IOServer Java API 设计说明书

文件编号：		项目编号：
项目名称：		部　　门：实时数据采集事业部
版 本 号：2.0	受控状态：0	密　　级：机密
总 页 数：25	正文：22	附　　录：0
编　　制：黄展智 日　　期：2017.4.25	审　　核： 日　　期：	批　　准： 日　　期：

北京亚控科技发展有限公司

IOServer Java API 程序说明文档

时间	文档版本	人员	说明
2016.--.--	V1.0	陈宝泉	创建文档
2017.04.25	V2.0	黄展智	3.6 版本相关修改
2017.06.21	V3.0	黄展智	增加 api 工程环境配置说明

目录

2 简介

2.1 目的

本文档是 IOServer3.53 的高层设计文档，阐述了 IOServer3.53 与客户端交互的具体 JAVA 接口定义。

本文档同时适用与 IOServer3.56。

该文档提供给开发人员，用于指导具体各模块的详细设计。

该文档提供给测试人员，用于设计自动化测试方案。

2.2 范围

IOServer 系统，版本 3.53，同时适用于版本 3.56

3 系统整体设计

3.1 设计前提与依据

根据《IOServer 产品需求》、《IOServer 产品定义》要求，确定如下作为 IOServer3.53 的设计前提与依据。

1、 IOServer 为一独立数据源服务器进程，独立配置的链路、设备和变量，单独运行不依赖其他程序。

2、 IOServer 支持大点数运行，支持同一机器多进程实例运行。

3、 IOServer 可以设置为服务程序后台运行，并提供界面程序。

IOServer 主要是采集现场设备的数据，可供其他客户端通过接口调用数据。此文档旨在说明 JAVA 相关接口。

3.2 系统整体描述

IOServer 是公司产品线中唯一与现场采集设备打交道的自动化软件，它对其他产品屏蔽了采集通道、采集设备等与现场实际运行有关的设备，而对外提供统一的 API 以统一的变量点形式与各应用软件进行数据交互。

IOServer 可以嵌入公司的 IDE 环境运行，也可以供其他客户端访问。

IOServer 可以通过 API、OPC 等方式对外交互，本文档主要针对 Java Api。

3.3 Java API 主要功能

（1） 服务器的连接与断开

（2） 采集对象控制（控制粒度为：变量）

（3） 浏览配置（工程访问）
（4） 同步、异步读写
（5） 订阅读取
（6） 辅助函数

4 接口详细设计

接口文件共包含共有三个包：

- com.ioserver.dll：JNA 映射的 c++接口函数的接口类，客户端类，数据存储类
- com.ioserver.bean：模拟 c++结构体的各种辅助结构类
- com.ioserver.callback:模拟 c++回调函数的回调结构类

4.1 接口类说明

4.1.1 客户端类:IOServerAPICilent

4.1.1.1 IOServerConnecton

```
/// <summary>
///  连接ioserver
/// </summary>
/// <param name="ip">
///         IOServer运行服务器IP
/// </param>
/// <param name="port">
///         IOServer工程端口号
/// </param>
/// <returns>
///         表示连接是否成功：true表示成功，false表示失败
/// </returns>
/// <remarks>
///         端口号可在工程设计网络配置中查看与配置
/// </remarks>
public boolean IOServerConnecton(String ip,int port);
```

4.1.1.2 IOServerDisConnect

```
/// <summary>
///  断开客户端与ioserver的连接
/// </summary>
/// <param name="Handle">
///         IOServer连接句柄
/// </param>
/// <returns>
```

```
///        表示是否断开：0表示成功，-1表示失败
/// </returns>
/// <remarks>
///        句柄可以从客户端getHandle方法获取
/// </remarks>
public int IOServerDisConnect(int Handle);
```

4.1.1.3 GetDeviceWorkStatus

```
/// <summary>
///        获取设备状态
/// </summary>
/// <param name="Handle">
///        IOServer连接句柄
/// </param>
/// <param name="DeviceName">
///        设备名称
/// </param>
/// <returns>
///        0：表示设备正常；1：表示系统控制挂起；2：表示设备故障
         -1:表示读取失败
/// </returns>
/// <remarks>
///        设备名称也就是在ioserver结构树上通道下面的设备名称，是唯一的
/// </remarks>
public  int GetDeviceWorkStatus(int Handle,String DeviceName);
```

4.1.1.4 SubscribeTagValuesChange

```
/// <summary>
///        订阅变量
    /// </summary>
/// <param name="Handle">
///        IOServer连接句柄
/// </param>
/// <param name="TagIDs">
///        变量ID数组
/// </param>
/// <param name="length">
///        数组长度
/// </param>
/// <returns>
///        0：订阅成功，其他表示失败
/// </returns>
/// <remarks>
///        订阅之后需要注册回调函数才可以回去订阅的值
```

```
    /// </remarks>
public int SubscribeTagValuesChange(int Handle,int[] TagIDs,int
length);
```

4.1.1.5 SyncReadTagsValueByIDs

```
    /// <summary>
    ///         以变量ID同步读取变量值
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <param name="TagIDs">
    ///         变量ID数组
    /// </param>
    /// <param name="length">
    ///         数组长度
    /// </param>
    /// <param name="DataSource">
    ///         数据源：缓存与设备
    /// </param>
    /// <returns>
    ///         读取的变量值数组
    /// </returns>
    /// <remarks>
    ///         Struct_TagInfo是模拟c++变量结构体。此方法为批量同步读值
    /// </remarks>
public Struct_TagInfo[] SyncReadTagsValueByIDs(
       int Handle,int[] TagIDs,int length,int DataSource);
```

4.1.1.6 SyncReadTagsValueByNames

```
    /// <summary>
    ///         以变量名称同步读取变量值
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <param name="TagNames">
    ///         变量名称数组
    /// </param>
    /// <param name="length">
    ///         数组长度
    /// </param>
    /// <param name="DataSource">
    ///         数据源：缓存与设备
```

```
    /// </param>
    /// <returns>
    ///        读取的变量值数组
    /// </returns>
    /// <remarks>
    ///        Struct_TagInfo是模拟c++变量信息结构体类。此方法为批量同步读值
    /// </remarks>
public Struct_TagInfo[] SyncReadTagsValueByNames(int Handle,WString[]
TagNames,int length,int DataSource);
```

4.1.1.7 AsyncReadTagsValueByIDs

```
    /// <summary>
    ///        以变量ID异步读取变量值
        /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <param name="TagNames">
    ///        变量ID数组
    /// </param>
    /// <param name="length">
    ///        数组长度
    /// </param>
    /// <param name="DataSource">
    ///        数据源：缓存与设备
    /// </param>
    /// <returns>
    ///        读取请求成功执行与否。0表示成功，其他表示失败
    /// </returns>
    /// <remarks>
    ///        此为批量异步读取变量值，返回结果在读完成回调接口的实现中
    /// </remarks>
   public int AsyncReadTagsValueByIDs(
 int Handle,int[] TagIDs,int length,int DataSource);
```

4.1.1.8 AsyncReadTagsValueByNames

```
    /// <summary>
    ///        以变量名称同步读取变量值
        /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <param name="TagNames">
    ///        变量名称数组
```

```
    /// </param>
    /// <param name="length">
    ///         数组长度
    /// </param>
    /// <param name="DataSource">
    ///         数据源：缓存与设备
    /// </param>
    /// <returns>
    ///         读取的变量值数组
    /// </returns>
    /// <remarks>
    ///         此为批量异步读取变量值，返回结果在读完成回调接口的实现中
    /// </remarks>
public int AsyncReadTagsValueByNames(int Handle,
    WString[] TagNames,int length,int DataSource);
```

4.1.1.9 SyncReadTagsValueReturnNames

```
    /// <summary>
    ///         以变量名称同步读取变量值
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <param name="TagNames">
    ///         变量名称数组
    /// </param>
    /// <param name="length">
    ///         数组长度
    /// </param>
    /// <param name="DataSource">
    ///         数据源：缓存与设备
    /// </param>
    /// <returns>
    ///         读取的变量值数组
    /// </returns>
    /// <remarks>
    ///         Struct_TagInfo_AddName是在Struct_TagInfo基础上添加了变量的名称。此
            方法为批量同步读值
    /// </remarks>
public Struct_TagInfo_AddName[] SyncReadTagsValueReturnNames(
int Handle,WString[] TagNames,int length,int DataSource)
```

4.1.1.10 SyncWriteTagsValueByIDs

```
    /// <summary>
```

```
        ///         以变量ID同步写入变量值
            /// </summary>
        /// <param name="Handle">
        ///         IOServer连接句柄
        /// </param>
        /// <param name="valuelist  ">
        ///         要写入的变量值数组
        /// </param>
        /// <param name="TagIDs">
        ///         变量ID数组
        /// </param>
        /// <param name="length">
        ///         数组长度
        /// </param>
        /// <returns>
        ///         写入成功与否：0表示写入成功，其他表示失败
        /// </returns>
        /// <remarks>
        ///         采用add方式向valuelist存程序，此为批量写入
        /// </remarks>
public int SyncWriteTagsValueByIDs(int Handle,
List<E> valuelist,int[] TagIDs);
```

4.1.1.11 SyncWriteTagsValueByNames

```
        /// <summary>
        ///         以变量名称同步写入变量值
            /// </summary>
        /// <param name="Handle">
        ///         IOServer连接句柄
        /// </param>
        /// <param name="   valuelist  ">
        ///         要写入的变量值数组
        /// </param>
        /// <param name="TagNames">
        ///         变量名称数组
        /// </param>
        /// <param name="length">
        ///         数组长度
        /// </param>
        /// <returns>
        ///         写入成功与否：0表示写入成功，其他表示失败
        /// </returns>
        /// <remarks>
        ///         valuelist是模拟c++变量值结构体的类，此为批量写入
```

```
    /// </remarks>
public int SyncWriteTagsValueByNames(int Handle,
List<E> valuelist,WString[] TagNames)
```

4.1.1.12 AsyncWriteTagsValueByIDs

```
    /// <summary>
    ///         以变量ID异步写入变量值
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <param name="  valuelist ">
    ///         要写入的变量值数组
    /// </param>
    /// <param name="TagIDs">
    ///         变量ID数组
    /// </param>
    /// <param name="length">
    ///         数组长度
    /// </param>
    /// <returns>
    ///         写入请求成功与否：0表示请求成功，其他表示失败
    /// </returns>
    /// <remarks>
    ///         valuelist是模拟c++变量值结构体的类，此为批量写入
            写入成功的结果在写完成回调接口的实现类中显示
    /// </remarks>
public int AsyncWriteTagsValueByIDs(int Handle,
List<E> valuelist,int[] TagIDs)
```

4.1.1.13 AsyncWriteTagsValueByNames

```
    /// <summary>
    ///         以变量名称异步写入变量值
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <param name="  valuelist ">
    ///         要写入的变量值数组
    /// </param>
    /// <param name="TagIDs">
    ///         变量名称数组
    /// </param>
    /// <param name="length">
```

```
    ///        数组长度
    /// </param>
    /// <returns>
    ///        写入请求成功与否：0表示请求成功，其他表示失败
    /// </returns>
    /// <remarks>
    ///        valuelist是模拟c++变量值结构体的类，此为批量写入
               写入成功的结果在写完成回调接口的实现类中显示
    /// </remarks>
public int AsyncWriteTagsValueByNames(int Handle,
List<E> valuelist,WString[] TagNames)
```

4.1.1.14 RegisterConnectStatusChangedCallbackFunc

```
    /// <summary>
    ///        注册连接状态变化回调接口
        /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <returns>
    ///        注册成功与否
    /// </returns>
    /// <remarks>
    ///        使用默认的回调接口实现
    /// </remarks>
public short RegisterConnectStatusChangedCallbackFunc(
          int Handle);
```

4.1.1.15 RegisterWorkStatusChangedCallbackFunc

```
    /// <summary>
    ///        注册ioserver工作状态变化回调接口
        /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <returns>
    ///        注册成功与否
    /// </returns>
    /// <remarks>
    ///        使用默认的回调接口实现
    /// </remarks>
public short RegisterWorkStatusChangedCallbackFunc(
             int Handle)
```

4.1.1.16 RegisterCollectValueCallbackFunc

```
    /// <summary>
    ///         注册订阅变量值变化回调接口
        /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <returns>
    ///         注册成功与否
    /// </returns>
    /// <remarks>
    ///         使用默认的回调接口实现
    /// </remarks>
public short RegisterCollectValueCallbackFunc(
                int Handle)
```

4.1.1.17 RegisterReadCompleteCallbackFunc

```
    /// <summary>
    ///         注册读完成回调接口
    /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <returns>
    ///         注册成功与否
    /// </returns>
    /// <remarks>
    ///         使用默认的回调接口实现
    /// </remarks>
public short RegisterReadCompleteCallbackFunc(
                int Handle)
```

4.1.1.18 RegisterWriteCompleteCallBackFunc

```
    /// <summary>
    ///         注册写完成回调接口
    /// </summary>
    /// <param name="Handle">
    ///         IOServer连接句柄
    /// </param>
    /// <returns>
    ///         注册成功与否
    /// </returns>
    /// <remarks>
```

```
    ///       使用默认的回调接口实现
    /// </remarks>
public short RegisterWriteCompleteCallBackFunc(
              int Handle)
```

4.1.1.19 BrowserProjects

```
    /// <summary>
    ///       浏览所有的工程属性
    /// </summary>
    /// <param name="Handle">
    ///       IOServer连接句柄
    /// </param>
    /// <param name=" Mask ">
    ///       节点名称，空表示根节点
    /// </param>
    /// <returns>
    ///       工程属性数组
    /// </returns>
    /// <remarks>
    ///       Struct_IOServerProperty是模拟c++工程属性结构体的类， mask一般为空
    /// </remarks>
public Struct_IOServerProperty[] BrowserProjects(
          int Handle,WString Mask)
```

4.1.1.20 BrowserChannels

```
    /// <summary>
    ///       浏览工程下的通道属性
    /// </summary>
    /// <param name="Handle">
    ///       IOServer连接句柄
    /// </param>
    /// <param name=" Mask ">
    ///       工程名称，空表示浏览所有工程的通道
    /// </param>
    /// <returns>
    ///       通道属性数组
    /// </returns>
    /// <remarks>
    ///     Struct_ChannelProperty是模拟c++通道属性结构体的类， mask一般为空
    /// </remarks>
public Struct_ChannelProperty[] BrowserChannels(
              int Handle,WString Mask)
```

4.1.1.21 BrowserDevices

```
    /// <summary>
    ///        浏览通道下设备属性
    /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <param name=" Mask ">
    ///        通道名称，空表示浏览所有通道下设备属性
    /// </param>
    /// <returns>
    ///        设备属性数组
    /// </returns>
    /// <remarks>
    ///        Struct_DeviceProperty是模拟c++工程属性结构体的类
    /// </remarks>
public Struct_DeviceProperty[] BrowserDevices(
            int Handle,WString Mask)
```

4.1.1.22 BrowserCollectTags

```
    /// <summary>
    ///        浏览设备下的变量属性
    /// </summary>
    /// <param name="Handle">
    ///        IOServer连接句柄
    /// </param>
    /// <param name=" Mask ">
    ///        设备名称，空表示浏览所有设备下的变量属性
    /// </param>
    /// <returns>
    ///        变量属性数组
    /// </returns>
    /// <remarks>
    ///        Struct_TagProperty是模拟c++变量属性结构体的类
    /// </remarks>
public Struct_TagProperty[] BrowserCollectTags(
            int Handle,WString Mask)
```

4.1.2 回调数据存储类：ClientDataBean

回调数据存储就是存储客户端类执行某些操作后的数据：连接状态、工作状态、订阅变量值、异步读变量值、异步写结果

4.1.2.1 getConnectionStatus

```
/// <summary>
///     获取连接状态
/// </summary>
/// <returns>
///     最新连接状态
/// </returns>
/// <remarks>
///
/// </remarks>
public int getConnectionStatus();
```

4.1.2.2 getWorkingStatus

```
/// <summary>
///     获取ioserver工作状态
/// </summary>
/// <returns>
///     最新工作状态
/// </returns>
/// <remarks>
///
/// </remarks>
public int getWorkingStatus();
```

4.1.2.3 getTagValueByID

```
/// <summary>
///     根据ID获取订阅变量变化回调值
/// </summary>
/// <param name=" TagID ">
///       变量ID
/// </param>
/// <returns>
///     变量值类模拟指针
/// </returns>
/// <remarks>
///
/// </remarks>
public Struct_TagInfo.ByReference getTagValueByID(Integer TagID)
```

4.1.2.4 getReadComTagValueByID

```
/// <summary>
///     根据ID获取读完成变量变化回调值
/// </summary>
```

```
    /// <param name="  TagID ">
    ///       变量ID
    /// </param>
    /// <returns>
    ///     变量值类模拟指针
    /// </returns>
    /// <remarks>
    ///
    /// </remarks>
public Struct_TagInfo.ByReference     getReadComTagValueByID(Integer
TagID)
```

4.1.2.5 getErrorCodeByID

```
    /// <summary>
    ///     根据ID获取写完成是否成功的错误码
    /// </summary>
    /// <param name="TagID">
    ///       变量ID
    /// </param>
    /// <returns>
    ///     对应变量的错误码；0表示成功，其他表示失败
    /// </returns>
    /// <remarks>
    ///
    /// </remarks>
    public Integer getErrorCodeByID(Integer TagID);
```

4.1.2.6 打印回调值函数

- 打印订阅变量值变化回调值：showTagValues
- 打印读完成回调值：showCompleteTagValues

4.1.3 全局客户端存储类：GlobalCilentBean

全局客户端存储是理由单例存储多个客户端返回的数据，主要功能基本上是：增删改查功能。

4.1.3.1 getClientByHandle

```
    /// <summary>
    ///     根据连接句柄获取客户端回调数据存储对象
    /// </summary>
    /// <param name="Handle">
    ///       连接句柄
    /// </param>
    /// <returns>
    ///     相应客户端回调数据存储对象
    /// </returns>
```

```
    /// <remarks>
    ///
    /// </remarks>
public ClientDataBean getClientByHandle(Integer Handle);
```

4.1.3.2 getTagIDbyName

```
    /// <summary>
    ///     根据变量名称获取变量ID
    /// </summary>
    /// <param name="TagName">
    ///     变量名称
    /// </param>
    /// <returns>
    ///     变量ID
    /// </returns>
    /// <remarks>
    ///     若不存在则为0
    /// </remarks>
public Integer getTagIDbyName (WString TagName);
```

4.2 辅助数据结构类说明

4.2.1 变量值结构体类

用于映射ioserver c++接口要使用的变量值结构体。具体属性如下：

```
public class Struct_TagValue{
    public short ValueType;             //变量值类型
    public Union_DataType TagValue; //变量值共用体模拟类
    public static class ByReference …  //模拟结构体指针
    public static class ByValue …      //模拟结构体对象
}
```

4.2.2 数据类型 Union 类

```
public class Union_DataType {
    public boolean          bitVal;
    public byte          i1Val;
    public short            i2Val;
    public NativeLong       i4Val;
    public long             i8Val;
    public byte             ui1Val;
    public short            ui2Val;
    public NativeLong       ui4Val;
    public long             ui8Val;
    public float            r4Val;
```

```
    public double             r8Val;
    public WString         wstrVal;
}
```

4.2.3 时间戳结构体类

```
    public class Struct_TimeStamp {
    public NativeLong Seconds;
 //自1970/01/01 00:00:00(UTC)以来的秒数
    public short MillSeconds; //毫秒数
    public static class ByReference …
    public static class ByValue …
}
```

4.2.4 变量信息结构体类

```
    public class Struct_TagInfo {
    public int TagID;   //变量ID
    public int TagFeildID; //变量域ID
    public Struct_TagValue.ByValue TagValue; //变量值
    public Struct_TimeStamp.ByValue TimeStamp;
    //变量时间戳
    public int  QualityStamp; //变量质量戳
    public static class ByReference …
    public static class ByValue …
}
```

4.2.5 带有变量名称的变量信息结构体类

同3.2.4，区别在于在类中添加了：

```
public WString TagName; //变量名称
```

4.2.6 变量属性结构体类

```
    public class Struct_TagProperty{
        public NativeLong TagAccessID;  // 变量访问ID
        public WString  TagName;  // 变量名称 :  NAME
        public WString  TagFullName;
        /// 变量名称 :  CHANNEL.DEVICE.GROUP.NAME
        public WString  Description; /// 变量描述
        public short    TagDataType; ///变量数据类型
        public Struct_TagValue.ByValue MaxRawValue ;          /// 变
量原始最大值
        public Struct_TagValue.ByValue MinRawValue;             ///
变量原始最小值
        public Struct_TagValue.ByValue MaxValue;                ///
```

```
变量工程最大值
        public Struct_TagValue.ByValue MinValue;                    ///
变量工程最小值
        public float     DeadBand;   /// 死区百分比
        public Boolean CollectControl;
        /// 采集控制(是否采集标志)
        public NativeLong  CollectInterval;
        public static class ByReference…
        public static class ByValue…
}
```

4.2.7 设备属性结构体类

```
    public class Struct_DeviceProperty {
    public NativeLong  DeviceID;    //设备ID
    public WString     DeviceName;  //通讯卡名
    public WString     DeviceSeries;       //驱动系列
    public WString     DeviceDescription; //设备描述
    public WString  DeviceAddrString;  //设备地址串
    public static class ByReference
    public static class ByValue
}
```

4.2.8 通道属性结构体类

```
public class Struct_ChannelProperty{
    public int      ChannelID;   //采集通道ID
    public WString  ChannelName; //采集通道名称
    public Struct_GUID.ByValue   ClsID;//clsid
    public WString  ChannelDescription;    //采集通道描述
    public WString     ChannelDriver;
    public short    ChannelType; //通道类型,
    public int      Timeout;            //链路超时ms
    public static class ByReference
    public static class ByValue
}
```

4.2.9 工程属性结构体类

略

4.3 回调接口类说明

4.3.1 连接状态回调接口：ConnStatusChangeCallBackInf

略

4.3.2 工作状态回调接口：WorkStatusChangeCallBackInf

略

4.3.3 订阅变量值变化回调接口：TagValueChangeCallBackInf

略

4.3.4 读完成回调接口：ReadCompleteCallBackInf

略

4.3.5 写完成回调接口：WriteCompleteCallBackInf

略

4.3.6 回调接口默认实现说明

- 连接状态变化实现类：ConnStatusCallBackImp
- 工作状态实现类：WorkStatusCallBackImp
- 订阅变量值变化实现类：CollectTagValueCallBackImp
- 读完成实现类：ReadCompleteCallBackImp
- 写完成实现类：WriteCompleteCallBackImp

4.4 变量意义

变量值结构类中变量类型映射关系：

```
final Type[] DATATYPE_TYPES_2 =
{short.class,boolean.class,byte.class,byte.class,
   short.class,short.class,NativeLong.class,
   NativeLong.class,long.class,float.class,
   double.class,WString.class
};
```

数据源类型定义：

```
final int IO_DATASOURCE_CACHE = 0x00; //从缓存中读取
final int IO_DATASOURCE_DEVICE = 0x01;//从设备中读取
```

具体使用见 demo。

5 JAVA 工程使用说明

5.1 函数接口调用

见新版demo使用说明:《Java Demo使用文档》

5.2 工程设置

5.2.1 添加 dll 依赖

将发布文件中的依赖放置到工程目录IOServerAPI/ IOServerJavaAPIcode下面，如图：

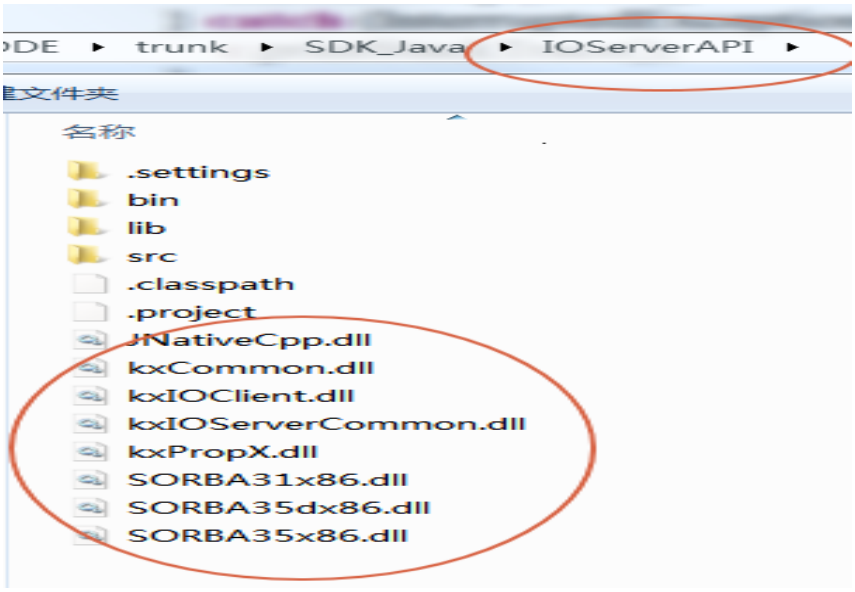

图5-1 添加依赖

注：如果不先添加dll，载入工程后，添加的dll不会被识别，需要重新载入。即把dll放入文件夹后，需要重新载入工程。

5.2.2 载入工程

File->Import->Existing Projects into Workspace ，导入工程：

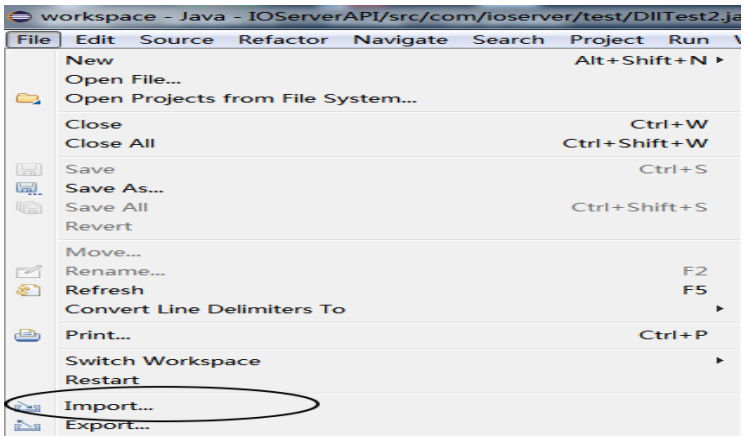

图5-2 导入工程

5.2.3 调试文件

可以是用包com.ioserver.test中的文件进行调试，DllTestMain.java为老版本的demo，该包下的文件调试时都应该根据需要进行修改，不能直接启动调试。另外修改文件后，请保存文件，否则会出现意外崩溃。

5.2.4 打包输出

随便点击工程中任意文件，右键输出，如图，选择jar输出，勾选包com.ioserver.bean,com.ioserver.callback,com.ioserver.dll三个包，然后输出到所需的文件夹即可。默认文件名为IOServerAPI.jar。

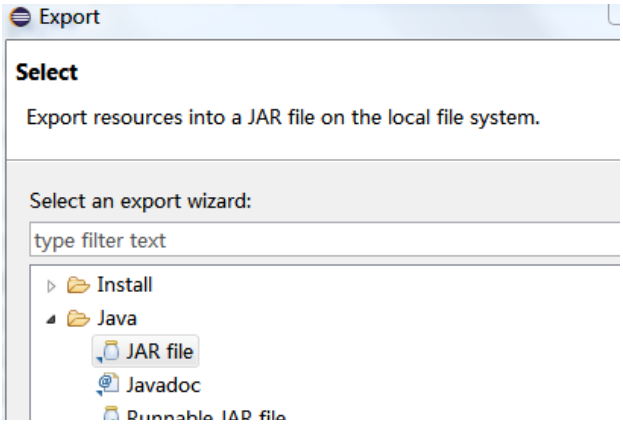

图5-3 打包输出

5.2.5 使用新版 demo 调试

新版demo带有界面。经过上一步打包之后，将打包后的文件IOServerAPI.jar放置到Demo目录下的lib文件中。Lib文件中还必须包含jna调用的两个文件：jna.jar和jNative.jar。

5.2.6 修改版本信息

使用压缩软件打开IOServerAPI.jar封装包，可以看到META-INF，打开该文件既可以修改版本号。