目录

[第1章 工程创建 1](#_Toc418498816)

[1.1 创建动态Web工程 1](#_Toc418498817)

[1.2 将AppFrame加入到工程 2](#_Toc418498818)

[第2章 登陆 3](#_Toc418498819)

[2.1 LoginFilter 3](#_Toc418498820)

[2.2 ExpireResponseHeaderFilter 3](#_Toc418498821)

[2.3 BaseListenerServer 4](#_Toc418498822)

[2.4 CentralControlServle 4](#_Toc418498823)

[2.5 web.xml配置 4](#_Toc418498824)

[2.5 登录功能-非SSO 5](#_Toc418498825)

[第3章 业务开发 5](#_Toc418498826)

[3.1开发业务类 5](#_Toc418498827)

[3.2 发送请求方式 5](#_Toc418498828)

[3.3 原理 6](#_Toc418498829)

[第4章 数据层开发 6](#_Toc418498830)

[4.1 数据源配置 6](#_Toc418498831)

[4.2 Configtools的使用 9](#_Toc418498832)

[4.3 数据容器 11](#_Toc418498833)

[4.3.1 BOGruceeBookEngine 11](#_Toc418498834)

[4.3.2 DataContainerInterface 11](#_Toc418498835)

[4.3.3 ObjectType 12](#_Toc418498836)

[4.4 开发Dao层-保存/修改 12](#_Toc418498837)

[4.5 DAO层调用 13](#_Toc418498838)

[4.6 开发Dao层-查询 14](#_Toc418498839)

[4.7 无事务访问数据库 14](#_Toc418498840)

[第5章 服务开发 14](#_Toc418498841)

[5.1 服务开发 14](#_Toc418498842)

[5.2 服务调用方法 15](#_Toc418498843)

[5.3 FAQ 15](#_Toc418498844)

[第6章 分表 16](#_Toc418498845)

[6.1 分表配置 16](#_Toc418498846)

[6.2 分表转换 16](#_Toc418498847)

[6.3 分表查询 17](#_Toc418498848)

[第7章 中心、路由 18](#_Toc418498849)

[7.1 中心配置 18](#_Toc418498850)

[7.2 中心类型实现类 18](#_Toc418498851)

[7.3 中心路由规则 19](#_Toc418498852)

[7.4 总结 19](#_Toc418498853)

[7.5 前台请求设置中心 19](#_Toc418498854)

[第8章 缓存 20](#_Toc418498855)

[8.1 缓存开发 20](#_Toc418498856)

[8.2 缓存配置 21](#_Toc418498857)

[8.3 缓存调用 21](#_Toc418498858)

[第9章 AppFrameDemo下载 21](#_Toc418498859)

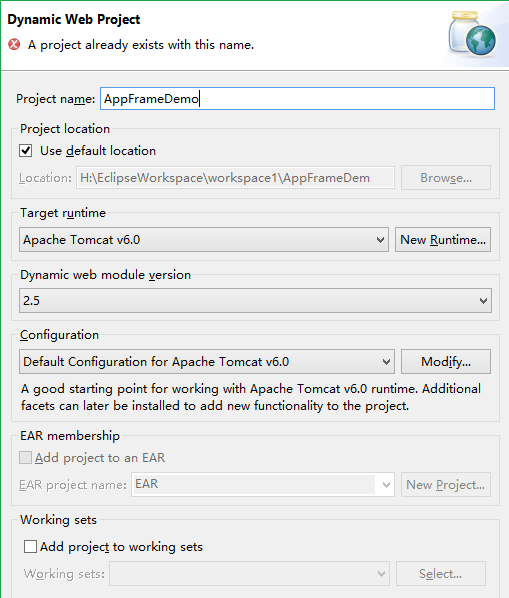
# 第1章 工程创建

可以使用Appframe进行Web开发或者后台开发，只需要依赖相应的jar包以及配置文件。本文将以Web工程AppFrameDemo为例，介绍如何使用AppFrame进行开发。

## 创建动态Web工程

需要注意两点：

1. AppFrame依赖JavaEE 5实现，并且只能使用JDK1.5编译；因此，Dynamic web module version只能选择2.5版本；
2. Dynamic web module version限定了使用的Servlet规范版本、JavaEE规范版本；其对应关系，参见图2。



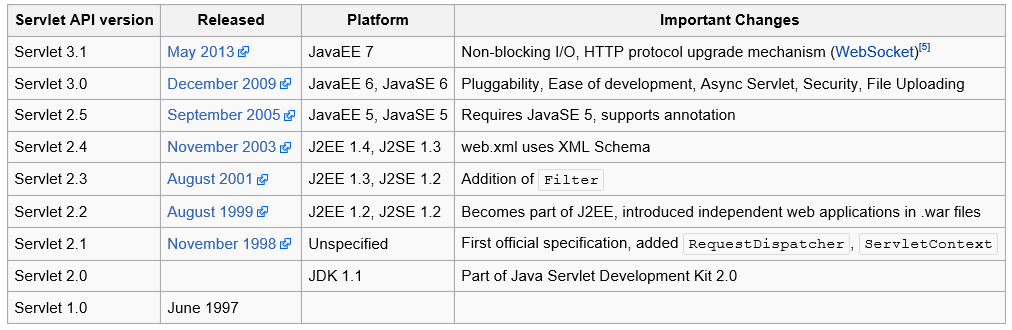
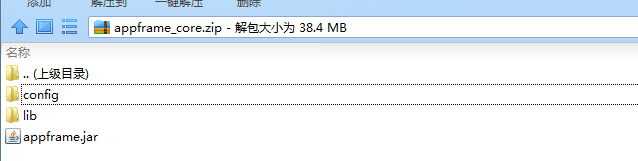


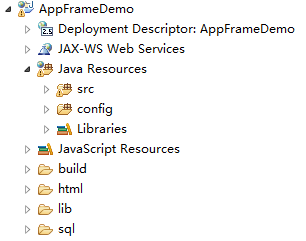
图2 Servlet规范和JavaEE规范的对应关系

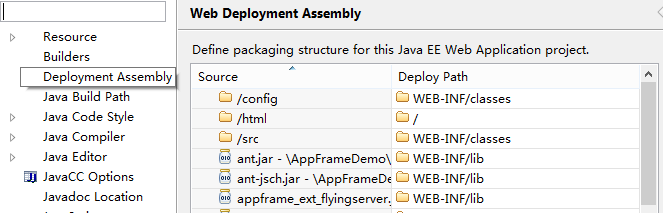
## 将AppFrame加入到工程

AppFrame的发布包如下：



将Appframe加入到工程中，并配置Demployment Assembly以及BuildPath：





# 第2章 登陆

AppFrame提供了几个重要的过滤器、Listener以及Servlet，我们来认识一下

## 2.1 LoginFilter

该过滤器主要实现下面的功能：

* 验证会话
* 限制某些url只能通过get方式访问
* 非法字符的过滤
* 鉴权

要实现完善的登录功能，AppFrame提供了用户管理和鉴权的切入点，可以自定义实现。

## 2.2 ExpireResponseHeaderFilter

设置gif、jpg等资源的响应过期时间。

|  |
| --- |
| response.addHeader("Cache-Control", "Private");  response.addDateHeader("Expires", timeMillis); |

在AIConfig.xml中的配置：

|  |
| --- |
| <ConfigKind name=*"HttpExpireConfig"*>  <ConfigItem name=*"EXPIRE\_ON\_OFF\_SWITCH"* remarks=*"打开和关闭超时设置功能(ON/OFF),如无此项默认为ON"*>OFF</ConfigItem>  <ConfigItem name=*"\*.gif"* remarks=*"访问后8个小时内客户端有效"*>A28800</ConfigItem>  <ConfigItem name=*"\*.jpg"* remarks=*"访问后8个小时内客户端有效"*>A28800</ConfigItem>  <ConfigItem name=*"\*.png"* remarks=*"访问后8个小时内客户端有效"*>A28800</ConfigItem>  <ConfigItem name=*"\*.js"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"\*.vbs"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"\*.swf"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"\*.css"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"/jsv2/Globe\_v2.jsp"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"/jsv2/i18n/AILocale.jsp"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"/jsv2/AppFrameCss.jsp"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"/webframe/common/cookie.jsp"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  <ConfigItem name=*"/webframe/css/AIFrameDBCSS\_css.jsp"* remarks=*"访问后2个小时内客户端有效"*>A7200</ConfigItem>  </ConfigKind> |

## 2.3 BaseListenerServer

主要做Session销毁后的处理：用户登出、缓存清理。

## 2.4 CentralControlServle

业务请求/business/\*的分发。

## 2.5 web.xml配置

来看一下AppFrameDemo中web.xml的配置：

|  |
| --- |
| <display-name>AppFrameDemo</display-name>  <filter>  <filter-name>DefaultBaseFilter</filter-name>  <filter-class>com.ai.appframe2.web.filter.LoginFilter</filter-class>  </filter>    <filter-mapping>  <filter-name>DefaultBaseFilter</filter-name>  <url-pattern>\*.jsp</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>DefaultBaseFilter</filter-name>  <url-pattern>/business/\*</url-pattern>  </filter-mapping>  <listener>  <listener-class>com.ai.appframe2.web.BaseListenerServer</listener-class>  </listener>    <servlet>  <servlet-name>CentralControl</servlet-name>  <servlet-class>com.ai.appframe2.web.action.CentralControlServlet</servlet-class>  </servlet>    <servlet-mapping>  <servlet-name>CentralControl</servlet-name>  <url-pattern>/business/\*</url-pattern>  </servlet-mapping> |

## 2.5 登录功能-非SSO

在正确配置了web.xml后，就很容易实现登入、登出以及会话功能了。

1、访问任何需要鉴权的页面，如果没有登录，跳转到登录页面—通过LoginFilter实现

2、访问index.jsp页面，如果已经登录，则跳转到main.jsp

3、除登录页面外的页面，均提供登出按钮。

# 第3章 业务开发

AppFrameDemo的登入、登出使用的是原生的Servlet；Appframe提供了一种方便的方式开发业务，无需在web.xml中配置。

## 3.1开发业务类

继承BaseAction，实现相应的业务方法，示例如下：

|  |
| --- |
| /\*\*  \* 业务操作  \* **@author** 孙德东(24204)  \*/  **public** **class** BookAction **extends** BaseAction{  **public** **void** addBook(HttpServletRequest req, HttpServletResponse resp) **throws** Exception {  ...  } |

## 3.2 发送请求方式

|  |
| --- |
| $("#book\_submit\_button").click(**function**(){  **var** postData = "name=" + bookName + "&author=" + author + "&publisher=" + publisher + "&remarks=" + remarks;  $.post("<%=request.getContextPath() %>/business/grucee.demo.action.BookAction?action=addBook", postData,  **function**(data){  //abc  **if** (data.result == 'T') {  alert("add book success.");  } **else** {  alert("add book fail:" + data.msg); // John  }  }, "json");  }); |

## 3.3 原理

Servlet：com.ai.appframe2.web.action.CentralControlServlet通过反射实现对所有/business/\*请求的分发。

# 第4章 数据层开发

## 4.1 数据源配置

我们来看一个数据源的最简单的配置：

|  |
| --- |
| <clazz name=*"com.ai.appframe2.complex.datasource.impl.LocalMutilDataSourceImpl"*>  </clazz>  <pool name=*"base"* primary=*"true"*>  <property name=*"driverClassName"* value=*"oracle.jdbc.OracleDriver"*/>  <property name=*"url"* value=*"jdbc:oracle:thin:@10.11.20.84:1521:njdev84"*/>  <property name=*"username"* value=*"oppf"*/>  <property name=*"password"* value=*"oppf"*/>  <property name=*"initialSize"* value=*"0"*/>  <property name=*"maxActive"* value=*"2"*/>  <property name=*"maxIdle"* value=*"2"*/>  </pool> |

生产环境上的高级配置

|  |
| --- |
| <!-- maxReturnRows表示最大返回结果集记录条数 -->  <clazz name=*"com.ai.appframe2.complex.datasource.impl.LocalMutilDataSourceImpl"*>  <property name=*"tableName"* value=*"cfg\_db\_acct"*/>  <property name=*"urlTableName"* value=*"cfg\_db\_url"*/>  <property name=*"relatTableName"* value=*"cfg\_db\_relat"*/>    <property name=*"prefetch"* value=*"true"*/>  <property name=*"prefetch.driverClassName"* value=*"oracle.jdbc.OracleDriver"*/>  <property name=*"prefetch.url"* value=*"jdbc:oracle:thin:@(DESCRIPTION\_LIST=(LOAD\_BALANCE=off)(FAILOVER=on)(DESCRIPTION=(ADDRESS\_LIST=(LOAD\_BALANCE=OFF)(FAILOVER=ON)(ADDRESS=(PROTOCOL=TCP)(HOST=zjpub1.yw.zj.chinamobile.com)(PORT=1521)))(CONNECT\_DATA=(SERVICE\_NAME=zjpub)(INSTANCE\_NAME=zjpub1)(FAILOVER\_MODE=(TYPE=session)(METHOD=basic)(RETRIES=4)(DELAY=1))))(DESCRIPTION=(ADDRESS\_LIST=(LOAD\_BALANCE=OFF)(FAILOVER=ON)(ADDRESS=(PROTOCOL=TCP)(HOST=zjpub2.yw.zj.chinamobile.com)(PORT=1521)))(CONNECT\_DATA=(SERVICE\_NAME=zjpub)(INSTANCE\_NAME=zjpub2)(FAILOVER\_MODE=(TYPE=session)(METHOD=basic)(RETRIES=4)(DELAY=1)))))"*/>  <property name=*"prefetch.username"* value=*"aiams"*/>  <property name=*"prefetch.password"* value=*"{RC2}dz5+rB4DRbd7"*/>  <property name=*"prefetch.initialSize"* value=*"1"*/>  <property name=*"prefetch.maxActive"* value=*"1"*/>  <property name=*"prefetch.maxIdle"* value=*"1"*/>  </clazz>  <!--根据base取数据库配置，然后以本配置中的覆盖除base以外的连接池的配置-->  <!--默认 type=dbcp db=oracle -->  <pool name=*"base"* primary=*"true"* type=*"dbcp"* db=*"oracle"* init=*"true"* isAdvanceUrl=*"true"*>  <property name=*"initialSize"* value=*"2"*/>  <property name=*"maxActive"* value=*"5"*/>  <property name=*"maxIdle"* value=*"1"*/>  <property name=*"minIdle"* value=*"1"*/>  <property name=*"maxWait"* value=*"2000"*/>  <property name=*"connectionProperties"* value=*"oracle.net.CONNECT\_TIMEOUT=120000"*/>  <!--连接池有效性检查-->  <property name=*"validationQuery"* value=*"select 1 from dual"*/>  <property name=*"testWhileIdle"* value=*"true"*/>  <property name=*"testOnBorrow"* value=*"false"*/>  <property name=*"testOnReturn"* value=*"false"*/>  <!--最多20秒检查到连接失效-->  <property name=*"timeBetweenEvictionRunsMillis"* value=*"20000"*/>  <property name=*"minEvictableIdleTimeMillis"* value=*"-1"*/>  <property name=*"softMinEvictableIdleTimeMillis"* value=*"1800000"*/>  <property name=*"numTestsPerEvictionRun"* value=*"2"*/>  </pool> |

exe.properties配置示例：

|  |
| --- |
| am\_app\_bal\_580.relat=EXE-CEN3-BD6  am\_app\_bal\_580.datasource=base,res,sec,so3,ams3,party  am\_app\_bal\_578.relat=EXE-CEN3-BD6  am\_app\_bal\_578.datasource=base,res,sec,so3,ams3,party |

涉及的表：

|  |
| --- |
| */\*appframe 数据源表\*/*  select \* from cfg\_db\_acct t;  select \* from cfg\_db\_jdbc\_parameter t;  select \* from cfg\_db\_relat t;  select \* from cfg\_db\_url t; |

数据源配置原则：

1. 预取模式

a）定义：在defaults.xml中配置了prefetch=”true”，需要同时配置一个预取数据源，其他数据源信息通过预取数据源获取；生产环境通常采用此种方式。

b）可以通过exe.properties控制进程可访问的数据源；该文件可以存在，也可以不存在，通过下面即将介绍的高级模式，也可以实现控制进程可访问的数据源；该文件存在的目的是方便配置，比如有多个进程可以访问的数据源是一样的，此时相当于将这些进程的ServerName分成了一个组。

c）预取模式模式下面获取数据源的方式又分为两种模式：高级模式和非高级模式。

高级模式：

定义：在defaults.xml中配置了isAdvanceUrl=”true”。

方法：通过cfg\_db\_relat表的DB\_ACCT\_CODE字段和SERVER\_NAME字段确定能够访问的数据库的URL。

非高级模式：

定义：不是高级模式的<pool>都是非高级模式。

方法：使用cfg\_db\_acct表中配置的host,port,sid组合成数据库URL；同时遵守数据源属性配置优先级规则。

1. 普通/开发模式

定义：即非预取模式。

限制（一）：由于没有预取数据源，所以primary=”true”的数据源的url是一定要配置的；因为，需要通过该数据源访问数据库配置表。

限制（二）：不支持高级模式。

1. 数据源属性配置优先级规则

启动参数中指定的属性 > cfg\_db\_jdbc\_parameter表中配置的属性 > 配置文件中配置的属性 > cfg\_db\_acct表中配置的属性。

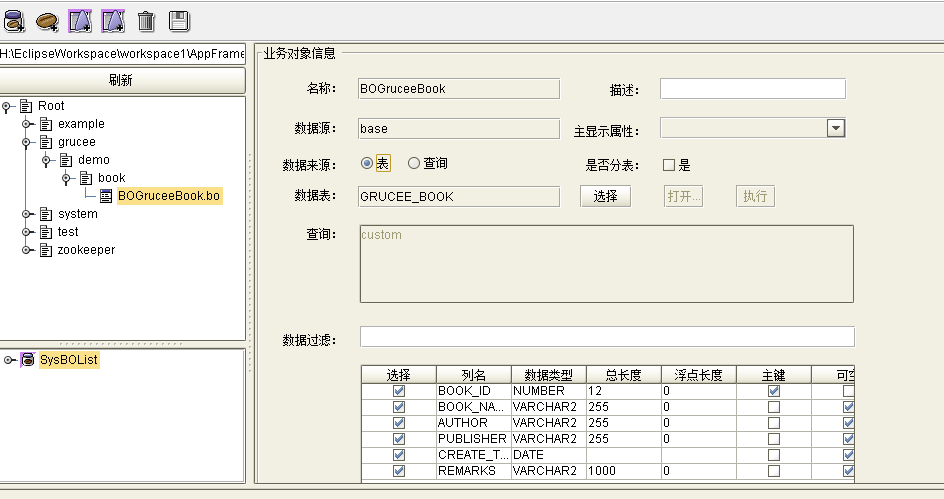
## 4.2 Configtools的使用

使用：参考《Appframe配置工具configtool使用手册.doc》

主要介绍如何生成两种bo

4.2.1 生成bo

1. table bo

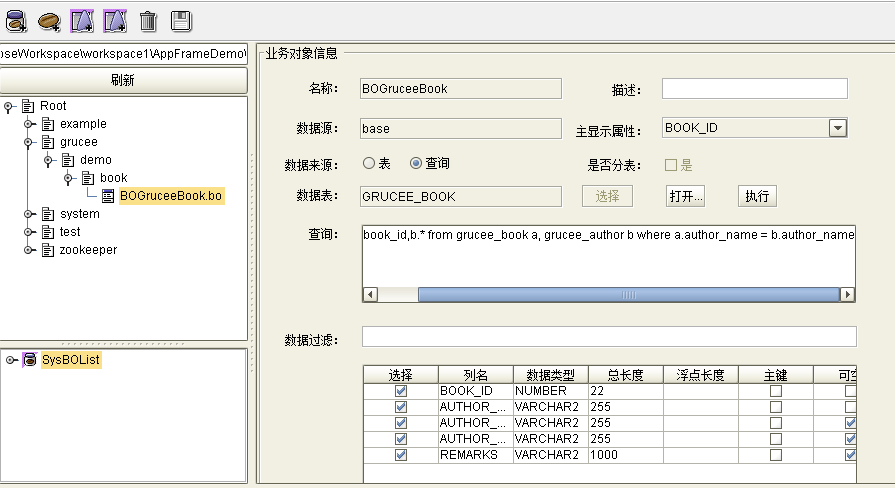


上述是单表的bo。

1. query bo

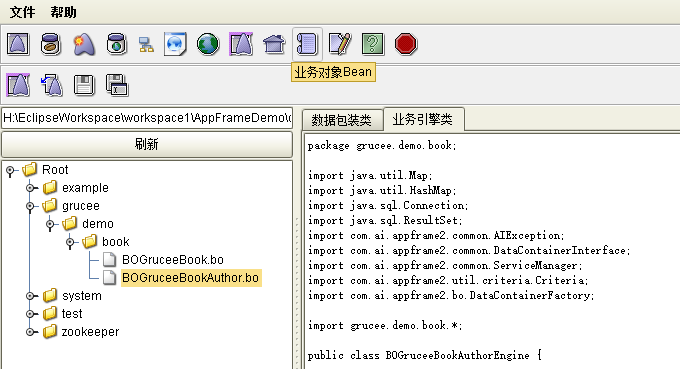
限制有个场景，根据书名，查询作者信息，并缓存起来（热点数据）。

query bo:可以关联表，生成bo



4.2.2 生成业务对象

选择一个刚才生成的bo

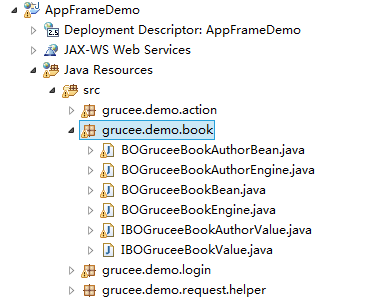


双击后，右侧能看到生成业务bean以及engine；点击保存。

点击左上角的接口生成，在弹出的对话框中，直接点击保存即可。



最后生成的类：



## 4.3 数据容器

### 4.3.1 BOGruceeBookEngine

数据库查询的引擎（代理）类，后面所有的数据库操作都可以通过该类实现。

### 4.3.2 DataContainerInterface

BOGruceeBookBean实现了DataContainerInterface，是数据库查询结果的一个容器；并且对bo文件里面的每个字段，都提供了相应的getter/setter方法，方便使用。

DataContainer容器中有三个Map比较重要：

|  |
| --- |
| **private** HashMap m\_back = **null**;  **private** HashMap m\_front = **null**;  //模糊化后的值  **private** HashMap mask = **null**;//new HashMap(); |

m\_front:修改后的值（调用bean的set方法）是保存到m\_front中的。

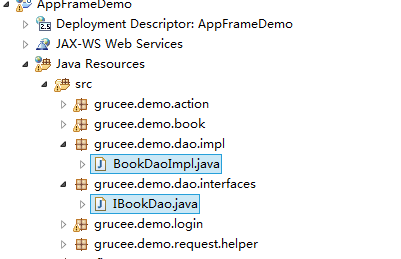
获取某个属性的值get(String name)的顺序:m\_front > m\_mask > m\_back。

### 4.3.3 ObjectType

ObjectType对象是对bo文件的抽象，即解析bo文件后生成ObjectType对象，保存每个bobean的属性列表、属性对应的java类型、属性对应的数据库类型。

每个Bean都维护了一个ObjectType对象。

## 4.4 开发Dao层-保存/修改



我们来看一个保存方法（同时还有其他的批量接口）：

|  |
| --- |
| @Override  **public** **void** saveBook(String name, String author, String publisher, String remarks) **throws** Exception {  BOGruceeBookBean bean = **new** BOGruceeBookBean();  // 主键生成  bean.setBookId(BOGruceeBookEngine.*getNewId*().longValue());  bean.setBookName(name);  bean.setAuthorName(author);  // 获取数据库时间  bean.setCreateTime(BOGruceeBookEngine.*getSysDate*());    BOGruceeBookEngine.*save*(bean);  } |

这里面有两个需要注意的地方:

1. 序列号生成
2. 需要在数据库中为表创建一个序列号
3. 在cfg\_id\_generator中加入一条数据，该表中有个字段STEP\_BY，用来对sequence缓存，避免频繁访问数据库;
4. 数据库时间获取

维护一个本地时间和数据库时间的差值，该差值定时更新；差值大于10分钟的情况，每次都去数据库查询当前时间。

## 4.5 DAO层调用

1、调用的java代码

|  |
| --- |
| //add to database  IBookDAO bookDao = (IBookDAO) ServiceFactory.*getService*(IBookDAO.**class**);  bookDao.saveBook(name, author, publisher, remarks); |

这个时候，如果你从界面上增加一本书的话，调用到上面的代码的时候，会报下面的错误：java.lang.Exception: 没有配置DAO实现:grucee.demo.book.dao.impl.BookDAOImpl对应的数据源范围

2、在defaults.xml中，为DAO服务配置数据源：

|  |
| --- |
| <!--默认数据源信息-->  <datasource>    <!--映射每个DAO模块对应的操作数据源-->  <mapping>  <!-- appframe demo -->  <property name=*"grucee.demo.book"* value=*"base"*/>  </mapping>  </datasource> |

加上上面的数据源配置，我们再重新调用，会报下面的错误

|  |
| --- |
| [2015-05-02 19:33:35,562] [http-8080-2] (admin) (DataStoreImpl.java:735) ERROR com.ai.appframe2.bo.DataStoreImpl - 只读连接不能进行update/delete/insert操作  (admin)java.sql.SQLException: 只读连接不能进行update/delete/insert操作 |

上述异常，表示我们获取的是一个只读的数据库连接，不能进行数据库更新操作。

Appframe对于数据库变更操作，要求必须包裹在事务中。

现在，请阅读第5章服务开发内容，然后再回来继续阅读。

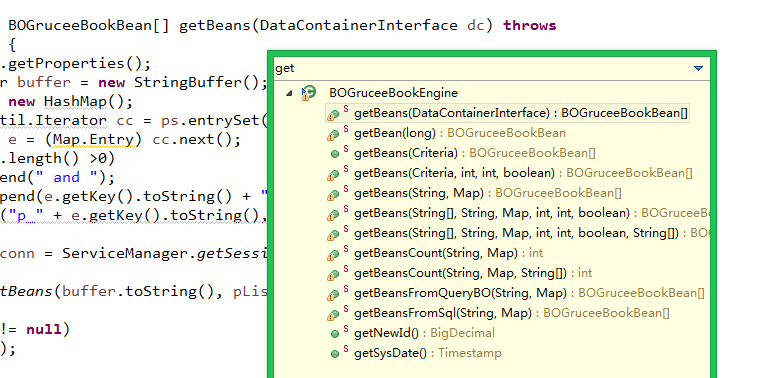
开发完服务后，我们的调用修改为通过服务调用的方式：

|  |
| --- |
| //add to database  IBookSV bookSV = (IBookSV) ServiceFactory.*getService*(IBookSV.**class**);  bookSV.saveBook(name, author, publisher, remarks); |

## 4.6 开发Dao层-查询

|  |
| --- |
| @Override  **public** BOGruceeBookBean[] queryBookByBookName(String bookName) **throws** Exception {  StringBuilder sb = **new** StringBuilder();  sb.append(BOGruceeBookBean.***S\_BookName***).append(" like :name");    Map parameters = **new** HashMap();  parameters.put("name", bookName);  **return** BOGruceeBookEngine.*getBeans*(sb.toString(), parameters);  } |

Engine类提供了不同查询接口方便传入查询条件：



## 4.7 无事务访问数据库

|  |
| --- |
| conn = ServiceManager.*getSession*().getNewConnection(dataSource); |

这种方式获取的数据库连接，没有事务也可以执行数据库变更操作；通常用在不需要回滚数据库，自己处理异常的地方。

此时需要注意执行完要调用close方法，将连接归还连接池。

# 第5章 服务开发

## 5.1 服务开发

|  |
| --- |
| <service id=*"com.ai.appframe.CommonTreeService"* invoke=*"pojo"* type=*"primitive"*>  <property name=*"interfaceClass"* value=*"com.ai.appframe2.util.tree.ICommonTreeService"*/>  <property name=*"implClass"* value=*"com.ai.appframe2.util.tree.impl.DefaultTreeService"*/>  </service> |

只需要开发一个接口以及相应实现，然后在defaults.xml中像上面一样配置就可以。

appframe还提供了默认的查找服务的方法，不过需要遵守下面的约定：

如果没有在services.xml中配置，则进行下面的转换

包转换：com.asiainfo.deploy.api.interfaces –》 com.asiainfo.deploy.api.impl

类名转换：IAppOperationSVProvider –》 AppOperationSVProviderImpl

必须包含SV，且大写。

## 5.2 服务调用方法

和DAO层调用一样（是在一个方法里面实现的）

|  |
| --- |
| ((IAppOperationSVProvider)ServiceFactory.*getService*(IAppOperationSVProvider.**class**)) |

## 5.3 FAQ

1、如果不想用Dao层，直接在服务层调用Engine类是否可以？

可以，使用Engine访问数据库只要保证当前上下文中已经设置数据源，并且已经启动事务即可；此时需要保证在服务层设置了数据源。

2、appframe支持多数据源，但是不支持分布式事务（指的是LocalMutilTransactionImpl实现）。

3、可以为服务指定数据源

|  |
| --- |
| <!--默认事务信息.默认采用Required的方式进行事务操作，采用的实现类为配置的实现类-->  <transaction type=*"Required"*>  <clazz name=*"com.ai.appframe2.complex.transaction.impl.LocalMutilTransactionImpl"*/>  <!--映射每个服务模块对应的事务数据源，可以具体到类名-->  <mapping>  <!--平台自带-->  <property name=*"com.ai.appframe2.complex.self.service.base"* value=*"base"*/>  <property name=*"com.ai.appframe2.complex.self.service.check"* value=*"base"*/> |

# 第6章 分表

只需要简单的配置，就可以使用appframe的分表功能

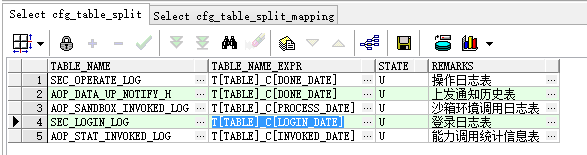
## 6.1 分表配置

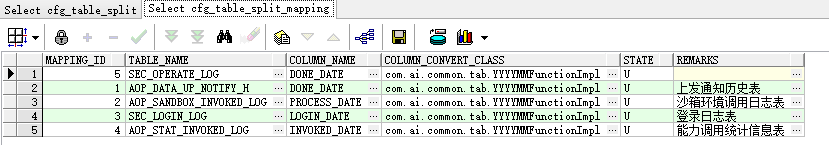
两张相关表：

*/\*分表\*/*

select \* from cfg\_table\_split;

select \* from cfg\_table\_split\_mapping;





## 6.2 分表转换

除了根据REGION\_ID和REGION\_CODE进行分表的情况，其他情况必须在cfg\_table\_split\_mapping配置分表字段的转换类（即分表字段的值映射到那张表的转换方法）；对于REGION\_ID和REGION\_CODE分表的情况，则当前线程上下文必须已经设置了中心信息，即CenterFactory.getCenterInfo().getRegion()不为null。

转换类实例(按年分表)：

|  |
| --- |
| **public** **class** YYYYFunctionImpl  **implements** IFunction  {  **public** YYYYFunctionImpl()  {  }  **public** String convert(Object value)  **throws** Exception  {  String rtn = **null**;  **if**(value **instanceof** Date)  {  java.text.DateFormat objDateFormat = **new** SimpleDateFormat("yyyy");  rtn = objDateFormat.format(value);  } **else**  {  ExceptionUtil.*throwBusinessException*("BAS0000011");  }  **return** rtn;  }  } |

## 6.3 分表查询

分表查询的时候，分表字段一定是要有值的，不能为空；否则不能确定哪张表。

比如说，如果某张表按月分表，现在我们想查询分表字段（时间字段）某一段范围的记录，

看一个示例：

|  |
| --- |
| java.util.Date[] tables = LoggerDateUtils.dividedByMonth(begin, end);  **if** (tables.length == 1) // 起止时间在一个月内，也就是在一个表内  {  sb.append(" and ").append(BOABGMonBusiErrorLogBean.***S\_CreateDate***).append(" between :begin and :end");  sb.append(*orderby*());  parameters.put("begin", **new** Timestamp(begin.getTime()));  parameters.put("end", **new** Timestamp(end.getTime()));  // 查询当前时间所在的表  parameters.put(BOABGMonBusiErrorLogBean.***S\_CreateDate***, *fomatter*.get().format(begin));  DataContainerInterface[] r2 = *queryOnce*(conn, flag, sb.toString(), parameters, beantype);  LoggerDBUtils.closeDBConnection(conn); |

# 第7章 中心、路由

## 7.1 中心配置

|  |
| --- |
| <!--中心的类型和实现类的对应关系-->  <center>  <!--  <property name="BillId:KeyNum" value="example.center.BillIdCenterImpl"/>  -->  </center> |

## 7.2 中心类型实现类

上述实现类，主要实现根据业务数据确定区域以及中心的功能。以浙江为例，通常业务是分中心部署的，然后按照区域分表，不会跨中心（数据库）操作。只有0中心的业务是会跨中心的；



对于跨中心的业务，会有一个业务字段来确定它是归属哪个中心、哪个区域的（规则是事先就定好的）；上述的实现类就是该规则的实现。

|  |
| --- |
| /\*\*  \* 帐户ID定位路由信息的中心实现类  \*/  **public** **class** AcctIdCenterImpl **implements** ICenter {  **public** AcctIdCenterImpl() {  }  **public** CenterInfo getCenterByValue(String value) **throws** Exception {  String regionId = CenterUtil.*getRegionIdByAcctId*(Long.*parseLong*(value));  **if** (StringUtils.*isBlank*(regionId)) {  ExceptionUtil.*throwBusinessException*("BAA0000006", value);  }  **return** **new** CenterInfo(DistrictUtil.*getCenterByRegionId*(regionId), regionId);  }  } |

## 7.3 中心路由规则

对于跨中心的数据源，需要在defaults.xml中配置template字段，如下示例：

|  |
| --- |
| <pool name=*"so"* primary=*"true"* template=*"so{CENTER}"*> |

在执行需要中心操作的业务时，使用下面的方式获取不同数据库连接：替换掉*{CENTER}*

|  |
| --- |
| String newds = **null**;  String template = DataSourceTemplate.*getCurrentTemplate*();  **if** (StringUtils.*contains*(template, DataSourceTemplate.***CENTER\_FLAG***)) {  newds = StringUtils.*replace*(template, DataSourceTemplate.***CENTER\_FLAG***, *getCenterByRegionId*(destRegionId));  }  conn = ServiceManager.*getSession*().getConnection(newds); |

## 7.4 总结

业务字段的值，经过中心实现类的规则处理，能够得出该条业务数据归属于哪个中心，哪个区域；中心确定数据库（不同中心对应不同数据库），区域确定库中的哪张表（如果按照区域分表的话）。

## 7.5 前台请求设置中心

可以在前台发送的请求中传入中心相关的两个参数（如下面的代码示例），Appframe会自动设置中心信息。

|  |
| --- |
| //中心的Center的处理  String centerType = request.getParameter(CenterFactory.***CENTER\_TYPE***);  String centerValue = request.getParameter(CenterFactory.***CENTER\_VALUE***);  **if** (!StringUtils.*isBlank*(centerType) && !StringUtils.*isBlank*(centerValue)) {  CenterFactory.*setCenterInfoByTypeAndValue*(centerType, centerValue);  }  **else** {  CenterFactory.*setCenterInfoEmpty*();  } |

# 第8章 缓存

## 8.1 缓存开发

|  |
| --- |
| /\*\*  \* （bookId，作者详细信息）的缓存  \*  \* **@author** 孙德东(24204)  \*/  **public** **class** BookAuthorCacheImpl **extends** AbstractCache {  @Override  **public** HashMap getData() **throws** Exception {  // 这里没有放在服务中，因为使用只读连接是可以做查询的；  IBookDAO bookDao = (IBookDAO)ServiceFactory.*getService*(IBookDAO.**class**);  BOGruceeBookAuthorBean[] beans = bookDao.getAllBookWithAuthorInfo();    HashMap map = **new** HashMap();  **if** (beans == **null**) {  **return** map;  }    **for** (BOGruceeBookAuthorBean bean : beans) {  map.put(bean.getBookId(), bean);  }  **return** map;  }  } |

## 8.2 缓存配置

|  |
| --- |
| <!-- appframe demo -->  <cache id=*"grucee.demo.cache.BookAuthorCacheImpl"*>  <property name=*"cronExpression"* value=*"0/30 \* \* \* \* ?"* />  </cache> |

其中，定时刷新的表达式配置，请参考QuartZ的规则。

## 8.3 缓存调用

|  |
| --- |
| @Override  **public** BOGruceeBookAuthorBean getAuthorInfoByBookid(**long** bookId) **throws** Exception {  //HashMap map = CacheFactory.getAll(BookAuthorCacheImpl.class);  **return** (BOGruceeBookAuthorBean) CacheFactory.*get*(BookAuthorCacheImpl.**class**, bookId);  } |

# 第9章 AppFrameDemo下载

