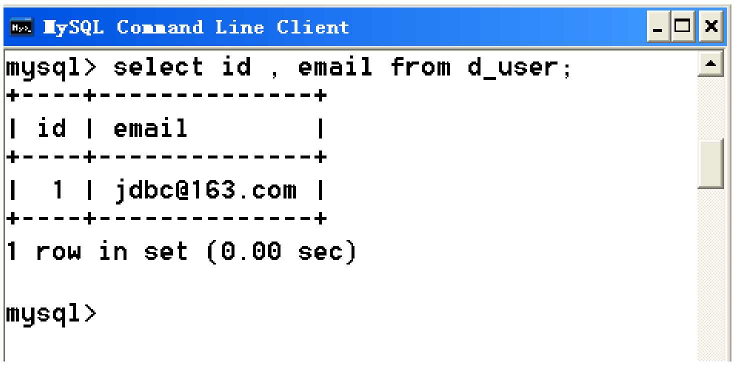
```
package tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import tarena.dao.UserDAO;
import tarena.entity.User;

public class TestUserDAO {
	@Test
	public void testAdd(){
		User user = new User();
		user.setEmail("jdbc@163.com");
		user.setNickname("jdbc");
		user.setPassword("1111");
		user.setUserIntegral(10);
		user.setEmailVerify(false);
		user.setEmailVerifyCode("asdfasdew");
		user.setLastLoginTime(System.currentTimeMillis());
		user.setLastLoginIp("192.168.2.1");

		ApplicationContext ac =
			new ClassPathXmlApplicationContext("schema.xml");
		UserDAO userDao = (UserDAO)ac.getBean("jdbcUserDao");
		userDao.save(user);

		User user1 = userDao.findById(1);
		System.out.println(user1.getEmail());
	}
}
```

14) 运行 TestUerDAO



```
<terminated> TestUserDAO [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012 9:45:
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
采用JDBC技术保存用户信息！
jdbc@tarena.com
```

查看数据库



```
mysql> select id , email from d_user;
+----+----------------+
| id | email          |
+----+----------------+
|  1 | jdbc@163.com   |
+----+----------------+
1 row in set (0.00 sec)

mysql>
```

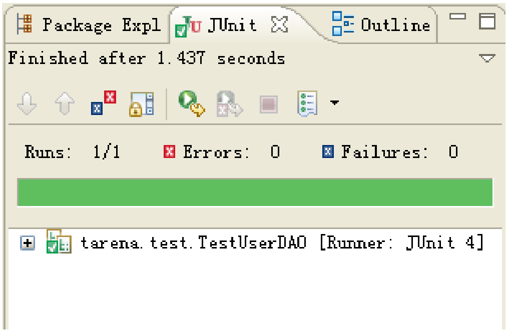
我们可以采用 JUnit 提供的断言来进行单元测试

15) 修改 TestUserDAO

采用断言的优势在于

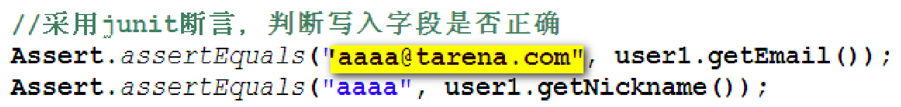
```java
package tarena.test;

import org.junit.Assert;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import tarena.dao.UserDAO;
import tarena.entity.User;

public class TestUserDAO {
	@Test
	public void testAdd(){
		User user = new User();
		user.setEmail("jdbc@tarena.com");
		user.setNickname("jdbc");
		user.setPassword("1111");
		user.setUserIntegral(10);
		user.setEmailVerify(false);
		user.setEmailVerifyCode("asdfasdew");
		user.setLastLoginTime(System.currentTimeMillis());
		user.setLastLoginIp("192.168.2.1");

		ApplicationContext ac =
			new ClassPathXmlApplicationContext("schema.xml");
		UserDAO userDao = (UserDAO)ac.getBean("jdbcUserDao");

//		userDao.save(user);

		User user1 = userDao.findById(1);
//		System.out.println(user1.getEmail());

		//采用junit断言，判断写入字段是否正确
		Assert.assertEquals("jdbc@tarena.com", user1.getEmail());
		Assert.assertEquals("jdbc", user1.getNickname());
	}
}
```

16) 运行 TestUserDAO



```
Finished after 1.437 seconds

Runs:  1/1    Errors:  0    Failures:  0

tarena.test.TestUserDAO [Runner: JUnit 4]
```

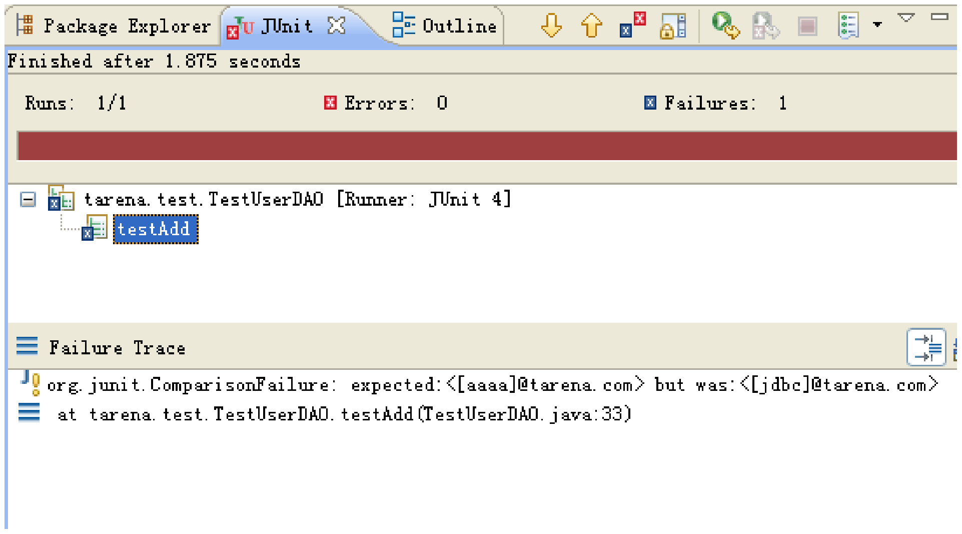
如果我们这样，测试的值不正确



```
//采用junit断言，判断写入字段是否正确
Assert.assertEquals("aaaa@tarena.com", user1.getEmail());
Assert.assertEquals("aaaa", user1.getNickname());
```

控制台提示

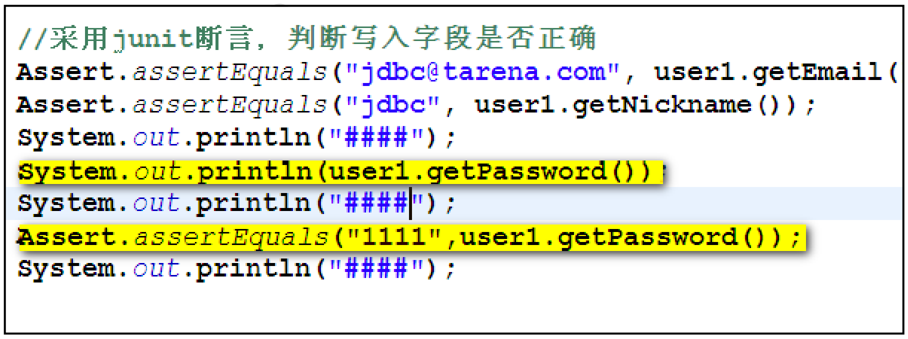
“Failure Trace”提示，值不相同



```
Finished after 1.875 seconds
Runs:  1/1          Errors:  0          Failures:  1

tarena.test.TestUserDAO [Runner: JUnit 4]
    testAdd

Failure Trace
org.junit.ComparisonFailure: expected:<[aaaa]@tarena.com> but was:<[jdbc]@tarena.com>
at tarena.test.TestUserDAO.testAdd(TestUserDAO.java:33)
```

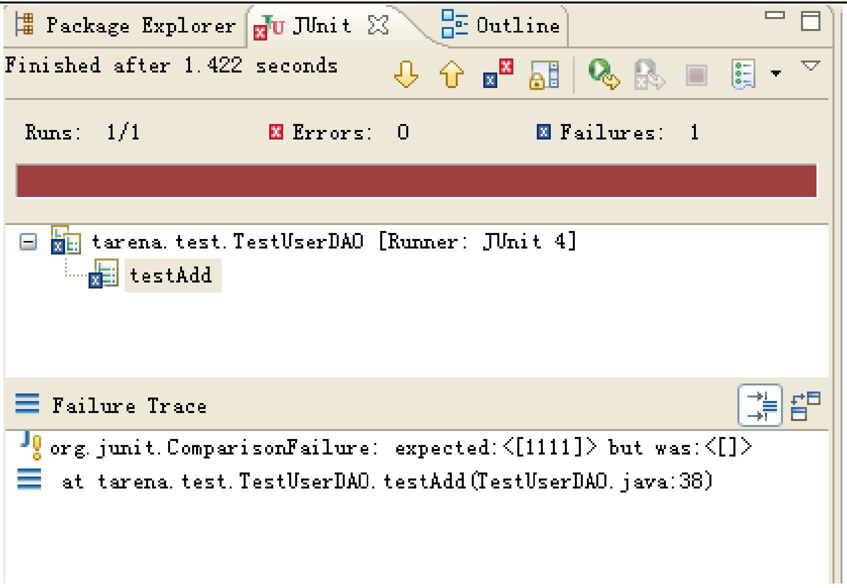
如果我们测试 password

首先，控制台没有输出



```
//采用junit断言，判断写入字段是否正确
Assert.assertEquals("jdbc@tarena.com", user1.getEmail(
Assert.assertEquals("jdbc", user1.getNickname());
System.out.println("####");
System.out.println(user1.getPassword());
System.out.println("####");
Assert.assertEquals("1111",user1.getPassword());
System.out.println("####");
```

其次，断言也不能通过



```
Finished after 1.422 seconds
Runs: 1/1    Errors: 0    Failures: 1
tarena.test.TestUserDAO [Runner: JUnit 4]
    testAdd
org.junit.ComparisonFailure: expected:<[1111]> but was:<[]>
    at tarena.test.TestUserDAO.testAdd(TestUserDAO.java:38)
```

（案例结束）

【案例 3】Spring 整合 Hibernate **

1) 使用 spring3 工程

请下载 spring3.zip

2) 导入 Hibernate 的 jar 包

请下载 hibernate_lib.zip

需要注意：做框架整合的时候，会遇到 jar 包冲突的问题，有些异常是由于 jar 包冲突引起的，请调试程序时注意。

Spring 对 Hibernate 的整合是在 JDBC 之上的

首先，配置好数据源 dataSource，
其次，加入 SessionFactory，
第三，配置映射文件

3) schema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
```

```
          xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver"></property>
        <property name="url"
            value="jdbc:mysql://localhost:3306/test"></property>
        <property name="username" value="root"></property>
        <property name="password" value="root"></property>
        <property name="maxActive" value="10"></property>
        <property name="initialSize" value="2"></property>
        <property name="minIdle" value="2"></property>
        <property name="maxIdle" value="3"></property>
    </bean>

    <bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

        <property name="dataSource" ref="dataSource"></property>
        <property name="mappingResources">
            <list>
                <value>tarena/entity/User.hbm.xml</value>
            </list>
        </property>

        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                        org.hibernate.dialect.MySQL5Dialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
            </props>
            </property>
        </bean>

</beans>
```

4)　新建映射文件 User.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping package="tarena.entity">
    <class name="User" table="d_user" catalog="test">
        <id name="id" type="integer">
            <column name="id" />
            <generator class="native"></generator>
        </id>
        <property name="email" type="string">
            <column name="email" length="50" not-null="true"
                 unique="true" />
        </property>
        <property name="nickname" type="string">
            <column name="nickname" length="50" />
        </property>
        <property name="password" type="string">
            <column name="password" length="50" not-null="true" />
        </property>
        <property name="userIntegral" type="integer">
            <column name="user_integral" not-null="true" />
        </property>
        <property name="emailVerify" type="yes_no">
            <column name="is_email_verify" length="3" />
        </property>
        <property name="emailVerifyCode" type="string">
            <column name="email_verify_code" length="50" />
        </property>
        <property name="lastLoginTime" type="long">
            <column name="last_login_time" />
        </property>
        <property name="lastLoginIp" type="string">
            <column name="last_login_ip" length="15" />
        </property>
    </class>
</hibernate-mapping>
```

5) User

```
package tarena.entity;

public class User implements java.io.Serializable {

    // Fields
    private Integer id;
    private String email = "";
    private String nickname = "";
    private String password = "";
    private Integer userIntegral = 0;
    private boolean emailVerify = false;
    private String emailVerifyCode = "";
    private long lastLoginTime = 0L;
    private String lastLoginIp = "";

    // Constructors

    /** default constructor */
    public User() {
    }

    /** minimal constructor */
    public User(String email, String password, Integer userIntegral) {
        this.email = email;
        this.password = password;
        this.userIntegral = userIntegral;
```

```java
	}

	public boolean isEmailVerify() {return emailVerify;}
	public void setEmailVerify(boolean emailVerify) {
		this.emailVerify = emailVerify;}
	public Integer getId() {return this.id;}
	public void setId(Integer id) {this.id = id;}
	public String getEmail() {return this.email;}
	public void setEmail(String email) {this.email = email;}
	public String getNickname() {return this.nickname;}
	public void setNickname(String nickname) {
		this.nickname = nickname;}
	public String getPassword() {
		return this.password;}
	public void setPassword(String password) {
		this.password = password;}
	public Integer getUserIntegral() {
		return this.userIntegral;}
	public void setUserIntegral(Integer userIntegral) {
		this.userIntegral = userIntegral;}
	public String getEmailVerifyCode() {
		return this.emailVerifyCode;}
	public void setEmailVerifyCode(String emailVerifyCode) {
		this.emailVerifyCode = emailVerifyCode;}
	public long getLastLoginTime() {
		return this.lastLoginTime;}
	public void setLastLoginTime(long lastLoginTime) {
		this.lastLoginTime = lastLoginTime;}
	public String getLastLoginIp() {
		return this.lastLoginIp;}
	public void setLastLoginIp(String lastLoginIp) {
		this.lastLoginIp = lastLoginIp;}
}
```

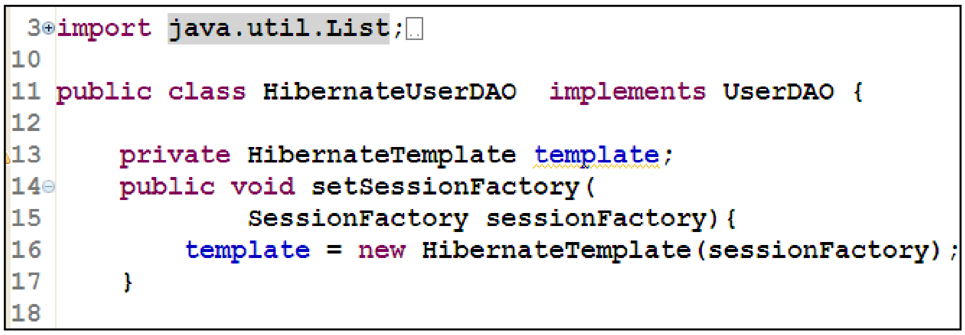
6) UserDao

```java
package tarena.dao;

import java.util.List;
import tarena.entity.User;

public interface UserDAO {
	public void save(User user);
	public void update(User user);
	public void deleteById(int id);
	public User findById(int id);
	public List<User> findAll();
	public int    count();
}
```

7) 新建 HibernateUserDao

```
package tarena.dao;


import java.util.List;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import tarena.entity.User;

public class HibernateUserDAO
	extends HibernateDaoSupport implements UserDAO {

	public int count() {
		String hql = "select count(*) from User";
		List list = this.getHibernateTemplate().find(hql);
		return Integer.valueOf(list.get(0).toString());
	}

	public void deleteById(int id) {
		User user = findById(id);
		this.getHibernateTemplate().delete(user);
	}

	public List<User> findAll() {
		String hql = "from User";
		return this.getHibernateTemplate().find(hql);
	}
	public User findById(int id) {
//		User user = (User)this.getHibernateTemplate().get(User.class, id);
		String hql = "from User where id=?";
		List list = this.getHibernateTemplate().find(hql,new Object[]{id});
		if(!list.isEmpty()){
			return (User)list.get(0);
		}
		return null;
	}

	public void save(User user) {
		this.getHibernateTemplate().save(user);

	}

	public void update(User user) {
		this.getHibernateTemplate().update(user);


	}

}
```

如果不继承 HibernateUserDao，那么我们使用 set 方法注入也是可以的



```
import java.util.List;

public class HibernateUserDAO  implements UserDAO {

	private HibernateTemplate template;
	public void setSessionFactory(
			SessionFactory sessionFactory){
		template = new HibernateTemplate(sessionFactory);
	}

```

接下来注入 DAO

8)　修改 schema.xml

首先，向 Spring 容器注入 bean 组件 DAO

其次，向注入 bean 组件 DAO 中注入 sessionFactory

```
<bean id="dataSource" destroy-method="close"
class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
               value="com.mysql.jdbc.Driver"></property>
        <property name="url"
               value="jdbc:mysql://localhost:3306/test"></property>
        <property name="username" value="root"></property>
        <property name="password" value="root"></property>
        <property name="maxActive" value="10"></property>
        <property name="initialSize" value="2"></property>
        <property name="minIdle" value="2"></property>
        <property name="maxIdle" value="3"></property>
</bean>

<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

        <property name="dataSource" ref="dataSource"></property>
        <property name="mappingResources">
               <list>
                      <value>tarena/entity/User.hbm.xml</value>
               </list>
        </property>

        <property name="hibernateProperties">
               <props>
                      <prop key="hibernate.dialect">
                             org.hibernate.dialect.MySQL5Dialect</prop>
                      <prop key="hibernate.show_sql">true</prop>
                      <prop key="hibernate.format_sql">true</prop>
               </props>
        </property>

</bean>

<bean id="hibernateUserDao" class="tarena.dao.HibernateUserDAO">
       <!--记得要注入sessionFactory-->
```

```
            <property name="sessionFactory" ref="sessionFactory"></property>
        </bean>
</beans>

```

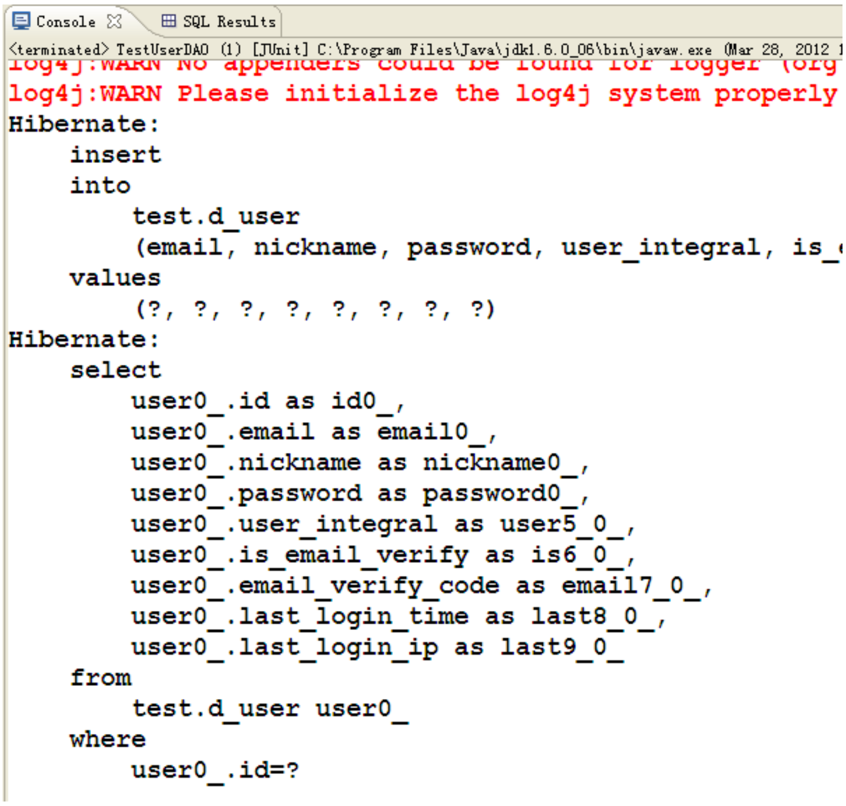
9) TestUserDAO

```
package tarena.test;

public class TestUserDAO {

    @Test
    public void testHibernateDelete(){
        ApplicationContext ac =
                new ClassPathXmlApplicationContext("schema.xml");
        UserDAO userDao = (UserDAO)ac.getBean("hibernateUserDao");
        userDao.deleteById(1);
    }


    public void testHibernateAdd(){
        User user = new User();
        user.setEmail("hibernate@163.com");
        user.setNickname("hibernate");
        user.setPassword("1111");
        user.setUserIntegral(10);
        user.setEmailVerify(false);
        user.setEmailVerifyCode("asdfasdew");
        user.setLastLoginTime(System.currentTimeMillis());
        user.setLastLoginIp("192.168.2.1");
        ApplicationContext ac =
                new ClassPathXmlApplicationContext("schema.xml");
        UserDAO userDao = (UserDAO)ac.getBean("hibernateUserDao");
        userDao.save(user);
        //采用junit断言，判断写入字段是否正确
        User user1 = userDao.findById(1);
        Assert.assertEquals("hibernate@163.com", user1.getEmail());
        Assert.assertEquals("hibernate", user1.getNickname());
//      Assert.assertEquals("1111",user1.getPassword());
```

```
    }
}
```

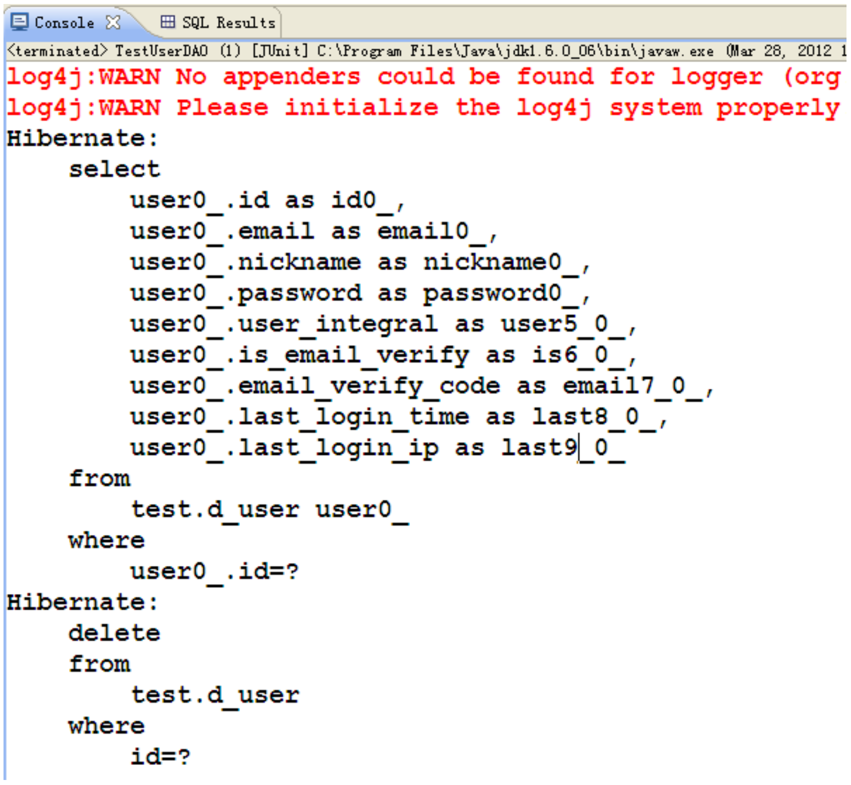
10）运行 TestUserDAO

插入&&查询



```
<terminated> TestUserDAO (1) [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 28, 2012 1
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Hibernate: 
    insert 
    into
        test.d_user
        (email, nickname, password, user_integral, is_
    values
        (?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: 
    select
        user0_.id as id0_,
        user0_.email as email0_,
        user0_.nickname as nickname0_,
        user0_.password as password0_,
        user0_.user_integral as user5_0_,
        user0_.is_email_verify as is6_0_,
        user0_.email_verify_code as email7_0_,
        user0_.last_login_time as last8_0_,
        user0_.last_login_ip as last9_0_ 
    from
        test.d_user user0_ 
    where
        user0_.id=?
```

删除



```
<terminated> TestUserDAO (1) [JUnit] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 28, 2012 1
log4j:WARN No appenders could be found for logger (org
log4j:WARN Please initialize the log4j system properly
Hibernate:
    select
        user0_.id as id0_,
        user0_.email as email0_,
        user0_.nickname as nickname0_,
        user0_.password as password0_,
        user0_.user_integral as user5_0_,
        user0_.is_email_verify as is6_0_,
        user0_.email_verify_code as email7_0_,
        user0_.last_login_time as last8_0_,
        user0_.last_login_ip as last9_0_
    from
        test.d_user user0_
    where
        user0_.id=?
Hibernate:
    delete
    from
        test.d_user
    where
        id=?
```

（案例结束）