创作人：刘谚琪

QQ: 172832876

MSN: lanlanq@hotmail.com

Webwork+spring+hibernate 配置过程心得

首先需要的 lib 环境不可少

持久层

首先创建 DAO 基类

abstract class _RootDAO extends HibernateDaoSupport

```
public abstract class _RootDAO extends HibernateDaoSupport {
    /**
     * Return the specific Object class that will be used for
class-specific
     * implementation of this DAO.
     * @return the reference Class
     */
    protected abstract Class getReferenceClass();

    protected Order getDefaultOrder() {
        return null;
    }


    public Object get(Class entityClass, Serializable id) throws
            DataAccessException {
        return get(entityClass, id, null);
    }

    public Object get(final Class entityClass, final Serializable id,
                      final       LockMode       lockMode)       throws
DataAccessException {
        return      getHibernateTemplate().get(entityClass,      id,
lockMode);
    }

    public Object get(String entityName, Serializable id) throws
            DataAccessException {
        return get(entityName, id, null);
    }
```

```
    public Object get(final String entityName, final Serializable id,
                      final       LockMode       lockMode)       throws
DataAccessException {
        return getHibernateTemplate().get(entityName, id, lockMode);
    }

……….
……,.
……

        public     int     updateQueryAndNamedParam(final     String
queryString,
                        final  String[]  paramNames,  final  Object[]
values)
                        throws DataAccessException {
     return this.updateQueryAndNamedParam( queryString, paramNames,
values );
        }

     */

}
```

然后创建 Manage 的基类

public abstract class _RootManage protected _RootDAO dao;

（引入 Dao 基类）

```
public abstract class _RootManage {
    protected _RootDAO dao;

        /**
         * Return the specific Object class that will be used for
class-specific
         * implementation of this DAO.
         * @return the reference Class
         */
        protected abstract Class getReferenceClass();

        protected Order getDefaultOrder () {
                return null;
        }

        public Object get(Class entityClass, Serializable id) throws
DataAccessException {
                return get(entityClass, id, null);
        }

        public Object get(final Class entityClass, final Serializable
id, final LockMode lockMode)
                        throws DataAccessException {
                return dao.get(entityClass, id, lockMode);
        }
```

```
        public Object get(String entityName, Serializable id) throws
DataAccessException {
                return get(entityName, id, null);
        }

        public Object get(final String entityName, final Serializable
id, final LockMode lockMode)
                        throws DataAccessException {
                return dao.get(entityName, id, lockMode);
        }

…….. ……..
……..
……

        public  int  updateQuery(final  String  queryString,  final
Object[] values) throws DataAccessException {
                return dao.updateQuery(queryString, values);
        }

 }
```

然后创建对应业务的持久类，继承基类 OnlineUserMonitorDAO(继承 DAO)

```
public class OnlineUserMonitorDAO extends _RootDAO {
    private static final Log log = LogFactory.getLog(
            OnlineUserMonitorHomeManage.class);

    public Class getReferenceClass() {
        return OnlineUserMonitor.class;
    }

    public Order getDefaultOrder() {
        return null;
    }


    public void persist(OnlineUserMonitor transientInstance) {
        log.debug("persisting OnlineUserMonitor instance");
        persist(transientInstance);
        log.debug("persist successful");

    }

   ……..

    public OnlineUserMonitor merge(OnlineUserMonitor detachedInstance) {
        log.debug("merging OnlineUserMonitor instance");
```

```
        OnlineUserMonitor result = (OnlineUserMonitor) merge(
                detachedInstance);
        log.debug("merge successful");
        return result;

    }

    public OnlineUserMonitor findById(java.lang.String id) {
        // log.debug("getting OnlineUserMonitor instance with id: " + id);
        OnlineUserMonitor instance = (OnlineUserMonitor) get(
                "OnlineUserMonitor", id);
        if (instance == null) {
            log.debug("get successful, no instance found");
        } else {
            log.debug("get successful, instance found");
        }
        return instance;
    }
    public java.util.List findAll(){
        System.out.println("操作错误 2");
        return find("from OnlineUserMonitor");
    }

}
```

然后创建对应得 Mange 实现类

public abstract class OnlineUserMonitorHomeManage extends jb.base._RootManage

```
public abstract class OnlineUserMonitorHomeManage extends jb.base._RootManage {

    private static final Log log = LogFactory.getLog(
            OnlineUserMonitorHomeManage.class);

    public Class getReferenceClass() {
        return OnlineUserMonitor.class;
    }

    public Order getDefaultOrder() {
        return null;
    }


    public void persist(OnlineUserMonitor transientInstance) {
        log.debug("persisting OnlineUserMonitor instance");
        persist(transientInstance);
        log.debug("persist successful");

    }

   ……..
```

```
    public OnlineUserMonitor merge(OnlineUserMonitor detachedInstance) {
        log.debug("merging OnlineUserMonitor instance");

        OnlineUserMonitor result = (OnlineUserMonitor) merge(
                detachedInstance);
        log.debug("merge successful");
        return result;

    }

    public OnlineUserMonitor findById(java.lang.String id) {
        // log.debug("getting OnlineUserMonitor instance with id: " + id);
        OnlineUserMonitor instance = (OnlineUserMonitor) get(
                "OnlineUserMonitor", id);
        if (instance == null) {
            log.debug("get successful, no instance found");
        } else {
            log.debug("get successful, instance found");
        }
        return instance;
    }
    public java.util.List findAll(){
        return dao.find("from OnlineUserMonitor");
    }

    public void setOnlineuserdao(OnlineUserMonitorDAO onlineuserdao) {
        this.dao = onlineuserdao;
    }

    public OnlineUserMonitorDAO getOnlineuserdao() {
        return (OnlineUserMonitorDAO)dao;
    }

}
```

当然前提要配置好映射文件。配置和获得映射文件的过程在《Eclipse 配置 Hibernate3 插件.pdf》这篇文章中详细说明。可以参考。

然后可以做一个持久类的实现类和 action 进行关联

public class OnlineUserMonitorImp extends jb.dao.OnlineUserMonitorHomeManage

```
public class OnlineUserMonitorImp extends jb.dao.OnlineUserMonitorHomeManage {
    private static final Log log = LogFactory.getLog(
            OnlineUserMonitorImp.class);
    public OnlineUserMonitorImp() {
    }
```

```
    public java.util.List findAll() {
        java.util.List list = new ArrayList();
        Iterator iterate = iterate("from OnlineUserMonitor");
        if(iterate ==null){
            log.error("不能获得内容");
            return null;
        }else{
            while (iterate.hasNext()) {
                OnlineUserMonitor onlineUserMonitor = (OnlineUserMonitor)
                        iterate.
                        next();
                System.out.println(onlineUserMonitor.getUserGuid());
                list.add(onlineUserMonitor);
            }
            return list;
        }
    }

}
```

这样做的目的就是为了，webwork 的 action 可以通过 spring 的配置获得借口类的实例。

然后创建 action

public class OnlineUserMonitorActionBean implements Action, ModelDriven

把持久类的实例对象通过 spring 来获得，注意的是在配置中要增加引入设置（xwork.xml）

```
    <!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.0//EN"
"http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork>
    <include file="webwork-default.xml"></include>
   ……
    列出 webwork 需要的配置即可
</xwork>
```

这样就可以直接在 action 中设置属性来获得持久类实例的值，名字要对应 spring 配置中的 bean 的 id,返回对象要设置接口类型，这样可以做到依赖注入

```
public class OnlineUserMonitorActionBean implements Action, ModelDriven {
    OnlineUserMonitorModel model = new OnlineUserMonitorModel();
    public OnlineUserMonitorHomeManage onlineUserManager;
    public Object getModel() {
        return model;
    }

    public String execute() throws Exception {
        if (model.getValue1().equals("hello")) {
            model.setInfo(this.getOnlineUserManager().findAll());
```

```
            return "success";
        } else {
            return "falsity";
        }
    }

    public void setOnlineUserManager(OnlineUserMonitorHomeManage
                                     onlineUserManager) {

        this.onlineUserManager = onlineUserManager;
    }

    public OnlineUserMonitorHomeManage getOnlineUserManager() {

        return onlineUserManager;
    }

}
```

对应的 action 出来了，就可以从页面上显示出来了

```
<%@ include file="/taglibs.jsp"%>
<html>
<head>
<title>
insert
</title>
</head>

<body bgcolor="#f4f4f4">
<h1>
JBuilder Generated JSP
</h1>
<table>

<#list model.info as infolist>
<tr>
<td>
${infolist.guid}
</td>
</tr>
</#list>
</body>
</html>
```

下面把配置贴出来

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <!-- Spring 的配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/ApplicationContext-*.xml</param-value>
    </context-param>

    <filter>
        <filter-name>OpenSessionInViewFilter</filter-name>

    <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewF
ilter</filter-class>
        <init-param>
            <param-name>sessionFactoryBeanName</param-name>
            <param-value>sessionFactory</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>OpenSessionInViewFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>OpenSessionInViewFilter</filter-name>
        <url-pattern>*.action</url-pattern>
    </filter-mapping>

    <listener>

    <listener-class>org.springframework.web.context.ContextLoaderListener</list
ener-class>
    </listener>

    <!--webwork 迁入-->
……
……

</web-app>
```

Application-hibernate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
```

```
        ..............................
        ..............................
    </bean>

    <!--Hibernate 的 SessionFactory-->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"
lazy-init="true">
        <property name="mappingResources">
            <list>
                <!--在此可以添加相应的对象映射-->
                <value>jb/dao/OnlineUserMonitor.hbm.xml</value>
<!--                <value>vjob.hbm.xml</value>-->
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.Oracle9Dialect
                </prop>
                <!--是否根据 Model 的改变自动更新数据库字段-->
<!--                <prop key="hibernate.hbm2ddl.auto">update</prop>-->
                <!--是否显示 SQL-->
<!--                <prop key="hibernate.show_sql">true</prop>-->
<!--                <prop key="hibernate.jdbc.fetch_size">50</prop>
                <prop key="hibernate.jdbc.batch_size">25</prop> -->
                <prop key="hibernate.c3p0.minPoolSize">5</prop>
                <prop key="hibernate.c3p0.maxPoolSize">20</prop>
                <prop key="hibernate.c3p0.timeout">60</prop>
                <prop key="hibernate.c3p0.max_statement">50</prop>
                <prop
key="hibernate.c3p0.testConnectionOnCheckout">false</prop>
                <!--是否对查询进行缓存-->
                <prop key="hibernate.cache.use_query_cache">true</prop>
                <!--使用哪个缓存类进行缓存-->
                <prop key="hibernate.cache.provider_class">
                    org.hibernate.cache.OSCacheProvider
                </prop>

            </props>
        </property>
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
    </bean>

    <bean                                                    name="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource">
            <ref bean="dataSource" />
```

```
        </property>
    </bean>

    <bean id="hibernateTemplate"
        class="org.springframework.orm.hibernate3.HibernateTemplate"
        lazy-init="true">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
        <property name="cacheQueries">
            <value>true</value>
        </property>
        <property name="queryCacheRegion">
            <value>sdbys_query_cache</value>
        </property>
    </bean>

    <bean id="hibernateTemplateForDOM4J"
        class="org.springframework.orm.hibernate3.HibernateTemplate"
        lazy-init="true">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
        <property name="cacheQueries">
            <value>true</value>
        </property>
        <property name="queryCacheRegion">
            <value>sdbys_query_cache</value>
        </property>
    </bean>

    <bean id="hibernateTemplateForMAP"
        class="org.springframework.orm.hibernate3.HibernateTemplate"
        lazy-init="true">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
        <property name="cacheQueries">
            <value>true</value>
        </property>
        <property name="queryCacheRegion">
            <value>sdbys_query_cache</value>
        </property>
    </bean>

    <bean id="hibernateTransactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager"
        lazy-init="true">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
```

```
        </property>
    </bean>

    <bean id="txAttributes"

    class="org.springframework.transaction.interceptor.MatchAlwaysTransactionAt
tributeSource">
        <property name="transactionAttribute">
            <value>PROPAGATION_REQUIRED</value>
        </property>
    </bean>
    <bean id="onlineuserdao" class="jb.dao.OnlineUserMonitorDAO" >
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <bean id="onlineUserManager" class="jb.dao.OnlineUserMonitorImp" >
        <property name="onlineuserdao">
            <ref local="onlineuserdao" />
        </property>
    </bean>
</beans>
```

如果遇到 resultset 关闭要看看下面内容是否引入。

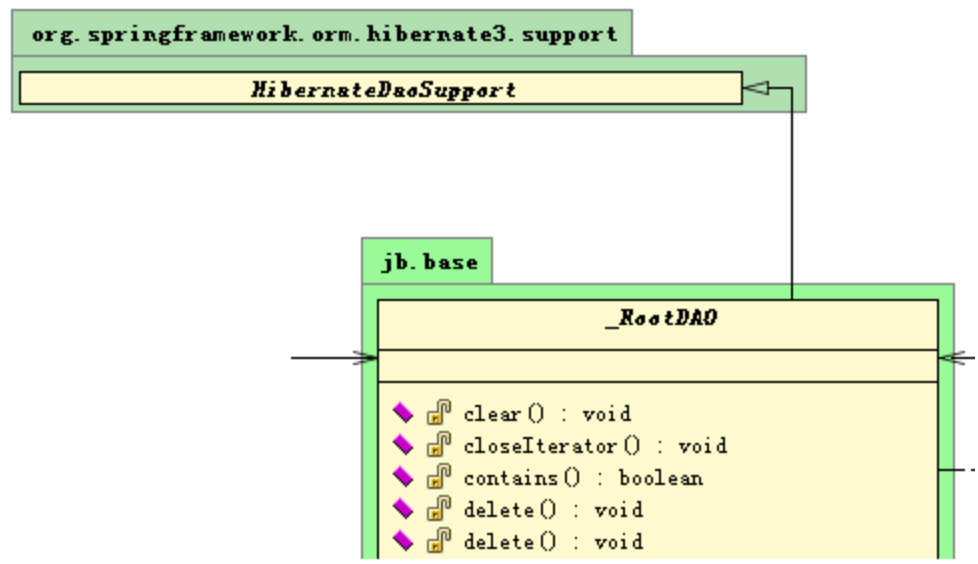
```
<!-- 这个Filter的主要功能是确保一个Request过程中只会产生一个Hibernate Session,
        相当于一个请求一个数据库连接
        -->
    <filter>
        <filter-name>OpenSessionInViewFilter</filter-name>

    <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewF
ilter</filter-class>
        <init-param>
            <param-name>sessionFactoryBeanName</param-name>
            <param-value>sessionFactory</param-value>
        </init-param>
    </filter>

    <filter-name>OpenSessionInViewFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>OpenSessionInViewFilter</filter-name>
        <url-pattern>*.action</url-pattern>
    </filter-mapping>
```

在配置中遇到的问题

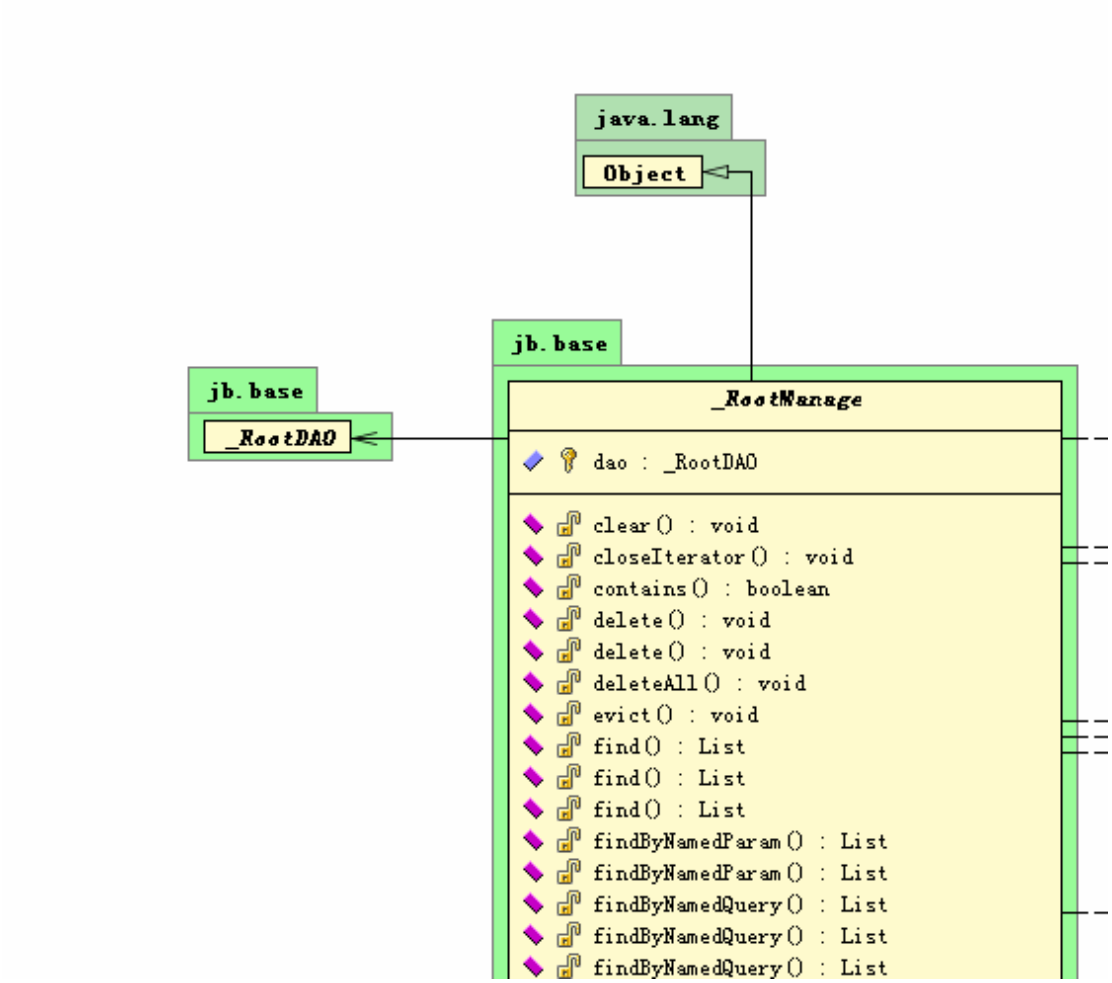
在配置基类后然后继承基类来实现持久层的工作。然后通过 spring 来映射

曾经晕过多次，不理解也不清楚怎么转。

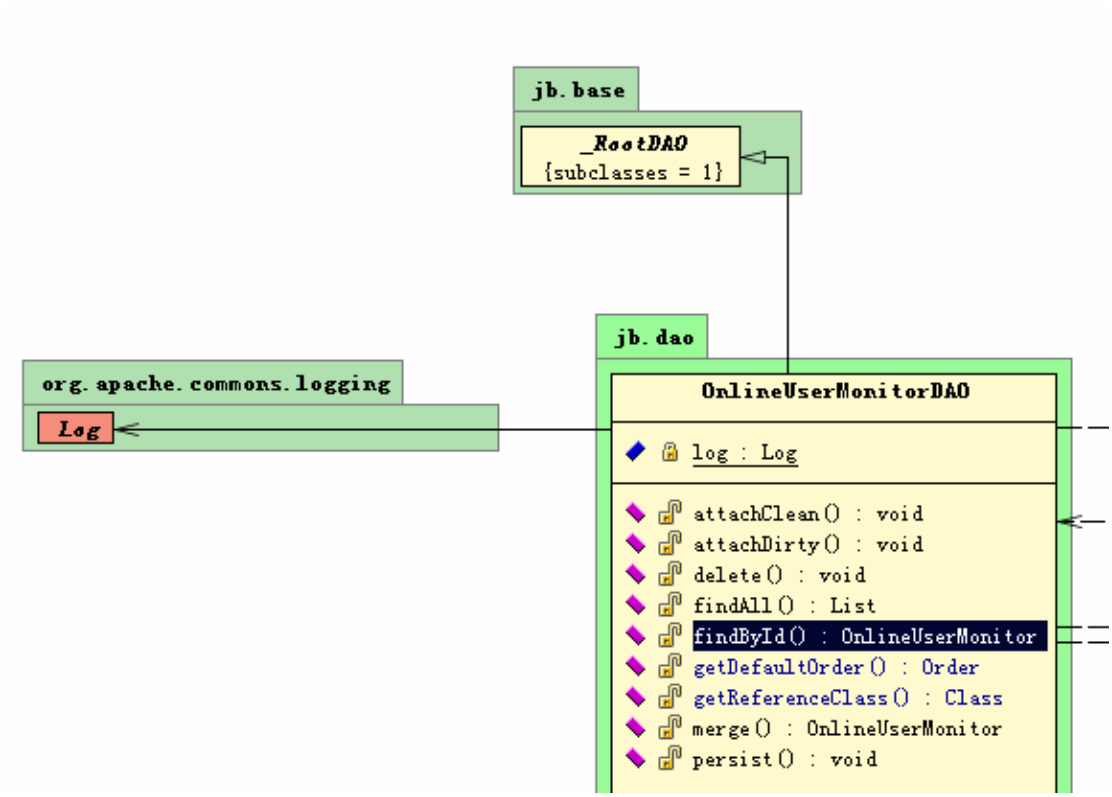
现在我想描述一下，



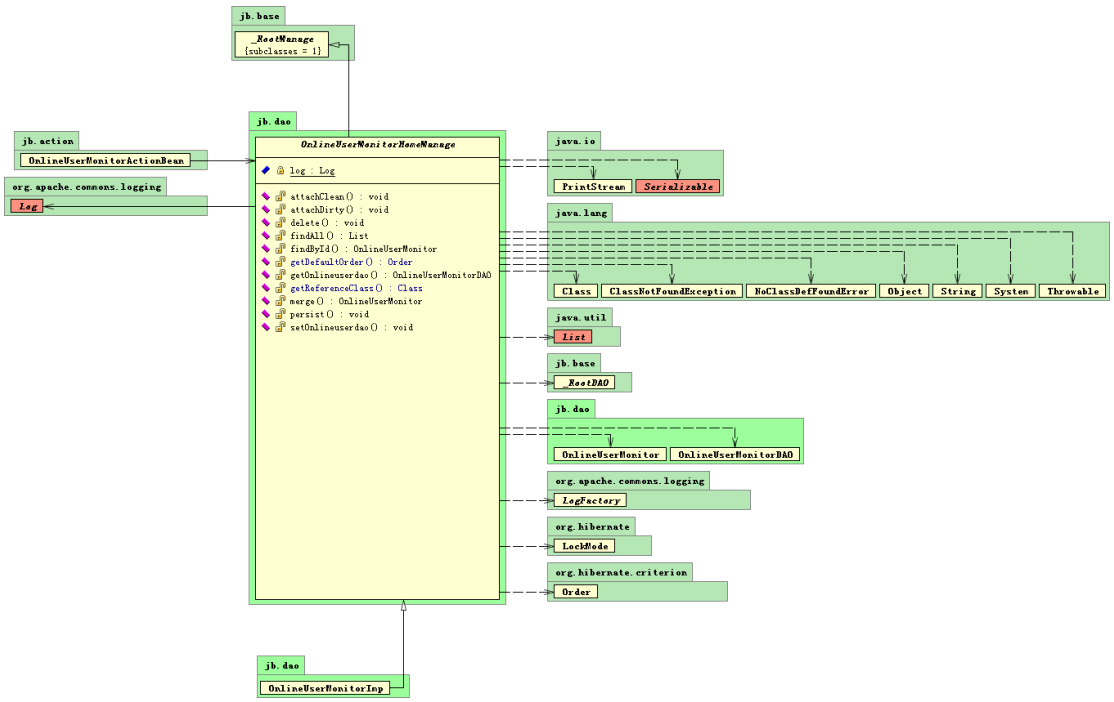
首先_RootDAO 继承 hibernateDaoSupport 来获得 hibernate 的对象。



而_RootManage 依赖_RootDAO，通过属性对象进行处理。



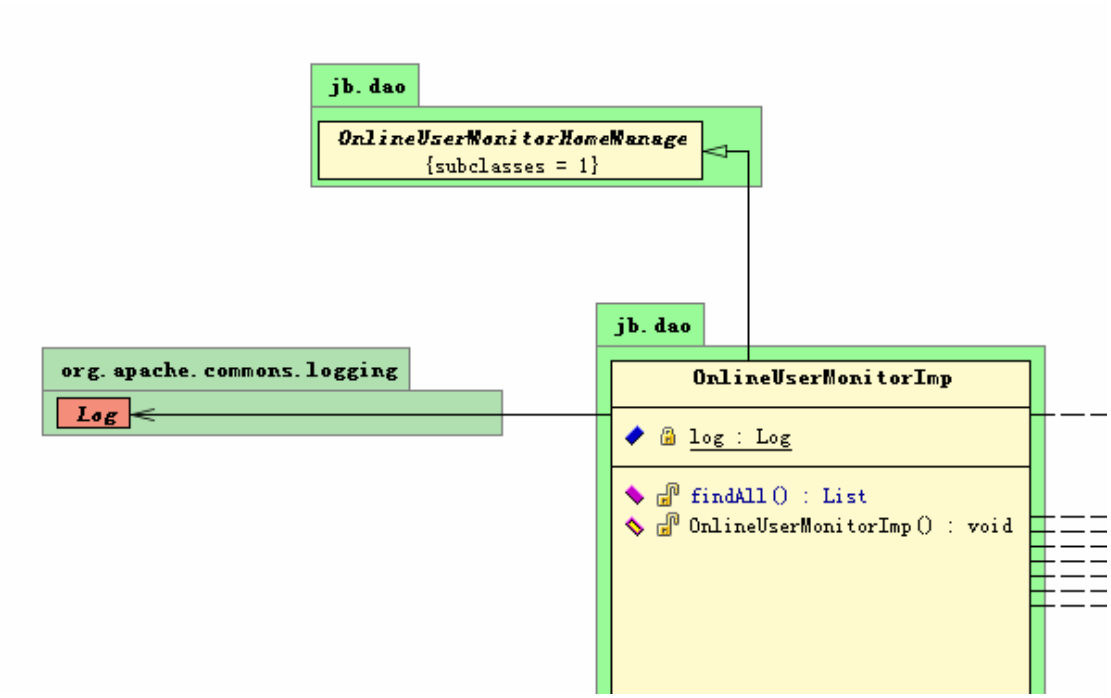
接着 OnlineUserMonitorDAO 依赖 Log 日志继承自_RootDAO.方法结合业务（数据库对象）对象。



从这里可以看出，让 OnlineUserMonitorHomeManage 依赖 DAO 不过没有真正依赖，而是继承。而通过 spring 进行处理后就变成依赖，通过配置获得 onlineuserdao 的对象传送给 DAO。然后通过 OnlineUserMonitorHomeManage 的继承基层 Manage 进行操作，事实上把 DAO 对象

通过 spring 配置来传输给 OnlineUserMonitorHomeManage，而 OnlineUserMonitorHomeManage 通过属性来传送给 DAO 基类。而 Manage 基类又依赖 DAO 基类，所以 Mange 基类就调用到通过 spring 传送过来的 Dao 对象内容。

```
<bean id="onlineuserdao" class="jb.dao.OnlineUserMonitorDAO" >
    <property name="sessionFactory">
        <ref local="sessionFactory" />
    </property>
</bean>

<bean id="onlineUserManager" class="jb.dao.OnlineUserMonitorImp" >
    <property name="onlineuserdao">
        <ref local="onlineuserdao" />
    </property>
</bean>
```



那么来继承 OnlineUserMonitorHomeManage 进行实际的视线操作就可以进行扩充和变更了。

那么可想而知。Action 通过 实现的接口类来获得持久层传来的数据