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。

需要了解概述时

需要构建时

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* 日志模块概述 4](#_Toc354501785)

[1.1 **模块简述** 5](#_Toc354501786)

[1.1.1 模块设计原则 5](#_Toc354501787)

[1.1.2 日志模块级别 5](#_Toc354501788)

[1.2 **日志模块设计概述** 6](#_Toc354501789)

[1.2.1 模块结构划分 6](#_Toc354501790)

[1.3 **日志设计** 8](#_Toc354501791)

[1.3.1 类图 8](#_Toc354501792)

[1.3.2 序列图 10](#_Toc354501793)

[1.3.3 成员说明 11](#_Toc354501794)

[1.3.4 函数接口说明 13](#_Toc354501795)

# 日志模块概述

本章介绍的是日志介绍，特性及功能。

## **模块设计说明**

日志是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事件日志，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。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，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）。

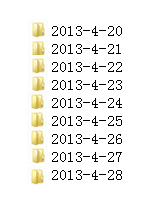
#### Log4net config

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

### 序列图

#### 创建日志文件

图 3创建日志文件



#### 写日志

图 4写日志



#### 日志文件rollingbackup

图 5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 |