详细设计

xxx-xxx-xxx

前言

本手册为T.B.D使用方法及参考资料的说明。

* 阅读指导

本手册结构如下所示，请根据使用目的，选择适当的章节进行阅读。

需要了解日志模块时

需要了解数据传输时

需要构建高可靠性时

第二章

第一章

第三章

阅读目的

章标题

推荐流程

可选流程

目录

[*1* 日志模块 4](#_Toc354566407)

[1.1 **模块设计说明** 5](#_Toc354566408)

[1.1.1 模块设计原则 5](#_Toc354566409)

[1.1.2 日志模块级别 5](#_Toc354566410)

[1.2 **模块设计概述** 6](#_Toc354566411)

[1.2.1 模块结构划分 6](#_Toc354566412)

[1.3 **模块详细设计** 8](#_Toc354566413)

[1.3.1 类图 8](#_Toc354566414)

[1.3.2 序列图 8](#_Toc354566415)

[1.3.3 成员说明 11](#_Toc354566416)

[1.3.4 函数接口说明 13](#_Toc354566417)

# 日志模块

本章介绍的是日志模块的说明、概述和详细设计。

## **模块设计说明**

日志模块主要用来记录在系统或程序中发生的、要求通知用户的任何重要事情，并在事件查看器中记录应用程序、安全和系统事件。通过使用事件查看器中的日志，可以获取有关硬件、软件和系统组件的信息，并监视远程设备上的安全事件。日志可帮助确定和诊断当前系统问题的根源，还可以帮助预测潜在的系统问题。

### 模块设计原则

1. 日期目录输出。按照日期将日志输出到不同文件，能够应对大数据量存储和对于问题数据精确到日期的日志文件提取。当达到一定的大小，整合后另存到相应文件中。
2. 可分级。对日志进行分级处理，以行的格式表示。
3. 可定位。日志信息出现的位置，行列信息等。
4. 可扩展性。例如将日志扩展到数据库和邮件中。
5. 可控性。对日志的输出级别控制其输出范围。

### 日志模块级别

通过设置配置文件，对每个日志对象都被分配了一个日志优先级别。从低到高分为五类，分别为Debug、Info、Warn、Error、Fatal：

表 1 日志分类

| 分类 | 描述 |
| --- | --- |
| Level DEBUG | DEBUG Level指出细粒度信息事件对调试应用程序是非常有帮助的。 |
| Level INFO | INFO level表明 消息在粗粒度级别上突出强调应用程序的运行过程。 |
| Level WARN | WARN level表明会出现潜在错误的情形。 |
| Level ERROR | ERROR level指出虽然发生错误事件，但仍然不影响系统的继续运行。 |
| Level FATAL | FATAL level指出每个严重的错误事件将会导致应用程序的退出。 |

## **模块设计概述**

### 模块结构划分

分为四个部分，如图 1 模块结果图所示。

图 1模块结构图



#### Logger

日志对象封装类，提供给客户代码日志文件输出接口。

#### Log4net logger

logger对象，指定方式完成日志产生和输出，输出到指定位置（控制台、文件、数据库、邮件等）。

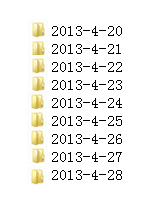
#### Log4net config

Log4net的配置文件，配置logger对象的相关属性。

#### File （日志保存文件）

采集设备的数据日志以日期为单位，用文件夹形式进行保存，如下图所示：

图 2 日志保存方式



## **模块详细设计**

本节介绍的是日志的类图、序列图和接口成员说明。

### 类图

图 3 类图



### 序列图

#### 创建日志文件

图 4创建日志文件



#### 写日志

图 5写日志



#### 日志文件rollingbackup

图 6日志文件rollingbackup



### 成员说明

#### Config设计

<configSections>

<section

name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net-net-1.2.11"></section>

</configSections>

<log4net>

<root>

<level value="WARN"/>

<appender-ref ref="LogFileAppender"/>

<appender-ref ref="RollingFileAppender" />

<appender-ref ref="ConsoleAppender"/>

</root>

<logger name="AppSender.Logging">

<level value="DEBUG"/>

</logger>

<appender name="LogFileAppender" type="log4net.Appender.FileAppender">

<param name="File" value="./log-file.txt" />

<param name="AppendToFile" value="true" />

<layout type="log4net.Layout.PatternLayout">

<param name="Header" value="[Header]" />

<param name="Footer" value="[Footer]" />

<param name="ConversionPattern" value="%d [%t] %-5p %c - %m%n" />

</layout>

<filter type="log4net.Filter.LevelRangeFilter">

<param name="LevelMin" value="DEBUG" />

<param name="LevelMax" value="WARN" />

</filter>

</appender>

<appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">

<!--日志文件路径，“/”与“/”作用相同，到达的目录相同，文件夹不存在则新建 -->

<file value="./Log/" />

<!--是否追加到文件-->

<appendToFile value="true" />

<!--记录日志写入文件时，不锁定文本文件，防止多线程时不能写Log,官方说线程非安全-->

<lockingModel type="log4net.Appender.FileAppender+MinimalLock" />

<!--使用Unicode编码-->

<Encoding value="UTF-8" />

<!--最多产生的日志文件数，超过则只保留最新的n个。设定值value="－1"为不限文件数-->

<maxSizeRollBackups value="-1" />

<!--是否只写到一个文件中-->

<staticLogFileName value="false" />

<!--按照何种方式产生多个日志文件(日期[Date],文件大小[Size],混合[Composite])-->

<param name="RollingStyle" value="Composite" />

<!--按日期产生文件夹和文件名［在日期方式与混合方式下使用］-->

<datePattern value="yyyy-MM-dd/&quot;app.log&quot;" />

<!--可用的单位:KB|MB|GB。不要使用小数,否则会一直写入当前日志-->

<maximumFileSize value="20KB" />

<!--过滤设置，LevelRangeFilter为使用的过滤器。-->

<filter type="log4net.Filter.LevelRangeFilter">

<param name="LevelMin" value="DEBUG" />

<param name="LevelMax" value="WARN" />

</filter>

<!--记录的格式。-->

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%d [%t] %-5p %c - %m%n" />

</layout>

</appender>

#### 读取config设计

AssemblyInfo.cs加入

// 读取log4net配置文件

[assembly: log4net.Config.XmlConfigurator(Watch=true)]

#### Logger设计

### 函数接口说明

表 2函数接口说明

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. | Return value | Function  name | Parameters | | | Function description |
| Name | Type | Description |
| 1 | Logger | getInstance | - | Void | - | 获得单体模式实体 |
| 2 | void | WriteInfoLog | format | String | 格式化输出串 | 写Info级别Log |
| 3 | args | params object[] | 参数 |
| 4 | void | WriteDebugLog | format | String | 格式化输出串 | 写Debug级别Log |
| 5 | args | params object[] | 参数 |
| 6 | void | WriteWarnLog | format | String | 格式化输出串 | 写Warn级别Log |
| 7 | args | params object[] | 参数 |
| 8 | void | WriteErrorLog | format | String | 格式化输出串 | 写Error级别Log |
| 9 | args | params object[] | 参数 |
| 10 | void | WriteFatalLog | format | String | 格式化输出串 | 写Fatal级别Log |
| 11 | args | params object[] | 参数 |
| 12 | void | FuncEntryLog | args | params object[] | 参数 | 函数入口日志，Info级别 |
| 13 | void | FuncExitLog | - | Void | - | 函数退出日志，Info级别 |

# 数据传输模块

本章主要介绍的是数据采集装置和数据中心之间的传输过程

## 数据采集

### 数据采集方式

1. 采集数据内容包括基本信息和监测指标的采集。
2. 数据采集方式包括人工采集方式和自动采集方式。
3. 通过人工采集方式采集的数据为示范项目的基本信息以及需要人工定期填写的监测数据，通过自动采集方式采集的数据为监测指标，由自动计量装置实时采集监测数据，通过自动传输方式实时传输至数据中心。

### 数据编码

## 通讯方式

数据采集装置与数据中心之间的数据通讯应使用基于IP 协议的数据网络，在传输层使用TCP 协议。 数据中心为服务器端，建立TCP 监听，接收来自数据采集装置的链接。数据采集装置为客户端，不启动TCP 监听。数据采集装置启动后向设定好的数据中心发起TCP 连接，TCP 连接建立后保持连接状态不主动断开，数据采集装置定时向数据中心发送心跳包并检测TCP 连接的状态，一旦连接断开则重新建立连接。

## 身份认证

数据采集装置与数据中心建立连接后，数据中心需对数据采集装置通过MD5算法进行身份认证，认证过程如下：

1. TCP 连接建立成功后，数据采集装置向数据中心发送身份认证请求；
2. 数据中心向数据采集装置发送一串由数据中心随机生成的随机序列；
3. 数据采集装置将接收到的随机序列和本地存储的认证用密钥串合并为一连续串，计算该串的MD5值并发送至数据中心；
4. 数据中心将随机序列和本地存储的认证用密钥串合并为一连续串，计算该串的MD5值并与接收到的MD5值进行比较，若相同则发送认证成功至数据采集装置，否则发送认证失败至数据采集装置；
5. 认证用密钥串在数据采集装置和数据中心中都存储在本地文件系统上，可手动进行认证用密钥串的更新。

## 数据远传

数据采集装置到数据中心的远传数据包为XML格式，所有数据采集装置和数据中心之间的数据通信包都需进行AES 加密。AES 加密密钥均存储在数据采集装置和数据中心本地，并且均可手动进行更换。

数据采集装置数据上传模式为主动发送及中心查询两种方式。主动发送模式下时间间隔可通过数据中心进行设定，时间间隔可设置为5 分钟至12小时。每一上传数据均需带有采集时间戳及采集质量码。数据采集装置远传数据发送失败或与数据中心连接断开时，须在本地保存历史数据。重新连接数据中心后进行历史数据恢复，历史数据恢复分为主动恢复和被动恢复两种类型。主动恢复类型在数据采集装置重新连接数据中心后向数据中心发送历史数据恢复请求，数据中心回应允许发送历史数据后方可向数据中心发送历史数据。被动恢复类型则是在数据采集装置接收到来自数据中心的对某一段时间的数据请求后，向数据中心发送该时间段内的历史数据。所有历史数据也均需带有采集时间及采集质量码。