

Albianj使用方法

大嘴

2016-02-01

- albianj结构
- albianj配置
- albianj启动
- albianj配置文件
- albianj实战

albianj的源码和jar包

- ▶ Albianj.Cached
- ▶ Albianj.Cached.Impl
- ▶ Albianj.Commons
- ▶ Albianj.Configuration
- ▶ Albianj.Configuration.Impl
- ▶ Albianj.DFS.YDB.Client
- ▶ Albianj.DFS.YDB.Client.Impl
- ▶ Albianj.DFS.YDB.Client.Test
- ▶ Albianj.Dispatcher
- ▶ Albianj.JobManager
- ▶ Albianj.JobManager.Impl
- ▶ Albianj.Kernel
- ▶ Albianj.Kernel.Impl
- ▶ Albianj.Loader
- ▶ Albianj.Main
- ▶ Albianj.Packing
- ▶ Albianj.Persistence
- ▶ Albianj.Persistence.Impl
- ▶ Albianj.Qidian.Test
- ▶ Albianj.Remote.Test
- ▶ Albianj.Restful
- ▶ Albianj.Restful.Impl
- ▶ Albianj.Restful.Jetty
- ▶ Albianj.Restful.Service
- ▶ Albianj.Security
- ▶ Albianj.Security.Impl
- ▶ Albianj.UNID
- ▶ Albianj.UNID.Client.Test
- ▶ Albianj.UNID.Impl



- Albianj.Cached.jar
- Albianj.Commons.jar
- Albianj.Configuration.Impl.jar
- Albianj.Configuration.jar
- Albianj.DFS.YDB.Client.Impl.jar
- Albianj.DFS.YDB.Client.jar
- Albianj.Kernel.jar
- Albianj.Loader.jar
- Albianj.Persistence.jar
- Albianj.Restful.Impl.jar
- Albianj.Restful.jar
- Albianj.Security.jar
- Albianj.spx
- Albianj.UNID.jar

albianj的jar包说明

- **Albianj.Cached.jar** 缓存的封装接口
- **Albianj.Commons.jar** 公用类
- Albian.Configuration.jar&Albianj.configuration.Impl.jar 配置的接口和实现
- Albianj.DFS.YDB.Client.jar&Albianj.DFS.YDB.Client.Impl.jar dfs的客户端接口和实现
- **Albianj.Kernel.jar** 内核的接口定义
- **Albianj.Loader.jar** 自定义classload和插件注册接口
- **Alban.Persistence.jar** 存储层的接口定义
- Albianj.Restful.jar&Albianj.Restful.Impl.jar 轻量级restful的接口定义和实现
- **Albianj.Security.jar** 安全、加密算法的接口定义
- **Albianj.UNID.jar** 唯一id客户端接口定义
- **Albianj.spx** 所有接口实现包

红色为albianj运行必须要加载的jar

spx文件存在的原因

- 接口简单： 只有接口定义和service
- 代码统一： 所有访问都是从service出
- 隐藏了所有的实现， 程序员不会陷入jar问题
- 线上维护简单， 不会出现版本， 打包方式不一样造成的不一致

Albianj提供了什么？

albian的主要功能

- 一个简单的Service服务，比spring轻量很多很多
- 一个简单的ORM框架
- 数据路由服务
- 分布式事务服务（支持强、弱两种模型）
- 简单的缓存服务
- 统一的Id服务
- 日志服务（这部分有待扩展）
- 密码安全服务
- 简单的restful服务
- 一个轻量级的配置服务

Albianj如何使用？

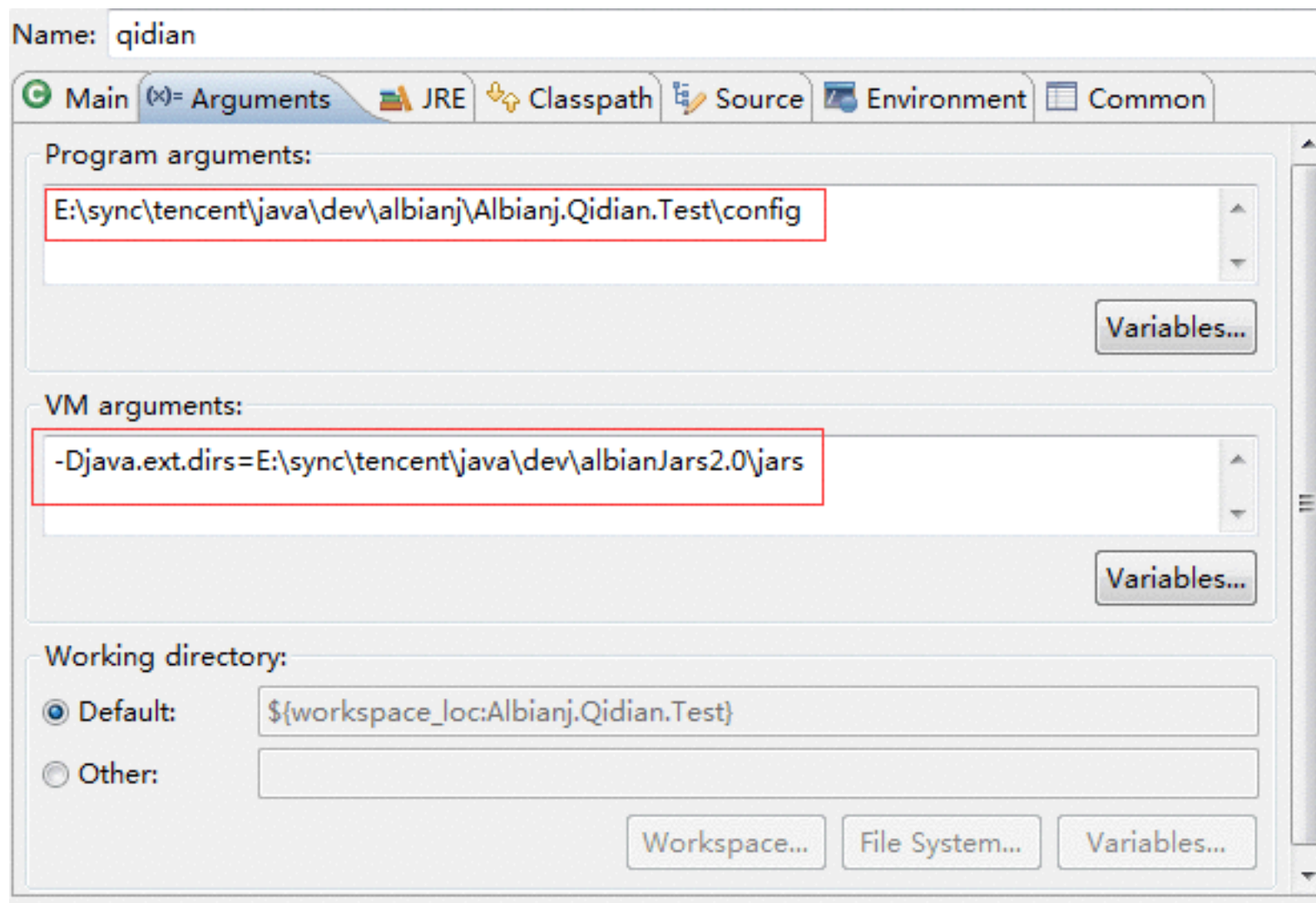
新建典型项目

albianj典型的项目布局



注意：因为只是一个example，所以接口和实现没有完全分离

第一步 环境配置



- 1: 配置config路径
- 2: 配置jar路径

第二步 启动albianj

启动albianj

- 进程启动的时候执行AlbianBootService.start () ；
- start方法有3个重载，所有默认的参数都是当前文件夹。那么3个参数代表的意思分别是：
 - classpath: Albianj.spx文件所在目录
 - kernelpath: kernel.properties文件所在的目录
 - configpath: 所有的xml配置文件所在的目录

albianj启动注意事项

- start方法在一个进程中只被执行一次
- 请自行处理start方法抛出的异常
- 90%的启动不成功都是因为配置问题
- 关于路径，没有系统区分，win和linux都可以支持
- 关于Albianj.spx文件，albianj搜寻该文件的次序依次为：classpath参数路径；当前路径；ext路径
- 成功消息：startup albianj engine is success!

第二步 写代码

规则

- albianj支持接口-实现类模式，其实就是IOC模式
- service需要在service.xml配置文件中配置
- 所有的service接口必须继承自IAlbianService，所有service类必须继承自FreeAlbianService
- 所有带Albian开头都是系统级service，业务service请不要使用Albian开头
- 所有的service，都通过AlbianServiceRouter的getService得到，所有的service在进程中都是单例的；
- 约定：每个service在接口中声明一个Name的字段，值为该service在service.xml中配置的名称

自定义接口

```
public interface IUserService extends IAlbianService {  
    final static String Name="UserService";  
  
    public boolean create(String sessionId,IUser user);  
    public boolean create(String sessionId,List<IUser> users);  
    public boolean modifyName(String sessionId,BigInteger id,String name);  
    public IUser load(String sessionId,BigInteger id);  
    public boolean remove(String sessionId,BigInteger id);  
}
```

实现接口的自定义类

```
public class UserService extends FreeAlbianService implements IUserService {  
    private static IAlbianPersistenceService getPersistenceService(){  
        IAlbianPersistenceService aps = AlbianServiceRouter.getService(  
            IAlbianPersistenceService.class, IAlbianPersistenceService.Name, true);  
        return aps;  
    }  
    @Override  
    public boolean create(String sessionId, IUser user) {  
        // TODO Auto-generated method stub  
        try {  
            getPersistenceService().create(sessionId, user);  
        } catch (AlbianDataServiceException e) {  
            return false;  
        }  
        return true;  
    }  
}
```

第三步 配置该service

service配置



第四步 使用service

service使用方法

```
        IUserService us =  
        AlbionServiceRouter.getService(IUserService.cl  
            ass, IUserService.Name, true);
```

内置Service—logger

获取logger

```
AlbianServiceRouter.getLogger()
```

logger详解

- logger的不同级别根据IAlbianLoggerService接口的方法名决定
- albianj的logger从log4j而来，配置文件为log4j.xml
- albianj内置了RunningLogger和SqlLogger
- albianj的logger会根据prefix和日志生成时间区分日志文件，所以两次启动日志文件将会是两个

logger配置扩展

```
<appender name="AlbianRunningAppender"
  class="org.albianj.logger.impl.AlbianRollingFileAppender">
  <param name="File" value="logs" /><!-- 设置日志输出文件名 -->
  <!-- 设置是否在重新启动服务时，在原有日志的基础添加新日志 -->
  <param name="Append" value="false" />
  <param name="MaxBackupIndex" value="-1" />
  <param name="MaxFileSize" value="5mb" />
  <!-- 请配置你的业务名称 -->
  <param name="prefix" value="prefix" />
  <param name="Format" value="HHmmss" />
  <param name="Suffix" value="running.log" />
  <param name="encoding" value="UTF-8" />
  <!-- 请配置你的log存放的文件夹路径，和上面的File保持一致 -->
  <param name="path" value="logs" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="[%d{yyyy-MM-dd HH:mm:ss}] %p %m%n" />
  </layout>
</appender>
```

自定义logger实现类

logger存放路径，两者必须保持一致

Service结束

数据访问层

声明实体接口

```
public interface IUser extends IAlbianObject {  
  
    public BigInteger getId();  
    public void setId(BigInteger id);  
  
    public String getName();  
    public void setName(String name);  
  
    public boolean getIsDelete();  
    public void setIsDelete(boolean isDelete);  
}
```

声明实体类

```
public class User extends FreeAlbianObject implements IUser {  
    BigInteger id = null;  
    String name = null;  
    boolean isDelete = false;  
    @Override  
    public BigInteger getId() {  
        // TODO Auto-generated method stub  
        return this.id;  
    }  
  
    @Override  
    public void setId(BigInteger id) {  
        // TODO Auto-generated method stub  
        this.id = id;  
    }  
}
```

配置

```
<AlbianObjects>
  <AlbianObject Interface="org.albianj.qidian.test.object.IUser"
    Type="org.albianj.qidian.test.object.impl.User">
    <Cache Enable="false" LifeTime="300" Name=""></Cache>
    <!-- 删除了缓存服务，所以这部分配置一直为false，等待开启了缓存服务，再更加实际情况配置 -->
    <Members> <!-- 所有对象的属性映射，如果使用默认，可以不需要配置 -->
      <Member Name="Id" FieldName="id" AllowNull="false"
        DbType="Bigint" IsSave="true" PrimaryKey="true" />
    </Members>
  </AlbianObject>
</AlbianObjects>
```

实体接口和类信息

是否集成缓存

实体的属性信息

注意：Albianj没有“约定大于配置”的功能，所以所有的实体都需要在persistence.xml显示的配置

cache配置注意事项

- enable: 表示是否对这个object打开默认的缓存。
如果enable=true, 表示开启缓存, 缓存需要Albian.Cached的支持。必须在service.xml中启用AlbianCachedService服务
- LifeTime: 表示缓存的时间, 单位是秒
- Name: 这个值是cached.xml中配置的名称值, 表示这个对象的缓存存储到这个cached中。

members配置注意事项

- Name: java程序中的属性名。
- FieldName: 数据库中对应的字段名
- AllowNull: 表示是否可以为空
- DbType: 表示在数据库的类型
- IsSave: 表示是否需要被保存到数据库
- PrimaryKey: 表示是否是数据库的主键字段
- 主键必须配置, 而且每个表必须还有主键
- 如果属性和数据库字段一致除了主键外别的属性不用一一配置

配置数据源

配置

-----o-----

```
<Storage>
  <Name>User</Name>
  <DatabaseStyle>MySQL</DatabaseStyle>
  <Server>127.0.0.1</Server>
  <Database>user</Database>
  <User>root</User>
  <Password>xuhf</Password>
  <Pooling>true</Pooling>
  <MinPoolSize>10</MinPoolSize>
  <MaxPoolSize>20</MaxPoolSize>
  <Timeout>60</Timeout>
  <Charset>utf8</Charset>
  <Transactional>true</Transactional>
</Storage>
```

数据源注意事项

- 数据源配置在storage.xml文件中
- storage节点只是一类数据的模板，也意味着一个storage配置信息可以被多个在同一台服务器上、拥有相同用户名和密码的、名字相关的数据库共享；
- storage的databasestyle目前支持mysql、oracle、mssql
- storage支持数据库信息加密，对于user和password节点，当kernel.properties中的level=Release的时候，这两个配置节将会被认为是加密的

数据路由

配置

```
public class UserDataRouter extends FreeAlbianObjectDataRouter {  
  
    static String WR = "UserDataRouterW";  
    static String RR = "UserDataRouterR";  
  
    @Override  
    public List<IDataRouterAttribute> mappingWriterRouting(Map<String, IDataRouterAttribute> routings,  
        IAlbianObject obj) {  
        // TODO Auto-generated method stub  
        IDataRouterAttribute dra = routings.get(WR);  
        List<IDataRouterAttribute> drs = new LinkedList<>();  
        drs.add(dra);  
        return drs;  
    }  
}
```

一般情况下，数据路由继承FreeAlbianObjectDataRouter

路由接口

```
● A mappingWriterRouting(Map<String, IDataRouterAttribute>, IAlbianObject) : List<IDataRouterAttribute>
● A mappingWriterRoutingStorage(IDataRouterAttribute, IAlbianObject) : String
● A mappingWriterRoutingDatabase(IStorageAttribute, IAlbianObject) : String
● A mappingWriterTable(IDataRouterAttribute, IAlbianObject) : String
● A mappingReaderRouting(Map<String, IDataRouterAttribute>, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : IDataRouterAttribute
● A mappingReaderRoutingStorage(IDataRouterAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
● A mappingReaderRoutingDatabase(IStorageAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
● A mappingReaderTable(IDataRouterAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
● A mappingExactReaderRouting(Map<String, IDataRouterAttribute>, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : IDataRouterAttribute
● A mappingExactReaderRoutingStorage(IDataRouterAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
● A mappingExactReaderRoutingDatabase(IStorageAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
● A mappingExactReaderTable(IDataRouterAttribute, Map<String, IFilterCondition>, Map<String, IOrderByCondition>) : String
```

- XXXRouting—选择要使用的一个/多个路由
- XXXStorage—选择路由下的storage
- XXXDatabase—选择storage的数据库
- XXXTable—选择表

配置

```
<AlbianObject Interface="org.albianj.qidian.test.object.IUser"
Type="org.albianj.qidian.test.object.impl.User"
  Router= "org.albianj.qidian.test.service.impl.UserDataRouter">
    <WriterRouters Enable="true">
      <WriterRouter Name="UserDataRouterW" StorageName="User"
        TableName="user" Enable="true"></WriterRouter>
    </WriterRouters>
    <ReaderRouters Enable="true">
      <ReaderRouter Name="UserDataRouterR" StorageName="User"
        TableName="user" Enable="true"></ReaderRouter>
    </ReaderRouters>
  </AlbianObject>
```

数据路由路径

写路由

读路由

路由注意事项

- 读/写路由都可以同时配置多个、更加自己的使用，使用mappingXXXRouting方法选择一个或者多个路由
- 读/写路由的name只要保证在该父节点内唯一
- StorageName是storage.xml中配置name
- TableName：主要配置表的基本名字就可以
- 如果读写都只有一个路由，可Router配置为：
org.albianj.persistence.impl.object.AlbianObjectDataRouter
- 一份数据保存多份：配置多个WriterRouting，并且mappingWriterRouting返回多个需要保存数据的路由；

使用办法

- 把service.xml中的数据访问层service打开
- 获取IAlbianPersistenceService的实例，进行数据库的操作
- IAlbianPersistenceService中每个方法都带的sessionid是指执行这个方法的用户id，如果开发者给null，albianj将会自动生成一个，这个id的作用是为了排错方便，能联系上下文
- albianj会自动判断是否需要开启二阶段提交的分布式事务，如果需要将会自动打开，对于开发者是透明的
- 为了更好的维护数据的完整性，请不要做物理删除操作，尽量使用逻辑删除

获取IAlbianPersistenceService实例

```
private static IAlbianPersistenceService getPersistenceService(){  
    IAlbianPersistenceService aps = AlbianServiceRouter.getService(  
        IAlbianPersistenceService.class, IAlbianPersistenceService.Name, true);  
    return aps;  
}
```

新建数据

```
public boolean create(String sessionId, List<IAlbianObject> users){  
    try {  
        getPersistenceService().create(sessionId, users);  
    } catch (AlbianDataServiceException e) {  
        return false;  
    }  
    return true;  
}
```

保存数据

```
public boolean modifyName(String sessionId, BigInteger id, String name) {  
    // TODO Auto-generated method stub  
    try {  
        LinkedList<IFilterCondition> wheres = new LinkedList<>();  
        wheres.add(new FilterCondition("Id", id));  
        IUser u = getPersistenceService().loadObject(sessionId, IUser.class, LoadType.exact, wheres);  
        if(null != u){  
            u.setName(name);  
        } else {  
            u = new User();  
            u.setId(id);  
            u.setName(name);  
        }  
        getPersistenceService().save(sessionId, u);  
    } catch (AlbianDataServiceException e) {  
        return false;  
    }  
    return true;  
}
```

一般查询

```
@Override
public IUser load(String sessionId, BigInteger id) {
    // TODO Auto-generated method stub
    try {
        LinkedList<IFilterCondition> wheres = new LinkedList<>();
        wheres.add(new FilterCondition("Id", id));
        IUser u = getPersistenceService().loadObject(sessionId, IUser.class, LoadType.dirty, wheres);

        return u;
    } catch (AlbianDataServiceException e) {
        return null;
    }
}
```

精确查询

```
public boolean remove(String sessionId, BigInteger id) {  
    // TODO Auto-generated method stub  
    try {  
        LinkedList<IFilterCondition> wheres = new LinkedList<>();  
        wheres.add(new FilterCondition("Id", id));  
        IUser u = getPersistenceService().loadObject(sessionId, IUser.class, LoadType.exact, wheres);  
        if(null != u){  
            u.setIsDelete(true);  
            return getPersistenceService().save(sessionId, u);  
        } else {  
            return true;  
        }  
    } catch (AlbianDataServiceException e) {  
        return false;  
    }  
}
```


AlbianPersistenceService使用注意

- 一般情况下，保存数据（不管是新建或者是更改）都使用save接口就可以
- 如果需要更改数据，请务必、必须、一定先从数据仓库中load一下数据，然后再更新
- 所有的save、create、modify、remove都支持同时对不属于统一类型的数据进行事务性操作

AlbianPersistenceService使用注意

- 查询数据分为3种情况：quickly：先查缓存，没有走readerRouting；dirty：直接从ReaderRouting查询数据；exact：精确查询，从WriterRouting查询数据，一般exact使用在读写分离需要更新数据的情况下
- 查询数据只支持单库单表获取，条件查询使用IFilterCondition和FilterCondition生成查询表达式，排序使用IOrderByCondition和OrderByCondition生成排序表达式
- 对于查询时需要对同一字段进行多条件查询的时候，例如 $a > 10$ and $a < 100$ ，请对于任一一一个a赋一个任意的、在当前条件中唯一的别名，方法：setAliasName
- albianj查询支持or/and条件，也支持基本的 $<$ $<=$ $=$ $>=$ $>$ 等详见LogicalOperation和RelationalOperator枚举

缓存层

Albian缓存

- albianj对于缓存做了统一的处理，也提供了统一的接口
- 启用albianj缓存，需要查看service.xml中是否已经开启了albianj的缓存服务
- albianj的缓存支持分布式和本地，分布式缓存支持redis
- albianj的缓存需要cached.xml配置文件

Albian缓存接口

```
public interface IAlbianCachedService extends IAlbianService{
    static String AlbianCachedServiceDefault = "AlbianCachedService";
    void init(Object initObject);

    void set(String cachedName,String k,Object v);
    void set(String cachedName,String k,Object v,int tto);
    void delete(String cachedName,String k);
    boolean exist(String cachedName,String k);
    <T> T get(String cachedName,String k,Class<T> cls);
    <T> List<T> getArray(String cachedName,String k,Class<T> cls);

    boolean freeAll(String nodeName);

    Object getCachedClient(String cachedName);
    void returnCachedClient(String cachedName, Object client);
}
```

Albany cached 配置文件

```
<CacheServers>  
  <CacheServer Name="AlbianConfigurtionCached" Enable="true"  
    Style="redis" ConnectTimeout="30" Type="" ConnectPoolSize="50">  
    <Server Host="10.97.19.43" Port="12345"></Server>  
    <Server Host="10.97.19.43" Port="1234"></Server>  
  </CacheServer>  
  <CacheServer Name="AlbianObjectCached" Enable="true"  
    Style="redis" Cluster="true" ConnectTimeout="30" Type="" ConnectPoolSize="50">  
    <Server Host="10.97.19.43" Port="12345"></Server>  
    <Server Host="10.97.19.43" Port="1234"></Server>  
  </CacheServer>  
  <CacheServer Name="local" Enable="false" Style="local"/>  
</CacheServers>
```

redis服务器列表

本地缓存

Albian缓存配置详解

- redis缓存支持hash和Cluster模式
- name为缓存的唯一名字，和数据层一起使用的时候需要用到，或者使用IAlbianCachedService接口操作缓存的时候，需要显式的指定
- style值为redis或者local
- type：缓存客户端类操作类，albianj会使用这个type来初始化缓存的客户端

唯一ID

Id服务

- albianj的Id分为客户端和服务端
- 服务端用c开发，可以支持256种类型的id生成
- 服务器最多可以支持16台机器同时提供服务，对于非java程序，服务器内置了web，可以使用类restful的方式直接访问获取
- 每秒每台机器可以给每种类型生成1w个id
- id服务的服务器可以水平的扩展
- id服务生成的id是一个uint64的值，有十进制和二进制两种算法可供使用
- id在单位时间内（1s内）不保证一定递增（分布式无法做到），但是保证在线性时间内（最短2s之间）的id肯定单调递增
- 正在开发的id生成器扩展组件可以保证id肯定递增和唯一

Id服务

- 启用albianj的id服务客户端，需要查看service.xml中是否已经开启了albianj的remote id服务
- albianj的id服务需要unid.xml配置文件
- 开启id客户端服务

```
<Service Id="AlbianRemoteIdService" Type="org.albianj.unid.service.impl.AlbianRemoteUNIDService" />
```

Id服务接口

```
static String Name = "AlbianRemoteIdService";  
public static final int TYPE_DEFAULT = 1;
```

```
public BigInteger createBookId();  
public BigInteger createAuthorId();  
public BigInteger createConfigItemId();  
public BigInteger createUNID();  
public BigInteger createUNID(int type);
```


创建ID

```
public void unpack(BigInteger bi, RefArg<Timestamp> time,  
                   RefArg<Integer> type);  
public void unpack(BigInteger bi, RefArg<Timestamp> time,  
                   RefArg<Integer> sed, RefArg<Integer> idx);
```

反解ID

unid.xml配置信息

```
<UNID>  
  <Servers>  
    <Server Host="10.97.19.43" Port="9048"  
      Timeout="30000" PoolSize="30"/>  
  </Servers>  
</UNID>
```



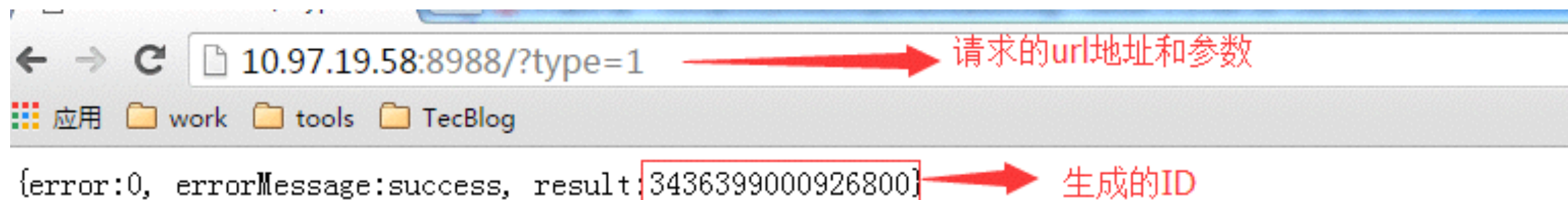
Id服务器信息

Id服务使用方法

客户端：

```
,  
  
IAlbianRemoteUNIDService arus = AlbianServiceRouter.getService(IAlbianRemoteUNIDService.class,  
    IAlbianRemoteUNIDService.Name);  
if (null == arus)  
    System.out.println("service is null.");  
  
for (int i = 0; i < 10000; i++) {  
    BigInteger bi = arus.createBookId();  
    System.out.println(bi.toString());  
}
```

http:



DFS的客户端使用

DFS服务

- albianj也集成了我们自主开发的dfs的客户端
- 服务端用c开发，可以支持任何类型的数据，目前，内容中心存储的是文本和mp3等音频文件
- 服务器没有限制，可以水平扩展，并且扩展中不会发生数据迁移导致的性能下降和不可用问题
- 鉴于目前的情况，我们对于upload和modify提供了2套api来处理，对于find和delete是一套，后面，我们会2套upload和modify合并为一套，这个工作可能会在4-5月份完成，届时会通知大家更新api

dfs服务

- 启用albianj的dfs服务客户端，需要查看service.xml中是否已经开启了albianj的AlbianYdbService服务
- albianj的dfs服务需要ydb.xml配置文件
- 开启dfs客户端服务

```
<Service Id="AlbianYdbService" Type="org.albianj.dfs.ydb.client.impl.service.AlbianYDBService" />
```


Id服务接口

```
public String upload(String groupname,byte[] data,long length,String suffix);
```

```
public byte[] find(String fid);
```

```
public boolean delete(String fid) ;
```

```
public String modify(String ofid,byte[] data,long length,String suffix);
```

```
public IAlbianYDBFileID parserFileId(String fid);
```

```
public String makeFileName(String fid);
```

```
public String upload(String groupname,byte[] data,long length,String suffix,BigInteger id);
```

```
public String modify(String ofid,byte[] data,long length,String suffix,BigInteger id);
```

内容中心使用API

视频，音频使用

目前预计，在4-5月份的时候，
两个upload和modify的api会统一掉

ydb.xml配置信息

```
<Ydb>  
  <Group Name="g001" Encoding="utf-8">  
    <Tracker Host="10.96.211.67" Port="4150" ConnectTimeout="30"></Tracker>  
  </Group>  
</Ydb>
```

存储的组名

tracker的配置信息

这些配置信息需要问运维要

upload使用方法

```
public static boolean testUpload(String groupname, String filename, String suffix, BigInteger id) {  
    IAlbianYDBService ays = AlbanServiceRouter.getService(IAlbianYDBService.class,  
        IAlbianYDBService.AlbianYdbServiceName);  
    if (null == ays)  
        return false;  
    byte[] buff = readFile(filename);  
    if (null == buff)  
        return false;  
    String fid = ays.upload(groupname, buff, buff.length, suffix, id);  
    AlbanServiceRouter.getLogger().info(AlbianYdbLogger, "fid:%s.", fid);  
    return true;  
}
```

modify使用方法

```
public static boolean testModify(String groupname, String filename, String suffix, String modifyFilename,
    String modifySuffix, BigInteger id) {
    IAlbianYDBService ays = AlbianServiceRouter.getService(IAlbianYDBService.class,
        IAlbianYDBService.AlbianYdbServiceName);
    if (null == ays)
        return false;
    byte[] buff = readFile(filename);
    byte[] mbuff = readFile(modifyFilename);
    if (null == buff || null == mbuff)
        return false;
    String fid = ays.upload(groupname, buff, buff.length, suffix);
    String mfid = ays.modify(fid, mbuff, mbuff.length, modifySuffix, id);
    AlbianServiceRouter.getLogger().info(AlbianYdbLogger, "fid:%s.mfid:%s.", fid, mfid);
    return true;
}
```

delete使用方法

```
public static boolean testDelete(String groupname, String filename, String suffix) {  
    IAlbianYDBService ays = AlbianServiceRouter.getService(IAlbianYDBService.class,  
        IAlbianYDBService.AlbianYdbServiceName);  
    if (null == ays)  
        return false;  
    byte[] buff = readFile(filename);  
    if (null == buff)  
        return false;  
    String fid = ays.upload(groupname, buff, buff.length, suffix);  
    boolean rc = ays.delete(fid);  
    AlbianServiceRouter.getLogger().info(AlbianYdbLogger, "fid:%s.", fid);  
    return rc;  
}
```