

# 智龙集成开发环境 V1.0.005

## 用户操作手册

### 目录

一、安装 .....	2
二、开发前准备 .....	2
三、主窗口 .....	3
四、连接开发板 .....	4
五、程序开发 .....	5
1、项目管理 .....	5
2、代码编辑 .....	5
3、生成/清理 .....	6
4、上传 .....	7
5、执行.py 程序 .....	8
六、串口监视器 .....	8
七、控制台标准输出 .....	10
八、Python 命令交互 .....	10
九、编辑器和扩展组件 .....	11
1、示波器 .....	12
2、代码模板 .....	12
3、计数器 .....	13
4、扩展组件管理 .....	14
十、烧写 .....	15
十一、Python 开发接口 .....	16
1、主窗口和应用程序 .....	17
2、串口 .....	18
3、源码编辑器 .....	19
4、串口监视器 .....	20
5、项目管理器 .....	21
6、.Isproj 项目 .....	21
7、运行环境 .....	22
8、烧写任务 .....	22
9、上传模块 .....	22
10、烧写模块 .....	23
十二、选项设置 .....	23
十三、如何为 IDE 开发扩展组件 .....	24
十四、其它 .....	25

智龙集成开发环境是针对智龙 1C 系列开发板的一个集成式开发环境。本文统一简称为 IDE 或智龙 IDE。

## 一、安装

智龙 IDE 支持多个 Linux 发行版和 Windows 多个版本，如 Loongnix、UOS、银河麒麟、Deepin、Ubuntu、Windows 7/8/10，并支持多种 CPU 架构，如 X86、Mips、LoongArch 等。

Windows 下使用 .msi 文件进行安装，建议最好不要装在 C 盘。安装前检查是否有安全软件、防火墙等限制安装，如果有，先停止相关服务。

Windows 下如果没有安装过 VC++ Redistributable，需要先安装一下。可以到我们公司网站首页 <https://www.bilive.com> 下载安装程序。

安装后，或程序启动时提示“不能启动：连接数据库失败”等信息，是因为安装目录没有访问权限造成的，可以修改目录的访问权限或卸载后选择其它安装目录。

Linux 版本如果是 deb 包，用命令“sudo dpkg -i deb 文件名”安装，如果是 rpm 包，用命令“sudo rpm -ivh 文件名 --nodeps --force”进行安装。

## 二、开发前准备

IDE 使用前需要先准备好相关环境，所需文件及安装设置方式请参考《嵌入式 Linux 系统设计及应用—基于国产龙芯 SoC》一书。

### 1、交叉编译工具

IDE 没有自带交叉编译工具，所以需要自己安装。

### 2、tftp 服务

安装好后，在 windows 下需启动 tftpd32.exe 程序，并设置好目录。在 Linux 下也需先启动 tftp server 服务。

### 3、串口

Windows 下可能需要先安装驱动。Linux 下需要将开发板连上 USB 口之后，查看 /dev 目录下是否有 ttyUSB\* 文件，如果当前用户不在 dialout 组里，需要先把当前用户加到 dialout 组里去，否则程序对这个设备没有访问权限就无法连接成功。

### 4、网络设置

开发板与上位机需要在同一网段，需要先进行设置，否则上传文件会失败。

环境准备好后，用 IDE 中主菜单“工具”—“检查环境”，检查一下是否都已准备好。如果没准备好，显示的信息类似下图：

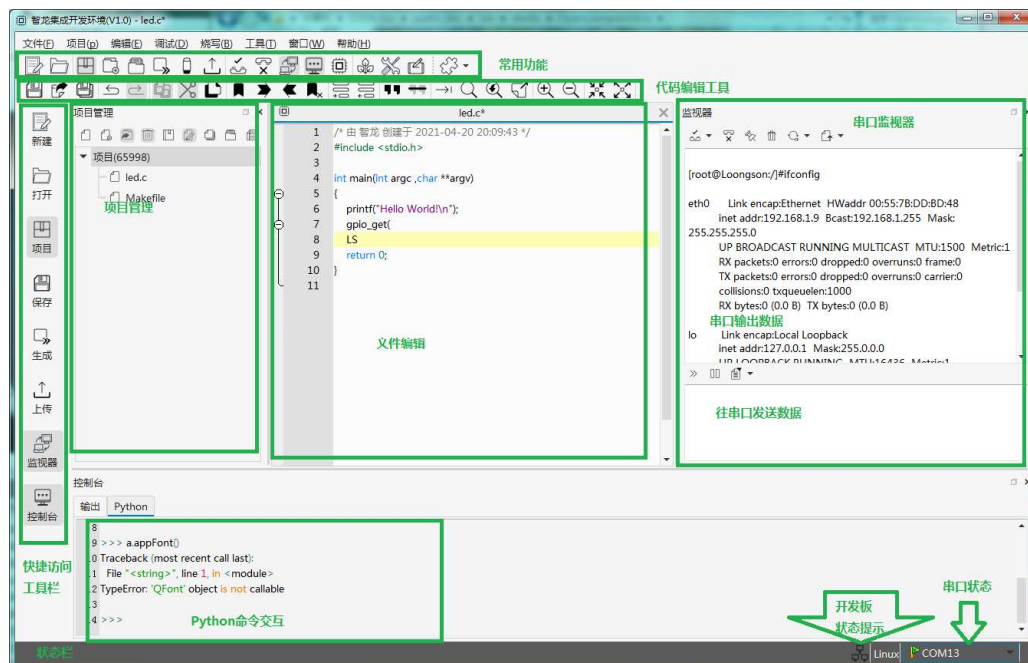


如果已准备好，显示的信息类似下图：



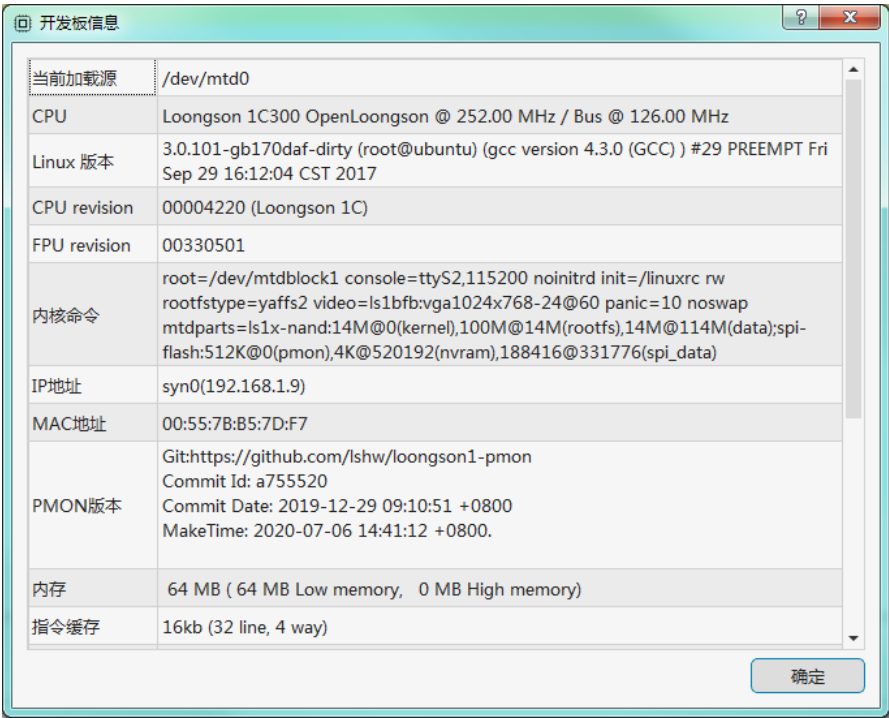
### 三、主窗口

主窗口及各区域说明如下图：



主窗口右下角“开发板状态”会显示开发板网络连接状态及当前是在 Pmon 还是在 Linux 操作系

统终端模式下。双击此处，会弹出开发板信息，如下图。一般只有在程序中连接好串口后重启过一次才看得到完整的信息。



## 四、连接开发板

先确保开发板已通过 USB 连接到电脑，并已连上网线。

再通过以下方式连接串口：

- 1、在主菜单“调试”中选择“连接串口”
- 2、在“监视器”窗口点击工具栏上的按钮“连接串口”
- 3、在主窗口右下角串口列表控件上点击右键，在弹出菜单中选择“连接串口”。

在弹出的串口设置窗口设置正确的连接参数后点“连接”。

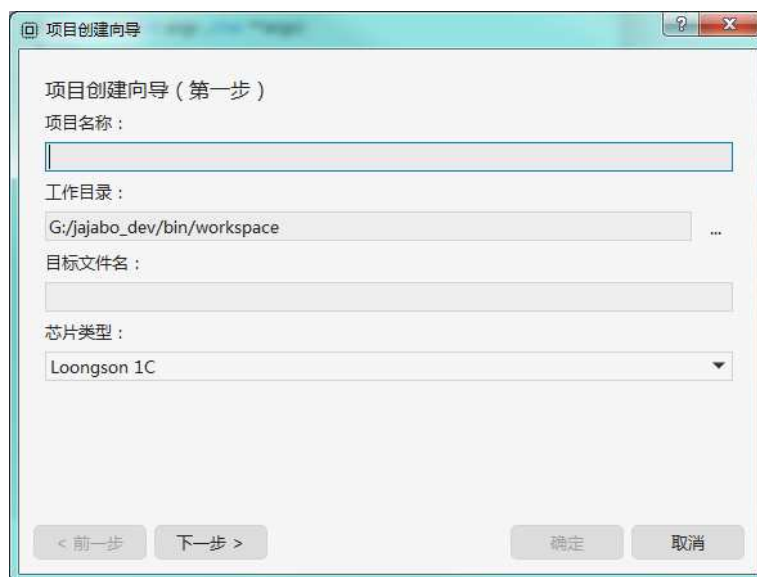


如果开发板尚未上电，就先上电源线。在“监视器”中就可以看到输出信息。

# 五、程序开发

## 1、项目管理

点击主菜单“项目” - “新建项目”，弹出以下窗口：



按界面提示，即可创建一个新项目。

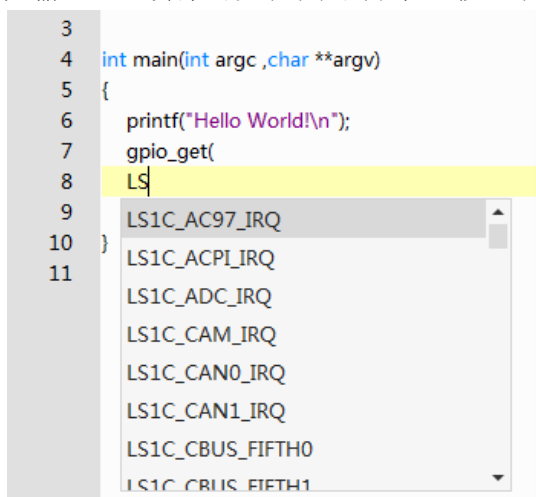
在主窗口左侧的“项目管理”停靠窗口中，可以看到打开的项目中的内容并进行各项管理。

## 2、代码编辑

IDE 提供的代码编辑功能，包括：

- 1) 语法着色
- 2) 自动完成

如输入 LS，会弹出以下下拉列表框，按上下键可进行选择，按 **TAB** 键完成输入



### 3) 调用提示

系统中集成了 OpenLoongsonLib1c 和 OpenLoongsonLib1B 中各.h 文件中的函数定义、宏定义等，在代码编辑中会提示调用方式，如下图所示：

```
3
4 int main(int argc ,char **argv)
5 {
6     printf("Hello World!\n");
7     gpio_get|
8     gpio_get(unsigned int gpio) -> unsigned int
9 }
10
```

### 4) UNDO/REDO

### 5) 代码块折叠展开

### 6) 标签及定位

### 7) 查找和替换

### 8) 字体缩放

### 9) 注释/取消注释

### 10) 增加/减少缩进

### 11) 指定行定位

### 12) 多文档窗口管理

### 13) 自动语法检查

IDE 会对.c 程序自动进行语法检查，有错误的语句，会在下方使用红色波浪线进行标注，鼠标移动此处，会提示错误信息，如下图所示：



```
sm001.c
1 /* 由 智龙 创建于 2021-07-23 14:08:30 */
2 #include <stdio.h>
3
4 int main(int argc ,char **argv)
5 {
6     printf("Hello World!\n");
7     sdfewr
8     ret
9 }
10
```

note: each undeclared identifier is reported only once for each function it appears in

## 3、 生成/清理

程序编写完成后，使用主菜单“调试”-“生成”就可以生成目标文件。生成过程中的信息会在“控制台”的“输出”窗口显示，如下图所示：

```
控制台
输出 Python
1 >>> 开始生成项目 [65998] <<<
2 mips-linux-gnu-gcc -Wall -I -IG:/jajabo_dev/bin/devlib/OpenLoongsonLib1c/src/include -IG:/jajabo_dev/bin/devlib/OpenLoongsonLib1c/src/lib -IG:/jajabo_dev/bin/devlib/OpenLoongsonLib1c/src/libc -IG:/jajabo_dev/bin/devlib/OpenLoongsonLib1c/src/libm led.c -o led
3 Makefile:12: recipe for target 'led' failed
4
5 led.c: In function 'main':
6 led.c:7:2: warning: implicit declaration of function 'gpio_get' [-Wimplicit-function-declaration]
7 led.c:8:2: error: 'LS' undeclared (first use in this function)
8 led.c:8:2: note: each undeclared identifier is reported only once for each function it appears in
9 led.c:9:2: error: expected ';' before 'return'
10 led.c:10:1: error: expected ';' before '}' token
11 led.c:10:1: warning: control reaches end of non-void function [-Wreturn-type]
12 mingw32-make: *** [led] Error 1
13
14
15 --- 完成 (代码: 2 状态: 正常) ---
16
17
```

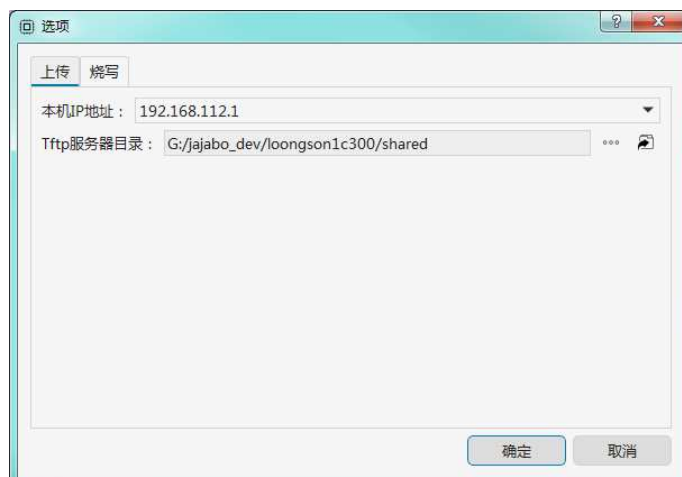
如果编译时报错，双击错误信息所在行，IDE 会自动跳转到出错信息所在的文件及对应的行并高亮显示。

使用主菜单“调试”-“清理”可以执行清理操作。

目前项目中使用 **Makefile** 来处理，在创建新项目时，可以选择按模板自动生成 **Makefile**，可以对之进行修改。

## 4、上传

上传前需要先设置选项，在主窗口中选择“烧写”-“上传和烧写选项”，出现以下窗口：



需要按正确的 IP 地址和目录进行设置，并确保开发板已连上网线，网络是连通的状态。在“项目属性”中，可以设置生成后自动上传并执行。

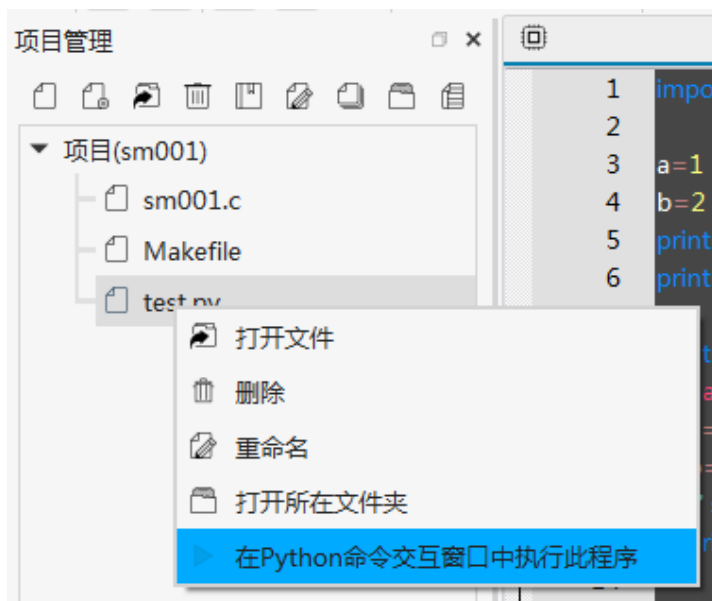


也可以使用串口监视器工具栏上的按钮进行手工上传。

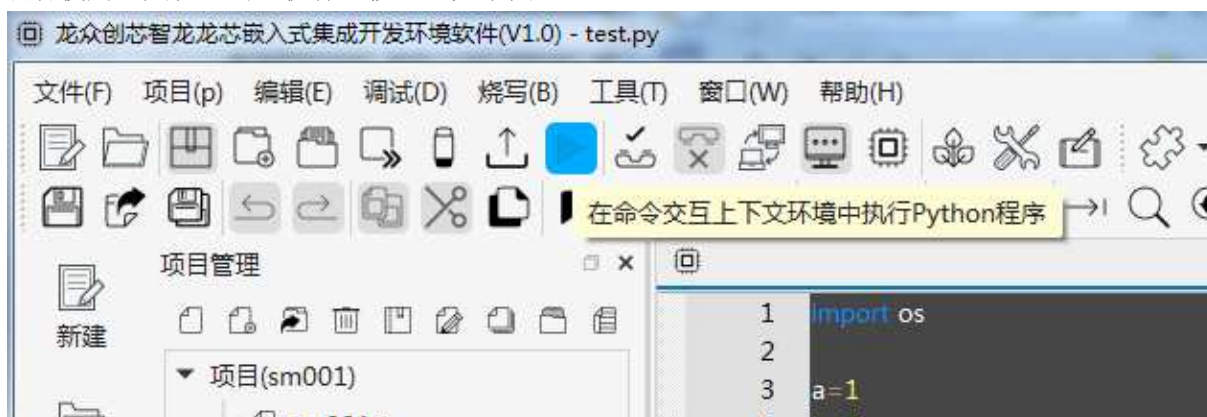
## 5、执行.py 程序

V1.0.005 中，已可以在开发项目（.lsproj）中添加.py 程序，并可在 IDE 中对之进行编辑。这些.py 程序可以在 Python 交互窗口中执行。与通过命令交互的方式相比，这样可以写较长的程序在 IDE 的环境中执行更复杂的程序。

在项目管理中的.py 程序右键菜单中选择“在 Python 命令交互窗口中执行此程序”，就可以执行所选的程序，如下图：



或者使用工具栏上的“执行”按钮，如下图：



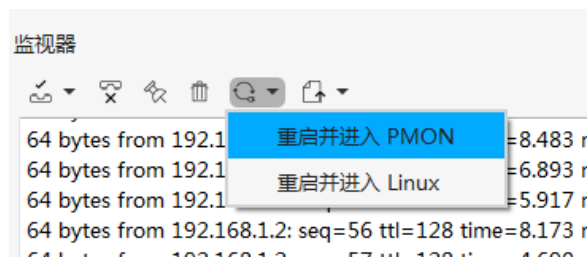
可以执行当前正在编辑的.py 程序。

在这类.py 程序中，可以使用智龙 IDE 提供的接口。

## 六、串口监视器

串口监视器上方的工具栏提供连接、断开连接、锁定滚动、清除内容、重启、上传等按钮。重启可以选择进入 PMON 还是 LINUX。

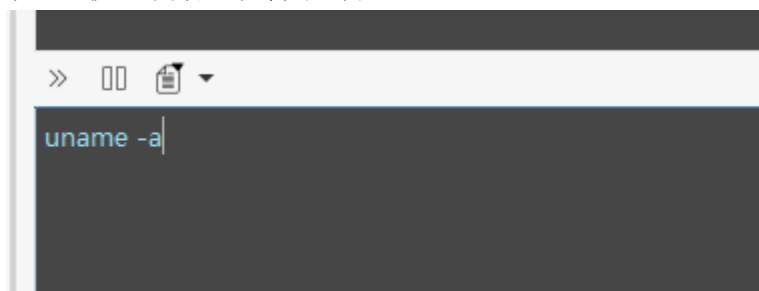




上传按钮会列出 tftp 共享目录下的文件，点击对应的菜单就可立即上传，也可以选择其它目录下的文件，IDE 会将之复制到 tftp 共享目录下后上传。



串口监视器下方是命令交互窗口：



在这里输入要发送的命令后按回车，命令会写入串口。  
可以用工具栏上的按钮调出之前执行过的命令重复执行。

## 七、控制台标准输出

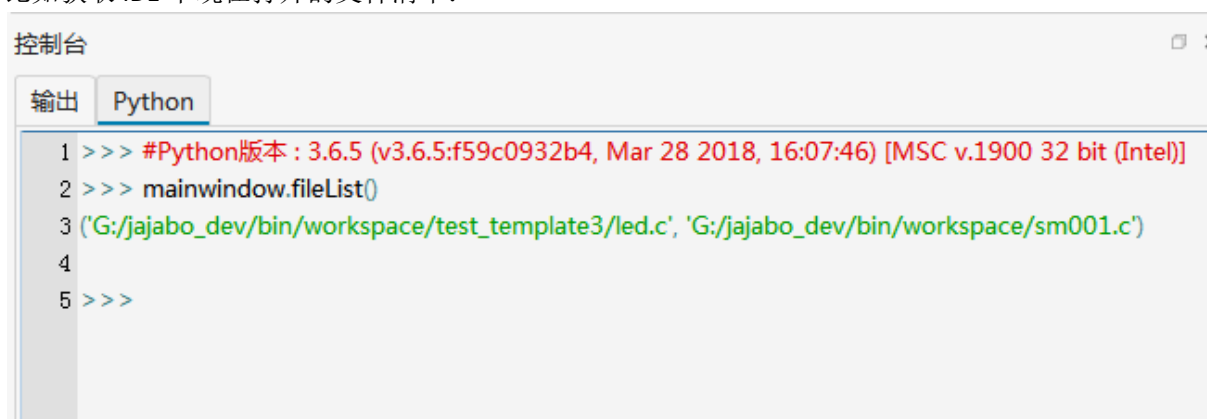
控制台中的标准输出窗口显示 IDE 中各模块运行时的一些提示、报错、警告信息。

## 八、Python 命令交互

控制台中的 Python 命令交互窗口，用于使用 Python 语句访问 IDE 开放的接口、运行 Python 脚本。使用的 Python 版本是 3.6.5。

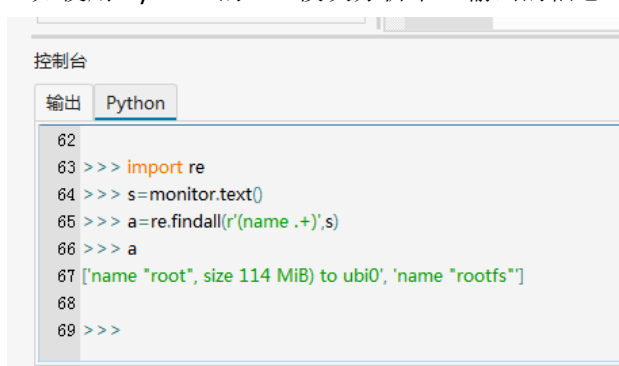
交互窗口输入语句后按回车即可，可以通过上下键调出刚才执行过的语句。

比如获取 IDE 中现在打开的文件清单：



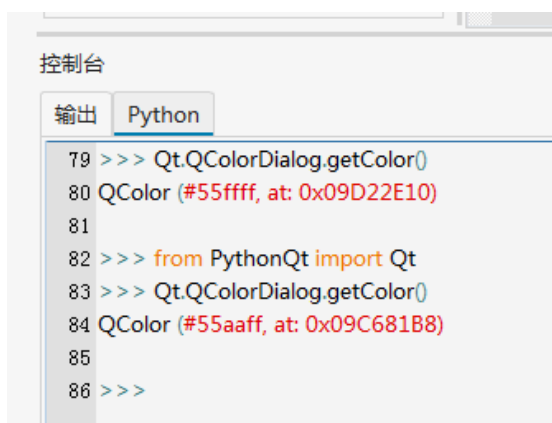
```
控制台
输出 Python
1 >>> #Python版本 : 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)]
2 >>> mainWindow.fileList()
3 ('G:/jajabo_dev/bin/workspace/test_template3/led.c', 'G:/jajabo_dev/bin/workspace/sm001.c')
4
5 >>>
```

比如使用 Python 的 re 模块分析串口输出的信息：



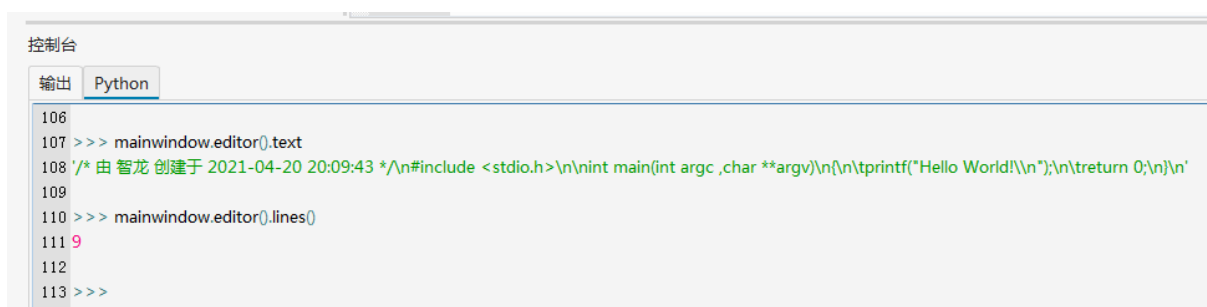
```
控制台
输出 Python
62
63 >>> import re
64 >>> s=monitor.text()
65 >>> a=re.findall(r'(name .+)',s)
66 >>> a
67 ['name "root", size 114 MiB) to ubi0', 'name "rootfs"']
68
69 >>>
```

比如使用 Qt 库 中的 QColorDialog 弹出一个选择颜色的对话框以便选择一个颜色值：



```
控制台
输出 Python
79 >>> Qt.QColorDialog.getColor()
80 QColor (#55ffff, at: 0x09D22E10)
81
82 >>> from PythonQt import Qt
83 >>> Qt.QColorDialog.getColor()
84 QColor (#55aaff, at: 0x09C681B8)
85
86 >>>
```

比如读取主窗口当前正编辑的子窗口的文本内容及行数：



```
控制台
输出 Python
106
107 >>> mainwindow.editor().text
108 /* 由 智龙 创建于 2021-04-20 20:09:43 */\n#include <stdio.h>\n\nint main(int argc, char **argv)\n{\n    printf("Hello World!\\n");\n    return 0;\n}\n
109
110 >>> mainwindow.editor().lines()
111 9
112
113 >>>
```

随安装包打包的 Python 第三方库参考文档：<http://api.bilive.com/#/1-10-python>

可使用的 Qt 库包括 'QtCore', 'QtGui', 'QtMultimedia', 'QtNetwork', 'QtOpenGL', 'QtSql', 'QtSvg', 'QtUiTools', 'QtXml', 'QtXmlPatterns', 'Qt namespace', 'QWidget' 等。参考文档：<http://api.bilive.com/#/1-11-qt>

也可以使用其它第三方 Python 库，具体操作方法参考 [bilive.com](http://api.bilive.com) 官方文档。

## 九、编辑器和扩展组件

IDE 有很强的可扩展性，架构设计为可以使用多种文件编辑器。目前发布的版本中只包括 C 程序编辑器、Makefile 编辑器，将来会随需要发布各种编辑器。

IDE 还支持扩展组件，也称插件。

文件编辑器以及扩展组件，都是使用的 PFF 文件，基于我们公司的 PFF 应用框架。

在插件中可以访问 IDE 开放的接口与 IDE 进行集成，也可以调用丰富的 Python 库和 Qt 库，比如用于代码模块、自动化任务、程序跟踪调试、开发过程管理、GIT 和 SVN 的集成等。

这些插件都是“即插即用”的，不需要重启 IDE。插件的升级只需要打开同一插件新版本 PFF 文件就可以了。

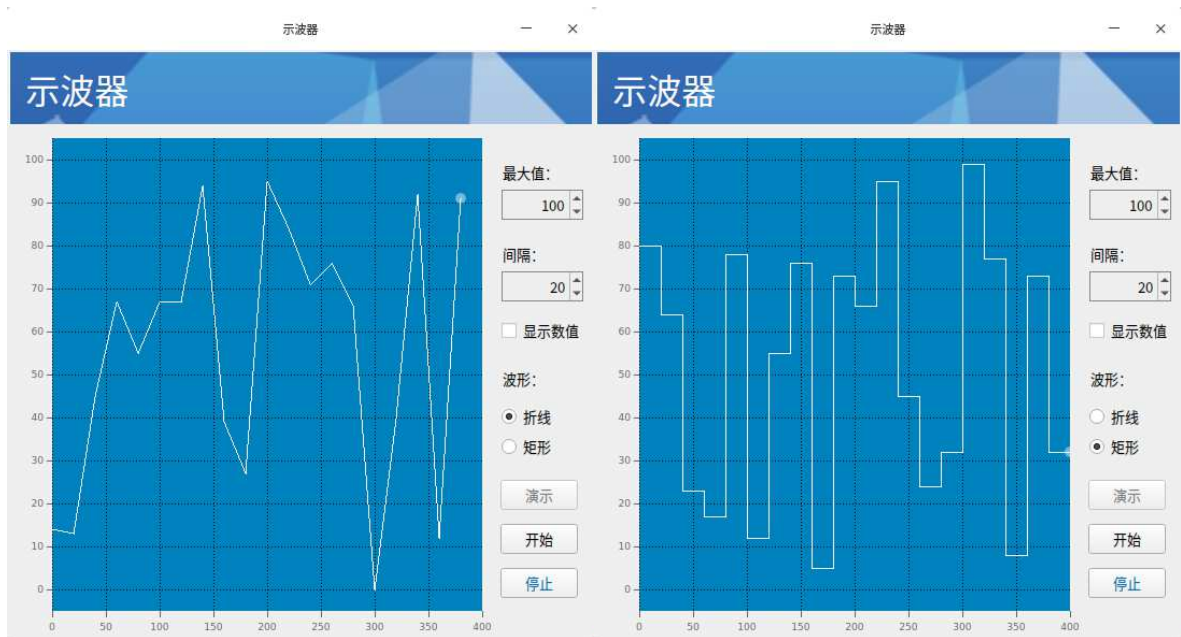
PFF 使用我们公司的开发工具 biForm 进行开发，具体参考公司网站：<https://www.bilive.com>。

biForm 开发参考文档: <http://api.bilive.com/#/>

在安装目录下的 `extensions` 子目录, 有几个扩展组件的示例, 使用主菜单“工具”-“扩展组件”就可以添加这些插件。

## 1、示波器

示波器从串口监视器实时读取数据, 以波形动态显示这些时序数据, 以方便开发者对程序结果进行观察。演示效果如下图所示:



## 2、代码模板

开发者可以自己设置代码模板, 模板中可设置变量, 在使用时只需要输入变量的值, 就可自动生成代码, 代码可直接插入到代码编辑器中, 或复制到剪切板, 以提高代码编写的效率。



### 3、计数器

计数器用于对串口接收到的数据进行统计,可设置统计的内容,比如统计“error”出现的次数,可用于程序运行结果的跟踪调试。



## 4、扩展组件管理

“扩展组件管理”本身也是一个扩展组件，用来对各个扩展组件进行管理。比如禁用（或恢复）某些组件、查看组件的版本及说明、或直接运行某个组件，都可通过“扩展组件管理”实现。



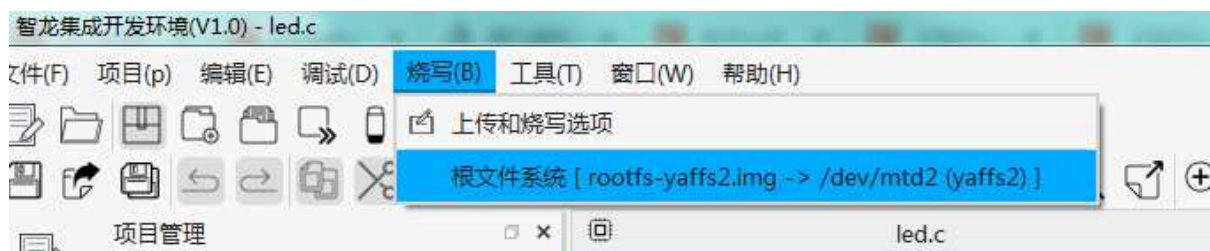
扩展组件的升级也很简单，直接打开新版本的 PFF 文件即可完成扩展组件的升级。

## 十、烧写

在主菜单“工具”-“烧写和上传选项”中可以管理烧写任务：



设置好的烧写任务，在“主菜单”-“烧写”中可以看到下拉菜单项：



点击，会提示是否进行烧写：



选择“是”就会开始启动烧写任务。

## 十一、Python 开发接口

IDE 开放了“应用程序主窗口”及“串口监视器”的 Python 访问接口。用户可以在“控制台”的“Python 命令交互”窗口通过 Python 命令以及在开发插件时通过 `import IDE` 模块访问这些接口。V1.0.001beta 版只开放了这两个内部对象，以后还将开放与上传、烧写、调试等相关的更多接口。除了可以访问这两个内置对象的接口，还可以使用各种 Python 内置功能，及其它第三方 Python 库。

IDE 安装目录下可以看到随安装程序发布的 Python 第三方库，也可以参考 biForm 官方网站 <https://www.bilive.com> 提供的“biForm 开发参考”。





## 1、主窗口和应用程序

“应用程序主窗口”通过 `mainwindow` 对象访问，IDE 为访问提供了以下接口：

方法	说明
<code>void showMonitor()</code>	显示监视器窗口
<code>void hideMonitor()</code>	关闭监视器窗口
<code>void showConsole()</code>	显示控制台窗口
<code>void hideConsole()</code>	隐藏控制台窗口
<code>void createNew()</code>	新建一个源程序文档
<code>void openFile(const QString&amp; &amp;filename)</code>	打开程序文件
<code>void closeCurrentSubWindow()</code>	关闭当前正在编辑的源程序
<code>SerialPortDelegate* serialPort()</code>	返回当前正在使用的串口对象
<code>codeEditorDelegate* editor()</code>	返回当前正在使用的代码编辑器对象
<code>void openPlugins(const QString&amp; UUID)</code>	打开指定的扩展组件
<code>void statusBarMessage(const QString&amp; msg)</code>	在主窗口状态栏显示信息
<code>void stdOutput(OutputType type,const QString&amp; module,const QString&amp; msg )</code>	在标准输出窗口显示信息
<code>void systemTrayMessage(const QString&amp; msg, SystemTrayIcon icon=Nolcon, int millisecondsTimeoutHint =10000)</code>	显示系统任务栏消息
<code>void viewBoardInformation()</code>	查看开发板信息
<code>QStringList fileList(bool fullpath = true)</code>	当前打开的源程序文件清单
<code>QStringList fileHistory()</code>	打开的源程序文件历史记录
<code>QStringList projectHistory()</code>	打开的项目历史记录
<code>void closeAllWindows()</code>	关闭所有子窗口
<code>void arrangeTitled()</code>	子窗口并排显示
<code>void arrangeCascade()</code>	子窗口层叠显示
<code>void nextWindow()</code>	切换到下一子窗口
<code>void previousWindow()</code>	切换到上一子窗口
<code>QStringList editorUUIDList()</code>	IDE 中注册的编辑器 UUID 清单
<code>CodeStyle codeEditorStyle()</code>	代码编辑器样式
<code>void setCodeEditorStyle(CodeStyle style)</code>	设置代码编辑器样式
<code>QString appName()</code>	应用程序名称
<code>QString appVersion()</code>	应用程序当前版本号
<code>QString appTitle()</code>	应用程序标题
<code>QString appPath()</code>	应用程序安装目录
<code>QString dataPath()</code>	应用程序保存数据文件的目录
<code>void build()</code>	构建当前项目（生成）
<code>void clean()</code>	清理当前项目

<code>void cleanAndBuild()</code>	清理并重新生成
-----------------------------------	---------

属性读取的用法如：`mainwindow.appFont`

修改属性的用法如：`mainwindow.workspace="d://test_path"`

属性	说明
<code>appFont</code>	返回应用程序缺省字体
<code>monitorFont</code>	返回监视器缺省字体
<code>pythonFont</code>	返回 <code>python</code> 命令交互窗口缺省字体
<code>stdoutFont</code>	返回控制台字体
<code>codeFont</code>	返回代码编辑器缺省字体
<code>workspace</code>	返回缺省的工作区目录
<code>monitorForegroundColor</code>	返回监视器前景色
<code>monitorBackgroundColor</code>	返回监视器背景色

枚举类型	说明
<b>OutputType:</b> NotDefine = 0, Information = 1, Warning = 2, Critical = 3, Question=4,	调用 <code>stdOutput</code> 时使用，表示输出的信息在类型
<b>SystemTrayIcon :</b> Nolcon = 0, InformationIcon = 1, WarningIcon = 2, CriticalIcon = 3,	调用 <code>systemTrayMessage</code> 时使用，表示消息气泡中使用的图标类型
<b>CodeStyle :</b> brightbackground = 0, darkbackground =1,	编辑器样式

## 2、串口

其中 `mainwindow.serialPort()` 返回的串口对象，提供以下访问接口：

属性	说明
<code>name</code>	串口的名称，只读，字符串类型

方法	说明
<code>qint32 baudRate()</code>	波特率的设置值
<code>QString stringBaudRate()</code>	波特率的描述文字
<code>QSerialPort::DataBits dataBits()</code>	数据位的设定值

QString stringDataBits()	数据位的描述文字
QSerialPort::StopBits stopBits()	停止位的设定值
QString stringStopBits()	停止位的描述文字
QSerialPort::FlowControl flowControl()	流控制的设定值
QString stringFlowControl()	流控制的描述文字
QSerialPort::Parity parity()	奇偶校验的设定值
QString stringParity()	奇偶校验的描述文字
bool isOpen()	是否已打开

### 3、源码编辑器

mainwindow.editor()则返回的是代码编辑器控件，接口很丰富，并且可以通过连接信号和槽，对代码的一些编辑事件做出响应。具体请参考 **biForm** 的相关文档。一些较常能用到的接口如下：

调用接口	说明
setFont(font)	设置缺省字体
setTabWidth(width)	设置 <b>tab</b> 宽度
append(text)	在最后追加文本
clear()	清除所有内容
copy()	复制所选内容
cut()	剪切所选内容
ensureCursorVisible()	确保当前输入光标可见
ensureLineVisible(line)	确保某行可见
foldAll()	折叠起所有代码块
foldLine(line)	折叠某行
Indent(line)	指定某行缩进
Insert(text)	在当前光标处插入文字
insertAt(text,line,index)	在指定行、指定位置处插入文字
paste()	从剪切板粘贴
redo()	重做上一个操作
removeSelectedText()	删除选中的文本
replaceSelectedText(text)	替换选中的文本
selectAll(True)	选择所有
selectAll(False)	撤消所有选择
setText(text)	设置全部文本
undo()	撤消上一步操作
unindent(line)	减少指定行的缩进
zoomIn()	放大
zoomOut()	缩小
length()	总字符数
lines()	总行数
text(line)	指定行的文本内容
text(fromline,toline)	指定几行的文本内容
selectedText()	当前选中的文本内容

gotoLine(line)	转到第几行
currentRow()	当前光标所在行

#### 信号:

可将信号通过 connect 函数连接到 Python 函数，比如以下语句

```
mainwindow.editor().connect('textChanged',somePythonFunction)
```

就可以连接文本改变的信号和一个 Python 函数 somePythonFunction，这样，当代码编辑器中的文本发生改变时，调用这个 Python 函数进行处理。

cursorPositionChanged(int line, int index)	光标位置发生改变时
copyAvailable(bool yes)	可复制文本时
linesChanged()	光标所在行发生改变时
selectionChanged()	选择范围发生改变时
textChanged()	文本内容发生改变时

## 4、串口监视器

内置对象 monitor 提供的接口如下:

调用接口 (方法)	说明
void setPin(bool checked)	设置是否冻结滚动
void setTimeStamp(bool checked)	设置是否添加时间
void clear()	清除所有内容
QString text()	返回所有文本内容
void copy()	复制所选内容
void zoomIn(int range=1)	放大
void zoomOut(int range=1)	缩小
void connectSerialPort()	连接到串口
void disconnectSerialPort()	断开串口连接
QString textOfLastRow()	最后一行的文本
void sendCommand(const QString& cmd)	发送命令
void setLocalEchoEnabled(bool set)	设置是否本地回显

#### 信号:

信号的使用方法与 mainwindow.editor()类似。比如可以通过

```
monitor.connect('getData(QByteArray)',somePythonFunction)
```

在串口接收到数据时，调用 somePythonFunction 进行处理。

信号	说明
getData(const QByteArray& data)	串口接收到数据时发出此信号
getLine(const QString &text)	串口接收到一行新的数据时发出此信号
cursorPositionChanged()	光标位置发生改变时
copyAvailable(bool yes)	可复制文本时
selectionChanged()	选择范围发生改变时

textChanged()	文本内容发生改变时
---------------	-----------

## 5、项目管理器

“项目管理器”通过 `projectManager` 对象访问，这个对象提供了以下接口：

方法	说明
<code>ProjectDelegate* currentProject()</code>	返回当前项目
<code>void openProject(const QString&amp; filename)</code>	打开项目文件

信号	说明
<code>void projectLoaded()</code>	项目加载后发出此信号

## 6、.lsproj 项目

通过 `projectManager` 对象的接口 `currentProject()` 返回的 `ProjectDelegate*` 对象，用于访问这个项目的一些详细信息：

属性	说明
<code>empty</code>	是否有打开项目，如果没有，这个属性值为 <code>True</code> ，否则为 <code>False</code> ，布尔型
<code>name</code>	项目名称，字符串类型
<code>filename</code>	项目文件 <code>.lsproj</code> 全路径文件名，字符串类型
<code>workspace</code>	项目工作区目录，字符串类型
<code>includePath</code>	包含头文件目录清单，字符串列表类型
<code>targetFileName</code>	生成目标文件名，字符串类型

方法	说明
<code>QVariantList itemList()</code>	<p>项目中包含的所有文件和过滤器清单</p> <p>返回的值是 Qt 的 <code>QVariantList</code> 类型，对应 Python 中 <code>tuple</code> 类型，如：</p> <pre>(( '项目(sm001)', 1, 'G:/jajabo_dev/bin/workspace/sm001.lsproj', 0), ('sm001.c', 3, 'G:/jajabo_dev/bin/workspace/sm001.c', 1), ('Makefile', 3, 'G:/jajabo_dev/bin/workspace/Makefile', 1))</pre> <p>其中每个元素又是一个 <code>tuple</code>，分别为“节点标题、类型、对应文件、层级”。</p> <p>其中类型为 1 表示是根节点，为 2 表示是过滤器，为 3 表示是文件，为 0 表示错误。</p>

## 7、运行环境

通过 `envManager` 对象可以访问 IDE 运行环境的一些接口：

方法	说明
<code>QString gccVersion()</code>	检测到的 <code>gcc</code> 的版本
<code>QString mingw32Version()</code>	检测到的 <code>mingw32</code> 的版本
<code>bool tftpstarted()</code>	检测 <code>tftp</code> 是否已启动
<code>QStringList gccIncludePath()</code>	返回 <code>gcc</code> 缺省的包含目录
<code>bool ready()</code>	环境是否已经准备好
<code>void startCheckEnv()</code>	重新检查环境

## 8、烧写任务

通过 `burnManager` 对象的接口可以添加、删除、执行烧写任务，烧写任务对象类型为 `BurnTaskDelegate`，其提供的接口：

属性	说明
<code>name</code>	任务名称（只读），字符串类型
<code>filename</code>	待烧写的文件名（可读写），字符串类型
<code>partition</code>	烧写到分区（可读写），字符串类型
<code>imageType</code>	镜像文件类型（可读写），返回值是 <code>rootfs kernel pmon</code> 或为空，字符串类型
<code>rootFileSystemType</code>	根文件系统类型（可读写），返回值是 <code>cramfs jffs2 ext4 yaffs2</code> 或为空，字符串类型
<code>args</code>	烧写命令参数（可读写），字符串类型
<code>commandsAfterBurn</code>	烧写后执行命令，可以设置多条命令（可读写），字符串列表类型

方法	说明
<code>QString toString()</code>	返回可读性更好的任务的描述文字
<code>QStringList commands()</code>	自动生成的命令清单
<code>bool save()</code>	保存对属性的修改
<code>bool remove()</code>	删除这个任务
<code>bool execute()</code>	执行这个任务
<code>bool isValid()</code>	任务属性设置是否有效

## 9、上传模块

通过 `uploadManager` 对象访问 IDE 中的上传模块，其提供的接口：

方法	说明
bool uploadFile(const QString& filename, bool exec = false, const QString & args = "", const QString& toPath = "", bool addExecProperty = false)	上传文件 filename 文件名 exec 上传后是否立即执行 args 执行时的参数 toPath 上传到指定目录，为空表示上传到当前目录 addExecProperty 是否自动添加可执行属性
QString hostIP()	开发板的 IP 地址
QString tftpServerPath()	上位机 tftp 服务器共享文件目录
void setLocalhostIP(const QString& ip)	设置上位机 IP 地址
void setTftpServerPath(const QString& path)	设置上位机 tftp 服务器共享文件目录
QStringList getLocalHostIPList()	上位机可用的 IP 地址清单

信号	说明
void tftpServerPathChanged(const QString& path)	tftp 服务器共享文件目录被修改时发出此信号（指通过 IDE 中的选项设置进行修改）
void localhostIPChanged(const QString& ip);	上位机 IP 地址被修改时发出此信号（指通过 IDE 中的选项设置进行修改）

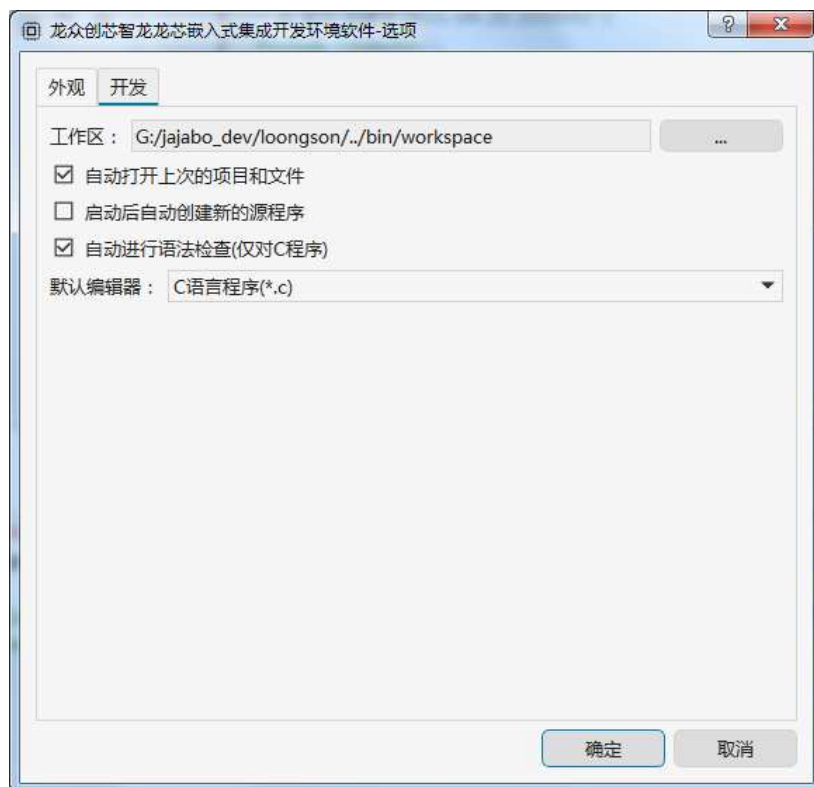
## 10、烧写模块

通过 burnManager 对象访问烧写模块，对应的接口：

方法	说明
BurnTaskDelegate* newTask(const QString& name)	创建一个新的烧写任务，这个还是个空的任務，还不会被添加到任务列表中
bool removeTask(BurnTaskDelegate* task)	删除掉一个烧写任务
QList<BurnTaskDelegate*> taskList()	所有烧写任务清单
BurnTaskDelegate* getTask(const QString& name)	通过名称获取一个烧写任务对象
bool executeTask(const QString& name)	执行指定名称的烧写任务
bool existTask(const QString& name)	通过名称判断是否存在某个烧写任务

## 十二、选项设置

对应用程序的一些选项进行设置：



## 十三、如何为 IDE 开发扩展组件

智龙 IDE 的扩展组件需要使用 biForm 进行开发,可以访问 <https://www.bilive.com> 了解 biForm 的相关信息。



biForm 是一个跨平台的开发工具，使用 Python3 做为脚本语言，并可以使用 Qt5 提供的基础库和 Python 第三方库进行各类应用程序的开发。开发的程序都以 PFF 文件发布，扩展名为 PFF。

使用 biForm 为 IDE 开发扩展组件与开发其它应用程序没有特别大的差别。

但需要注意一点，智龙 IDE 中的所有开放的对象和接口都在一个名为 IDE 的模块里。扩展组件如果需要访问这些接口，需要先 import IDE。但在 biForm 试运行环境中，是没有这个 IDE 模块的，所以直接在 biForm 中试运行，因为不能导入这个模块，可能会报错，或者得不到希望的结果。

因此需要在开发程序的过程中注意技巧，可以在代码中用：

```
if not this.form.isDebug():  
    from IDE import *
```

这样的方式，这样在 biForm 试运行时就不会因 import 报错。其它与 IDE 模块有关的代码也可以做类似处理。

这样，在 biForm 中可以在不使用 IDE 模块的情况下进行试运行，在其它内容调试完成后，再将打包发布的 PFF 在智龙 IDE 中进行测试和调试。

在测试过程中可以通过智龙 IDE 的 Python 交互窗口进行实时调试。

## 十四、其它

IDE 后台有 SQLite 数据库，会保存用户安装的插件、编辑器等一些数据，有些插件管理的数据（如代码模板）也都保存在数据库中。应用程序的选项设置等都保存在 loongson\_history.Config 文件中。

如果需要换一台电脑，将数据库文件和 Config 文件复制到新电脑就可以还原，不用重新安装插件或者重新设置。

南京龙众创芯电子科技有限公司  
武汉百利孚信息科技有限公司

2022 年 1 月