ClearWork开发参考手册

Author: huqi, gouruifeng

Version: V 1.0.0

前言

1. 文档说明

ClearWork提供一个供中小企业快速构建信息系统的框架和工具集合，同时提供能够快速上手一些优秀开源项目的示范。 主要面向中国用户，提供良好的中文支持。

这个团队的成员主要来自一些知名公司的富有经验的一线研发人员，对编程和开源长久的保持兴趣。所以，这个项目一开始就会具有很多从各种不同项目中提炼出来的、经过证明具有实用价值的功能……

ClearWork是基于JAVA技术的快速应用集成开发框架。 该框架主要致力于解决当前软件开发过程中的两个关键问题：软件复用问题和快速开发问题。 并通过对这些问题的解决来满足中国中小企业对软件质量、开发周期等方面的要求。 该开发框架不仅仅是一套快速开发应用软件的辅助工具，而且是一套提供很高复用度的软件开发模式。

请关注以下站点，关注ClearWork：

1. ClearWork@SourceForge [http://sourceforge.net/projects/clearwork]
2. ClearWork Wiki [http://clearwork.wiki.sourceforge.net/]

我们在任何时候都欢迎您亲自参与或加入ClearWork：

1. ClearWork Bug Reports (Bug 报告) [http://sourceforge.net/tracker/?func=add&group_id=202445&atid=981611]
2. ClearWork Feature Requests (功能/需求建议) [http://sourceforge.net/tracker/?func=add&group_id=202445&atid=981614]
3. ClearWork Forums (论坛) [http://sourceforge.net/forum/?group_id=202445]

本文档基于 DocBookBlank [http://sourceforge.net/projects/docbookblank] 项目构建，DocBookBlank 是ClearWork的子项目。

2. 目标定义

作为一个Framework，ClearWork严守对不同厂商和产品的中立，原则上不与任何开源软件产品或软件厂商绑定； ClearWork唯一依赖的是Spring框架(Spring Framework [http://springframework.org/])，它的大部分基础工具和组件都是基于对Spring框架的增强和扩展（事实上，Spring正是基于一个良好支持组件选择的集成架构）。

基于以上定义，ClearWork将以提供“优雅”“纯洁”的框架、工具集合以及被验证为具有实用价值的开源产品使用指南为目标。

ClearWork将JavaEE Web Application的后台架构划分为3个层次，它们分别是：

- 控制层(controller)——ClearWork在这一层对Struts等技术提供良好的集成\扩展\标准化示范。
- 服务/业务逻辑层(service)——ClearWork使这一层能够基于Axis2技术方便暴露为Web Service。
- 持久化层(dao)——ClearWork在这一层对JDBC以及Hibernate等技术提供良好的集成\扩展\标准化示范。

ClearWork的标准项目travelagency（设计中）以网上旅行社为蓝本， 通过建立一个供商家（旅行社）自由发布/销售其旅行产品的C2C信息平台来检验和展示这个框架的各个部分， 这包括组织、权限、报表、在线分析、SOA/Web Service、工作流……

ClearWork的示范和标准项目都采用Hsqldb / Mysql作为数据库，使用标准的SQL语句， 以保持对所有主流商业数据库的无缝兼容为指导目标，这包括：Oracle，MS SQL Server，IBM DB2。

see more... [http://clearwork.wiki.sourceforge.net/projectgoals]

第 1 章 Quick Start

1.1. 运行环境

设计支持运行在任何Web容器（默认Tomcat）和数据库系统（默认Hsqldb）中。

1.2. 如何快速入手

建议您仔细阅读ClearWork Wiki How to Quick Start [http://clearwork.wiki.sourceforge.net/QuickStart]

同时，您还可以参考ClearWork项目的目录结构设置 [http://clearwork.wiki.sourceforge.net/projectdirectory]

第 2 章 ClearWork应用工具集

位于net.sf.clearwork.core.utils包中。

2.1. 基础工具

工具名称	功能说明
MessageUtils	国际化工具类，获取国际化的消息资源，依赖于spring.xml 中的 messageSource Bean。
ResourceUtils	系统应用资源类，用于获取系统的资源和一些路径，这包括应用的相对路径和系统的绝对路径。支持regString参数查询，例如： 
SpringContextUtils	Spring上下文工具，主要提供getBean、getContext等功能
TestUtils	提供一些调试的常用方法
BeanUtils	Java Bean操作工具。它提供： 获取、设置私有属性； 执行私有方法； 获取超类； 获取类型的域名形式（即包名倒序）及其对应的XML Schema表示； 验证POJO（纯洁的Java对象，在Web Service中使用）类型等静态方法。
DateUtils	时间日期的工具类，提供常用的时间日期操作方法，可以看作是apache commons相应工具的有益补充。
NumberUtils	数值操作的工具类，提供常用的数值操作、格式化方法，可以看作是apache commons相应工具的有益补充。
StringUtils	提供常用的字符串操作方法，也包括字符编码转换、 HTML的encode/decode、URL的encode/decode等功能。
UniqueIdUtils / UUIDHexGenerator / UUIDUtils	这三种唯一ID生成工具可以在100～150线程并发的情况下安全的产生不重复的ID， 其格式分别为： XXXXXXXX-XXXXX / 32位字符串 / 49位字符串。源代码内置多线程测试的main方法，可以根据测试的结果和系统的需求选择使用。

c

config/**/spr

工具名称	功能说明
ImageUtils	图片处理工具，提供对类型 ImageFormat 枚举的图片格式的处理， 支持缩放、按比例缩放、缩略图、转存、获取图片信息、添加水印文字或图形等功能。 水印位置可参考枚举 WatermarkLayout 类。
DESUtils / RSAUtils / MD5Utils / SHAUtils	提供这四种加密算法的支持，可用于密码字符串加密。 注意：前两者是可逆的加密算法；后两者不可逆，即加密后无法解密。
WebUtils	继承自 org.springframework.web.util.WebUtils ，提供一些web层的通用工具方法。
XmlUtils	利用 W3C API构建的标准XML操作工具，同时还提供转换GBK编码格式XML文档的方法。

2.2. PopulateUtils——值注入工具

解决应用开发中普遍存在的场景——不同值对象之间同名域的注入（对应populate*方法）/拷贝（对应copyProperties*方法）。 populate方法支持将来自HTTP Post的表单数据注入值对象、将来自JDBC RecorderSet的记录集注入值对象、对两个值对象同名属性域值的拷贝等操作。

工具的通用规则：

- 需要相互注入拷贝的域（field）必须同名，但不区分大小写。
- 支持nameMapper参数，域名称转换映射
- 支持ignoreProperties参数，忽略的属性域列表
- 可以开发插件转化特殊的数据类型，例如默认的DatePopulatePlugin被用来支持如下转换：

```
<property name="plugins">
        <map>
                <entry key="java.sql.Date->*">
                        <ref local="datePopulatePlugin" />
                </entry>
                <entry key="java.sql.Timestamp->*">
                        <ref local="datePopulatePlugin" />
                </entry>
                <entry key="*->java.util.Date">
                        <ref local="datePopulatePlugin" />
                </entry>
        </map>
</property>
```

它依赖于 spring.xml 中的 populateUtils Bean。值转化插件需要实现IPopulatePlugin接口。 关于如何开发值转化插件建议参考类 net.sf.clearwork.core.utils.bean.DatePopulatePlugin 的源代码。

- 支持集合或数组的值对象拷贝。

您可以通过查看Populate Sample（net.sf.clearwork.sample.populateutils.SamplePopulateAction ）来具体了解ClearWork值注入工具的应用。

2.3. ContextUtils——上下文管理框架工具

ClearWork框架中获取上下文的入口类型 net.sf.clearwork.core.utils.context.ContextUtils ，它提供三组功能：

1. getDataChannelContext，获取线程级（即同一线程）的上下文，基本上相当于ThreadLocal这样的容器。

2. getUserDataContext，获取用户级（即连接Web应用的同一用户）的上下文，基于HttpSession的容器。

3. getApplicationContext，获取应用级（即整个应用程序全局）的上下文，基于内存中缓存的容器。

拥有了所需的上下文资源（此资源实现 net.sf.clearwork.core.utils.context.IContext 接口）句柄，即可对其进行增删改查等属性操作，例如：将用户信息对象放入用户级上下文；将事务操作对象放入线程级上下文等。

第 3 章 ClearWork异常处理框架

这个异常处理的框架由四个部分组成：

1. 异常类型（Java代码实现）—— 所有的ClearWork异常都来自基类net.sf.clearwork.core.exception.BaseCheckedException 和 net.sf.clearwork.core.exception.BaseUnCheckedException ，这两个异常基类分别继承自org.springframework.core.NestedCheckedException 和 org.springframework.core.NestedRuntimeException ，确定了异常的受检和非受检属性。

   再上一层次，将异常划分为：业务逻辑层异常、数据操作层异常、应用框架异常、操作系统异常、用户界面异常。

   ClearWork建议用户抛出异常时参考上述异常划分，根据自己的需要选择使用，这应该能解决绝大多数的异常情景。 同时，建议用户在定义自己的异常时，根据受检和非受检的需要， 继承BaseCheckedException或BaseUnCheckedException。

2. 异常描述（Spring配置文件）——位于 classpath*:config/exception.xml ， 利用Spring配置机制注册了异常的基类，及一些常见的应用程序异常，这些异常大部分来自Spring Framework，因为ClearWork的很多部分依赖于Spring。 同时， ClearWork建议用户将自己定义的异常按照这种方式在exception.xml中进行配置注册，以便异常处理框架进行管理。 在 exception.xml 中，也定义了异常信息的国际化资源文件位置。

3. 异常信息（国际化资源）——位于 classpath*:i18n ，默认的国际化资源是中文。 可以在 exception_cn.txt 中编写您的异常信息， 然后利用 make_exception.bat (需要配置Java_Home/bin Path系统环境变量)将其转换为UTF-8编码的 exception_cn.properties 文件。

4. 异常展现（web层的controller以及view）——包括在 struts.xml 中的全局异常控制器配置及在Web Application全局目录 global 中的 error.jsp ，它们被用来处理UI层异常导向和面向浏览器用户展现的逻辑。您可以通过修改 error.jsp 切换 alert exception 和 alert exception and back 策略。

第 4 章 ClearWork单元测试

4.1. Unit Test——介绍一下单元测试

4.1.1. 什么是单元测试

单元测试是最小粒度的测试，以测试某个功能或代码块。 一般由程序开发者来做，因为做单元测试需要知道程序设计和编码的细节。 单元测试的对象是软件设计的最小单位——模块。 单元测试的依据是详细设描述，单元测试应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。单元测试多采用白盒测试技术，系统内多个模块可以并行地进行测试。

单元测试是在软件开发过程中要进行的最低级别的测试活动，在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。 在一种传统的结构化编程语言中，比如C，要进行测试的单元/模块一般是函数或子过程。 在象 Java 这样的面向对象的语言中，要进行测试的基本单元/模块是类。 单元测试的原则同样被扩展到第四代语言(4GL)的开发中，在这里基本单元被典型地划分为一个菜单或显示界面。

单元测试不仅仅是作为无错编码一种辅助手段在一次性的开发过程中使用， 同时，单元测试必须是可重复的，无论是在软件修改，或是移植到新的运行环境的过程中。 因此，所有的测试都必须在整个软件系统的生命周期中进行维护。

经常与单元测试联系起来的另外一些开发活动包括：代码走读(Code review)，静态分析(Static analysis)和动态分析(Dynamic analysis)。 静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。 动态分析就是通过观察软件运行时的动作，来提供执行跟踪，时间分析，以及测试覆盖度方面的信息。

4.1.2. 单元测试的任务

1. 模块接口测试

2. 模块局部数据结构测试

3. 模块边界条件测试

4. 模块中所有独立执行通路测试

5. 模块的各条错误处理通路测试

4.1.3. 单元测试的工具

CppUnit，这是C++单元测试工具的鼻祖，免费的开源单元测试框架。 想了解CppUnit的朋友，建议读一下Cpluser所作的《CppUnit测试框架入门》。

C++Test，这是Parasoft公司的产品。C++Test是一个功能强大的自动化C/C++单元级测试工具， 可以自动测试任何C/C++函数、类，自动生成测试用例、测试驱动函数或桩函数， 在自动化的环境下极其容易快速的将单元级的测试覆盖率达到100%。

对于Java语言的单元测试，JUnit无疑是首选产品。它是由Erich Gamma和Kent Beck编写的一个回归测试框架（Regression Testing Framework）。 JUnit是一套框架（Framework）。对于JUnit的应用，简单来说，继承其TestCase类，就可以进行自动测试了。

4.2. ClearWork单元测试框架

利用JUnit可以很方便的对业务逻辑层进行单元测试，对于ClearWork来说，最简单的只需要继承junit.framework.TestCase 类就可以了，因为ClearWork的Spring Bean加载工具可以直接从classpath中的配置文件加载Bean。

但是，针对控制层（Controller，例如：Struts Action, Servlet）、 数据操作对象（DAO, Data Access Object）、 数据库单元（Database Unit）的测试， 单纯的JUnit就显得不足了，因此，ClearWork针对这三种场景提供了自己的单元测试支持—— 抽象出相应的基类供测试单元继承，就像使用JUnit时继承 TestCase 一样。它们分别是：

- net.sf.clearwork.core.testcase.AbstractActionTestCase ——作为对 strutstest 开源项目的扩展；测试Servlet/Struts Action的单元测试基类；自动加载了web / spring / struts的配置。Example:

```
setRequestPathInfo("/login.do");
addRequestParameter("username","admin");
addRequestParameter("password","clearwork");
actionPerform();
verifyForward("success");
assertEquals("clearwork",(String) getSession().getAttribute("admin"));
verifyNoActionErrors();
```

- net.sf.clearwork.core.testcase.AbstractDaoTestCase —— 继承自Spring Framework的AbstractTransactionalDataSourceSpringContextTests 。作为ClearWork/Spring数据操作对象(DAO)单元测试的基类， 它可以自动加载ClearWork的数据源配置（Spring Bean Name = “dataSource” ）。

- net.sf.clearwork.core.testcase.AbstractDBUnitTestCase ——对 dbunit 开源项目的扩展；作为基于ClearWork数据源单元测试的基类， 它可以自动加载ClearWork的数据源配置（Spring Bean Name = “dataSource” ）。

第 5 章 ClearWork应用架构——持久化层

MVC Model Layer

5.1. JDBC数据源支持

在 spring.xml 中利用 dataSource 作为基础的数据源/连接池； transactionManager 作为默认的JDBC事务管理器，被用来管理基于JDBC数据源的业务(Business)层的事务。

利用Spring2.0提供的AOP自动代理的底层架构（基于 AspectJ 切点语言），我们可以很方便的对特定的业务类、方法进行事务拦截、包装等处理， 例如我们可以这样定义一个对于JDBC事务的拦截器：

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
                <tx:method name="insert*" />
                <tx:method name="delete*" />
                <tx:method name="update*" />
                <tx:method name="*" read-only="true" />
        </tx:attributes>
</tx:advice>
```

很明显，它对于"insert", "delete", "update"开头的方法进行事务包装。 我们可以通过下面的配置使其绑定于特定的业务服务类：

```
<aop:config>
        <aop:advisor pointcut="execution(* *..IUserService.*(..))"
                advice-ref="txAdvice" />
</aop:config>
```

以上的应用示范位于ClearWork Sample的 spring-sample.xml ，以供参考 。

5.2. Hibernate数据源支持

作为轻量级ORM数据管理框架的优秀代表，Hibernate被ClearWork良好的支持和集成。同样的，需要在spring.xml 中利用 dataSource 作为基础的数据源/连接池；由于各模块需要定义自己的ORM文件(.hbm)，因此 sessionFactory 在全局配置中被声明为抽象的（abstract）， 您需要在模块级的Spring配置中继承这个sessionFactory，指定ORM配置(.hbm文件)，例如：

```
<bean id="sampleSessionFactory" parent="sessionFactory">
        <property name="mappingResources">
                <list>
                        <value>
                                net\sf\clearwork\sample\domain\hibernate\CContact.hbm.xml
                        </value>
```

```
                        <value>
                                net\sf\clearwork\sample\domain\hibernate\CNews.hbm.xml
                        </value>
                        <value>
                                net\sf\clearwork\sample\domain\hibernate\CUser.hbm.xml
                        </value>
                </list>
        </property>
</bean>
```

相应的，Hibernate的事务管理器也需要在相应的模块级Spring配置中指定：

```
<!-- Transaction manager for a single Hibernate SessionFactory (alternative to JTA) -->
<bean id="hibernateTransactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sampleSessionFactory" />
</bean>
```

接下来的情况与JDBC基本相同。利用Spring2.0提供的AOP自动代理的底层架构（基于 AspectJ 切点语言），我们可以很方便的对特定的业务类、方法进行事务拦截、包装等处理， 例如我们可以这样定义一个对于Hibernate事务的拦截器：

```
<tx:advice id="txAdviceHibernate"
        transaction-manager="hibernateTransactionManager">
        <tx:attributes>
                <tx:method name="insert*" />
                <tx:method name="delete*" />
                <tx:method name="update*" />
                <tx:method name="*" read-only="true" />
        </tx:attributes>
</tx:advice>
```

很明显，它对于"insert", "delete", "update"开头的方法进行事务包装。 我们可以通过下面的配置使其绑定于特定的业务类：

```
<aop:config>
        <aop:advisor pointcut="execution(* *..IUserService.*(..))"
                advice-ref="txAdviceHibernate" />
</aop:config>
```

以上的应用示范位于ClearWork Sample的 spring-sample.xml ，以供参考 。

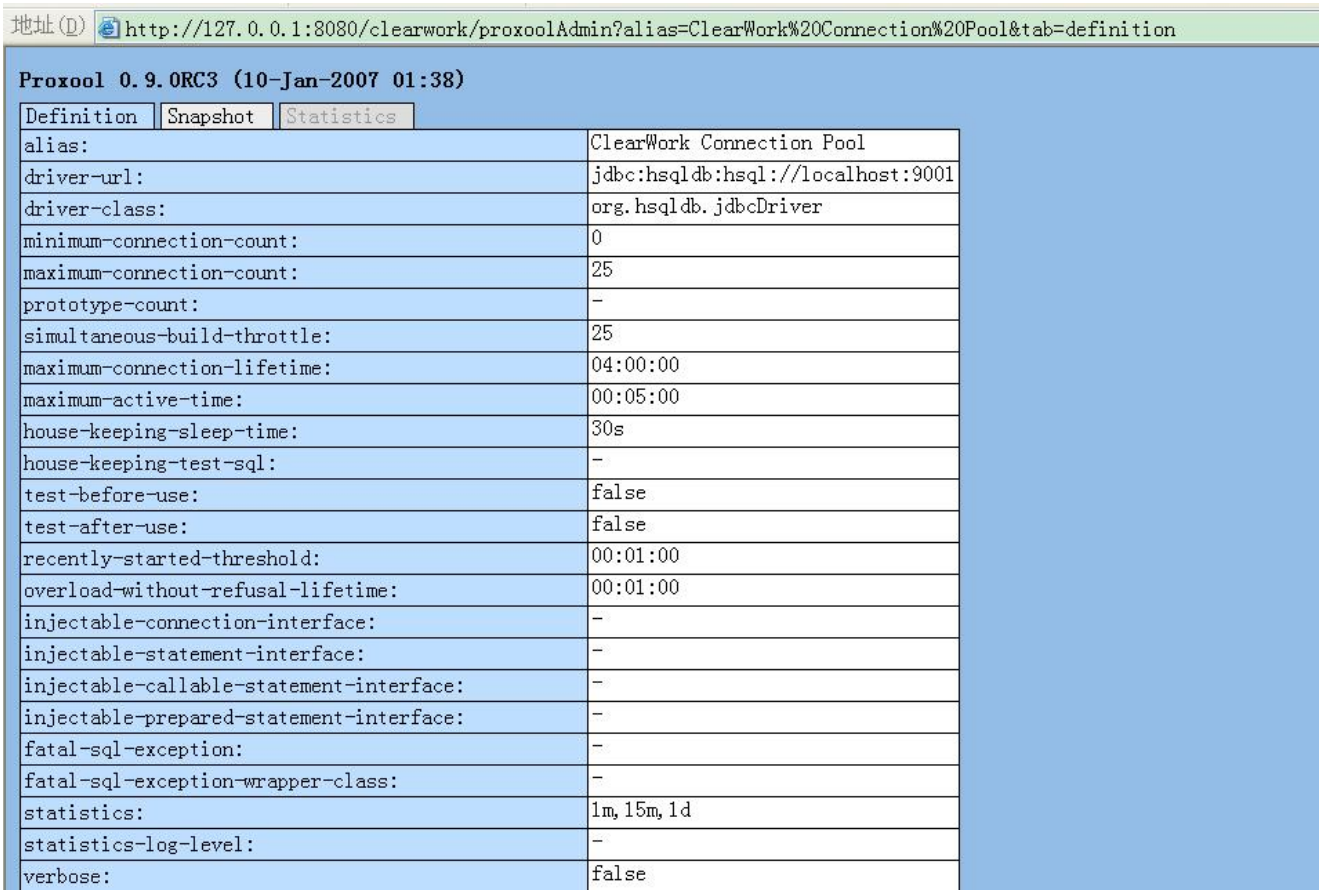
5.3. Proxool数据库连接池

Proxool这个开源项目可以提供对您选择的任何数据驱动的连接池封装。它可以非常简单的移植到现存的代码中。 完全可配置，能够有效、及时的回收连接资源；能够智能的分配、重用数据库连接资源。快速，成熟，健壮。可以透明地为您现存的JDBC驱动程序增加连接池功能。 ClearWork对其进行了有效的集成，使之能方便被使用在Spring框架中，作为数据源（DataSource）代码如下：

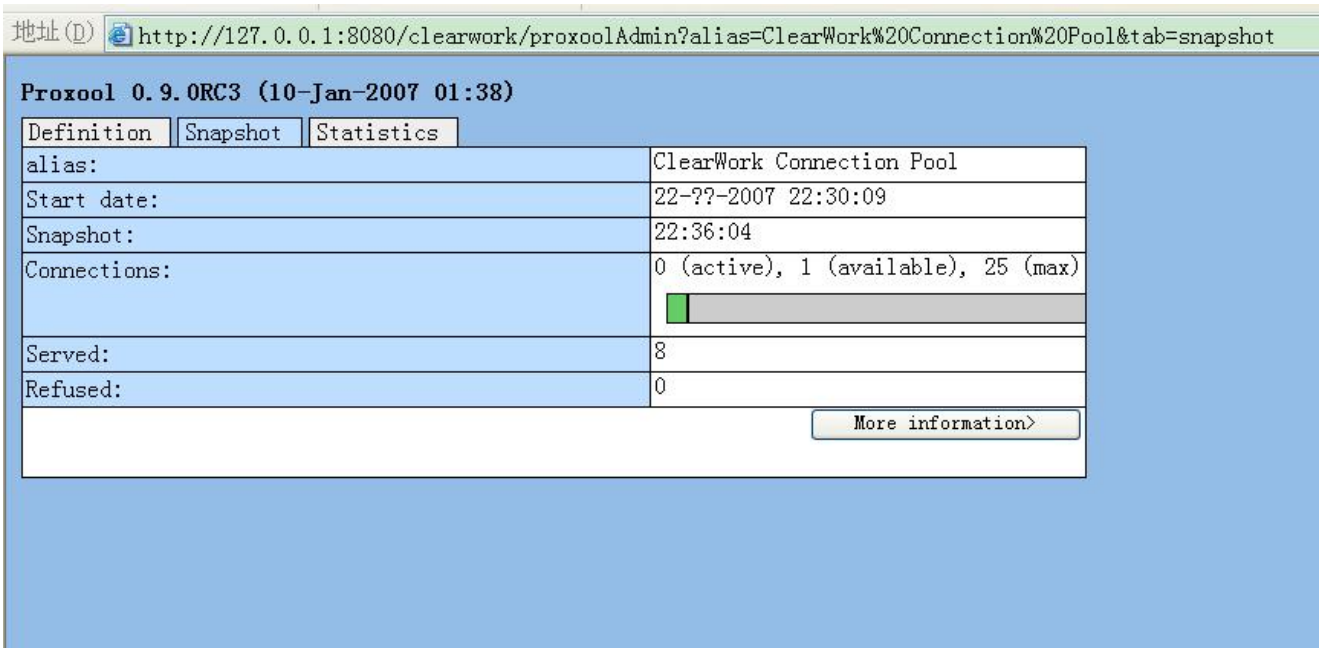
```
<bean id="dataSource" lazy-init="false"
        class="org.logicalcobwebs.proxool.ProxoolDataSource">
        <property name="alias" value="ClearWork Connection Pool" />
        <property name="statistics" value="1m,15m,1d" />
        <property name="simultaneousBuildThrottle" value="25" />
        <property name="maximumConnectionCount" value="25" />
        <property name="driver" value="${jdbc.driverClassName}" />
        <property name="driverUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
        <property name="delegateProperties">
                <value>user=${jdbc.username}</value>
                <!--value>user=sa,password=sa</value-->
        </property>
</bean>
```

注意： delegateProperties 这个属性的配置是必须的，你需要在这里再次指定连接数据库的user或user/password， 它是为了修正Proxool现存版本的一个小Bug。

Proxool为您提供了一个基于Web的管理控制台，能够方便的管理数据库连接池。



Proxool/ClearWork数据库连接池管理控制台_效果1



Proxool/ClearWork数据库连接池管理控制台_效果2

此管理控制台是一个Servlet，需要在web.xml中配置：

```
<servlet>
        <servlet-name>proxoolAdmin</servlet-name>
        <servlet-class>
                net.sf.clearwork.persistent.proxool.AdminServlet
        </servlet-class>
</servlet>
```

第 6 章 ClearWork应用架构——业务容器层

MVC Model Layer

6.1. 基于Spring2.0+的容器框架

基于Spring2.0+强大的IOC/AOP框架管理容器，ClearWork为您提供众多的企业应用架构中必需的、标准的业务组件。 在 spring.xml 中的全局业务组件Bean配置介绍：

- propertyConfigurer - 管理加载资源属性文件。
- messageSource - 国际化资源文件管理器，建议使用messages_xx.properties命名您的国际化资源， 例如：messages_cn.properties；支持使用<list>标签为basenames属性注入多个国际化资源文件。

ClearWork所有的配置文件（Spring, Hibernate, WSDD...）都存在于classpath的 config 目录中，因此，不需要做额外的载入操作，我们就可以通过 net.sf.clearwork.core.utils.base.SpringContextUtils 从指定的位置自动的载入Spring配置文件，做"getBean"之类的操作。

国际化资源相关文件都存在于classpath的 i18n 目录中。ClearWork对中文采用UTF-8编码，因此 make_messages.bat 和 make_exception.bat 的作用都是将相应的xxxx.txt中文原始文件转换为UTF-8编码的xxxx_cn.properties资源。 2cn.bat可以将指定的UTF-8编码的资源文件反转回GB2312中文编码的 2cn.txt 文件，可以用来验证资源的正确性。

6.2. 基于Apache Axis2的Web服务框架

请参考： 第 8 章 ClearWork SOA框架——Apache Axis2

第 7 章 ClearWork应用架构——Web层

MVC Controller and View Layer

7.1. Apache Struts支持

MVC Controller Layer

ClearWork的Struts配置文件位于classpath中， 随系统启动自动随需载入，不需要在配置文件中显示指定，需要在 web.xml 中进行如下配置：

```
<!--
        - 自动载入位于：
        - ClearWork\trunk\core\main\resources\config\struts.xml (全局配置)
        - ClearWork\trunk\core\main\resources\config\struts\ (模块配置)
        - 的配置文件
        -
        - A web app can just contain one such servlet.
        - If you need multiple namespaces, use Struts' module mechanism.
-->
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
                net.sf.clearwork.web.struts.StrutsActionServlet
        </servlet-class>
        <load-on-startup>3</load-on-startup>
</servlet>
```

net.sf.clearwork.web.struts.StrutsActionServlet 继 承自 org.apache.struts.action.ActionServlet

7.2. Ajax支持

MVC View Layer

ClearWork提供了一套轻量级的 Ajax 封装，可以方便的、自由的运用Ajax开发您的前端应用。支持Ajax+Struts的组合。

ajax_template.js 提供了如下script函数供Ajax应用调用：

1. ajaxGet (url, callbackFn) - 提供同步的、XML返回的、GET方式的Ajax调用；参数分别为：请求的URL、回调函数入口 （此2参数下同）。

2. ajaxGet2 (url, callbackFn, isSynchro, isXML) - 提供更加灵活的GET方式的Ajax调用；后2个参数分别为：是否同步（boolean, false为异步）、是否为XML返回（boolean, false为文本方式返回）

3. ajaxPost (url, callbackFn, formParamArr) - 提供同步的、XML返回的、POST方式的Ajax调用；最后一个参数为需要POST的表单域名称数组（array类型）。

4. ajaxPost2 (url, callbackFn, formParamArr, formParamValueArr, isSynchro, isXML) - 提供更加灵活的POST方式的Ajax调用；formParamValueArr参数为需要POST的表单域值数组（array类型，可以为null）。

对于Struts+Ajax的应用，必须要指定Forward作为输出缓冲，在ClearWork框架中已经预置，在全局配置 struts.xml 中：

```
<global-forwards>
        <forward name="ajax_outxml"
                path="/WEB-INF/global/ajax_outxml.jsp" />
        <forward name="ajax_outext"
                path="/WEB-INF/global/ajax_outext.jsp" />
</global-forwards>
```

具体的Struts Action写法，可以参照ClearWork Sample net.sf.clearwork.sample.ajax.SampleAjaxTemplateAction 。对于Servlet方式的应用，本文档不做叙述，可以参考通用的资料。

第 8 章 ClearWork SOA框架——Apache Axis2

ClearWork力求为基于Web Service技术的SOA架构整合完善的工具和框架，并提供全面的支持。

ClearWork Web Service框架的全局配置为 spring-webservice.xml ，其中的配置包括：Web Service事务资源池、Web Service事务模版、Web Service for JDBC事务管理器、Web Service事务服务的客户端配置。

8.1. Axis2介绍

Axis2 是新一代Apache Axis。Axis2具有更强的灵活性并可扩展到新的体系结构。Axis2 基于新的体系结构进行了全新编写。Axis2的特性包括：

- 采用名为 AXIOM（ AXI s2 O bject M odel，Axis 对象模型）的新核心 XML 处理模型
- 支持 In-Only 和 In-Out 消息交换模式 (MEP)
- 阻塞和非阻塞客户端 API（应用程序编程接口）
- 支持内置的 Web 服务寻址 (WS-Addressing)
- 支持 XML Beans 数据绑定
- 新部署模型
- 支持超文本传输协议 (HTTP)、简单邮件传输协议 (SMTP) 、传输控制协议 (TCP)等传输协议和REST (Representational State Transfer)结构

具体信息可以访问 Axis2官方网站 [http://ws.apache.org/axis2] 获取。

8.2. ClearWork对Axis2的扩展 - 基于WSDD描述文件的服务部署

在第一代的 Axis 中，服务的暴露和加载主要是依赖WSDD (Web Service Define Document)文档进行的。 Axis2本身支持以 aar 存档的形式部署和加载Web服务（支持热部署）。

现在，ClearWork将Axis通过WSDD服务描述文件部署加载Web服务的特性带入了Axis2！ 您只需在特定的目录（默认为： classpath:config/wsdd/ ）中利用 *.wsdd 描述您的业务服务类型，即可将其暴露为Web Service，其描述规范完全遵循Axis2 aar存档中 META-INF/services.xml 的规范。通过这种机制可以自动的将您应用中任何符合Service规范的类型暴露为Web Service。

相应的，您需要在 web.xml 中预置ClearWork的Axis2ServiceServlet（它继承自org.apache.axis2.transport.http.AxisServlet ）

```
<servlet>
        <servlet-name>AxisServlet</servlet-name>
        <display-name>
                Apache-Axis2 and ClearWork-Contribute Servlet
        </display-name>
        <servlet-class>
```

```
                net.sf.clearwork.service.axis2.server.web.deployment.Axis2ServiceServlet
        </servlet-class>
        <init-param>
                <param-name>axis2.xml.path</param-name>
                <param-value>classpath*:config/axis2.xml</param-value>
                <!--<param-name>axis2.xml.url</param-name>-->
                <!--<param-value>http://localhot/myrepo/axis2.xml</param-value>-->
                <!--<param-name>axis2.repository.path</param-name>-->
                <!--<param-value>/WEB-INF</param-value>-->
                <!--<param-name>axis2.repository.url</param-name>-->
                <!--<param-value>http://localhot/myrepo</param-value>-->
                <!--<param-name>axis2.wsdd.path</param-name>-->
                <!--<param-value>classpath*:config/**/wsdd/*.wsdd.xml</param-value>-->
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

默认的，您的WSDD服务描述文件都应位于 classpath*:config/**/wsdd/*.wsdd.xml ，WSDD文件的扩展名应遵循 *.wsdd.xml 规则。

8.3. ClearWork对Axis2的扩展 - Web Service服务及其数据定义的强验证工具

如何验证您的服务定义（WSDD）是否正确？如何验证您的业务服务类是否完全符合Web Service标准而能被预期的访问？这两个问题的异常如果在Runtime（运行时）被强制抛出将是很令人迷惑的， 特别是后者，有时候它并不表现为“异常”，而是给您无法预期的返回结果。

现在，ClearWork提供了 net.sf.clearwork.service.axis2.server.ServiceValidater 工具，它可以精密的验证您的服务描述(WSDD)、服务类(Class)、甚至服务类的输入输出数据类型(POJO原则)， 如果发现了可能问题，将会在加载时抛出相应的异常或打印明确的Log4j Debug提示。

ServiceValidater工具的用法非常简单：

```
new ServiceValidater(new FileInputStream(Your_WSDD_File)).validate();
```

验证通过返回true，否则返回false。

ClearWork系统默认对所有定义的Web Service服务采用此工具进行加载时的验证，具体情况可以参见第 8.2 节 “ClearWork对Axis2的扩展 - 基于WSDD描述文件的服务部署” 。

8.4. ClearWork对Axis2的扩展 - Web Service客户端辅助增强工具集

- net.sf.clearwork.service.axis2.client.support.CallbackSupport - 我们知道Aixs2提供的令人激动的新特性之一就是它可以很方便的支持客户端的异步Web Service回调，这是通过继承

org.apache.axis2.client.async.Callback 回调抽象类实现的，现在ClearWork提供了进一步的抽象—— CallbackSupport 。它使用execute()抽象方法替代了原来的onComplete()抽象方法，来执行异步Web Service调用的后续处理，与onComplete()方法不同的是，execute()方法提供的参数Object[] returnObjects 是经过反序列化处理生成的真正的应答对象。

同时，CallbackSupport简化了错误情况（onError()）的处理； execute()抽象方法支持一个布尔型的返回值以确认业务逻辑的正确执行。

- net.sf.clearwork.service.axis2.client.support.RPCServiceClientSupport - 这个类通过继承org.apache.axis2.rpc.client.RPCServiceClient 为您提供更加简单清晰的 动态执行 Web Service的客户端工具， 它增强了“无返回值同步执行”、“异步执行”、“有返回值同步执行”等常用的动态客户端操作方法。

8.5. ClearWork对Axis2的扩展 - 基于自定义策略的Web Service事务控制

大部分情况下，Web Service的SOAP消息是通过Http协议来传送的，正如我们所知的， 由于Http协议的无状态特性，基于其的SOAP消息很容易的穿过防火墙，但一个副作用就是无法保证Web Service客户端和服务端的一致性，即Web Service事务不可控制。

现在，ClearWork提供了一套简单有效的基于Spring配置的Web Service事务控制框架，可以在一定范围内解决Web服务客户端与服务端事务不可控制的问题。 基本原理就是扩展Spring的JDBC及Hibernate事务管理框架：在服务端执行完毕后，不立即提交事务， 而是将只支持同一线程的Spring事务资源放入一个“事务资源池”中（这个事务资源池需要实现net.sf.clearwork.service.axis2.server.transaction.IWSTransactionSynchronizationPool 接口。可以由用户自己实现，当然，ClearWork也提供默认的实现DefaultWSTransactionSynchronizationPool ），相当于远程“挂起”事务，然后，利用Axis2的异步执行Web Service客户端的特性，执行回调方法中客户端部分的事务，那么，将会有以下几种情况出现：

1. 客户端事务提交成功 - 则客户端会自动通知服务端提交“事务资源池”中相应的事务，两方面事务成功提交。
2. 客户端事务提交失败 - 无论是由于异常爆发或业务逻辑未通过，ClearWork都会自动的通知服务端“回滚”事务，以保证一致性。
3. 客户端与服务端失去联系 - 这种场景最常见的情况就是网络瞬断，那么，由于接收不到客户端的消息， 服务端被“挂起”的事务在指定的等待时间后，则会触发 事务超时策略 ，根据配置对相应的事务进行“回滚”或“提交”的补偿处理。

事务超时策略：作为 spring.xml 中全局配置“Web服务事务模板” webServiceTransactionTemplate 的 serviceTimeout 属性被配置，格式为

```
rollback
-
[second]
```

或

```
commit
_
[second]
```

。例如：rollback_4表示，4秒后执行回滚事务的超时策略。

4. 客户端事务成功提交

可以想象，如果在服务端的事务操作一开始就失败了，那么服务端会自动的立刻通知客户端，保证客户端的事务根本不会被执行。

从应用实现方面来说，如果需要ClearWork框架对您的Web Service事务进行控制，在开发中需要做到以下几点：

1. 服务端实现类需要继承 net.sf.clearwork.service.axis2.server.service.AbstractTransactionServiceSupport 。
2. 服务端返回的应答类型必须为 net.sf.clearwork.service.axis2.dto.AbstractTransactionDTO 的子类，ClearWork有两个默认实现： StringTransactionDTO 和 VoidTransactionDTO 。
3. 根据持久化实现的不同，选择配置 WSDataSourceTransactionManager for JDBC 或 WSHibernateTransactionManager for Hibernate，可以参考ClearWork Sample Resource - spring-sample.xml 。
4. 需要在 spring-webservice.xml 中正确配置 transactionServiceClient ，告知事务客户端正确的服务端地址和端口。
5. 客户端必须是异步执行模式，同时必须使用Web Service事务支持的回调抽象类型 net.sf.clearwork.service.axis2.client.support.CallbackTransactionSupport 。

具体的，如何实现ClearWork Web Service事务控制，建议您参考ClearWork Sample：

- 服务端示范 - net.sf.clearwork.sample.webservice.server.SampleUserTransactionService 和 net.sf.clearwork.sample.webservice.server.SampleUserHibernateTransactionService
- 客户端示范 - net.sf.clearwork.sample.webservice.client.SampleUserTransactionClient
- 配置示范 - 位于 spring-sample.xml 中

8.6. ClearWork对Axis2的扩展 - 基于元数据（Annotation）描述的Web Service服务注册机制

一般来说，一个普通的业务服务类型（Business Service）暴露为Web Service供他人访问， 通过诸如Axis2之类的“Web服务引擎”是很容易做到的，但仍然需要配置一些服务描述等等，而且， 并没有

一种强验证机制能在部署时检测原始的业务服务类型是否完全适合暴露为Web Service，例如：

- 请求/应答（Request/Response）参数是否符合POJO原则。
- 服务描述的XML是否符合“Web服务引擎”定义的规范。
- 对于 第 8.5 节 “ClearWork对Axis2的扩展 - 基于自定义策略的Web Service事务控制” 的验证。
- ……

现在，ClearWork框架提供了基于元数据（Annotation）描述的Web Service服务注册机制以解决上述问题，简化Web服务开发，强化Web服务验证。通过元数据接口 net.sf.clearwork.service.axis2.annotation.AsWebService 将特定的业务服务类型暴露。

AsWebService的参数： description 参数为服务描述，必需； schemaNamespace 如果不指定，则为包名的倒序；其余的参数均有默认值，可参考示例或Java Doc。

同时，需要在Spring中配置 net.sf.clearwork.service.axis2.server.ServiceRegister ，在其属性中声明暴露的服务类型，例如：

```
<!-- web service Register -->
<bean id="serviceRegister"
        class="net.sf.clearwork.service.axis2.server.ServiceRegister"
        lazy-init="true">
        <property name="services">
                <map>
                        <entry key="SampleHelloWorldService">
                                <value>
                                        net.sf.clearwork.sample.webservice.server.SampleHelloWorldService
                                </value>
                        </entry>
                </map>
        </property>
</bean>
```

具体的使用，可以参考 ClearWork 示例 spring-sample.xml 以及 net.sf.clearwork.sample.webservice.server.SampleHelloWorldService 和 net.sf.clearwork.sample.webservice.server.SampleUserTransactionService 。

第 9 章 ClearWork开发示例

ClearWork的一个重要目标是：提供整套企业应用各个方面的简明开发示范，供开发者快速上手。

目前主要包括：

- Ajax示例
- 持久化层示例
- 值注入工具示例
- Web Service示例