# PYTHON调试

张龙

# 前言

## 课程目标

1. 熟悉可能需要调试的场景或问题
   - Python脚本语言与调试相关的基本知识

2. 掌握基本的调试工具和方法
   - 土法调试
   - Pdb模块
   - pyDev+Eclipse

3. 了解高级的调试工具和方法
   - 调试内存泄露
   - 调试段错误
   - 调试死锁等
   - 调试多线程等

# Agenda

1. 介绍
2. 基础调试
3. 进阶调试

# 介绍

下面的情况需要调试吗？

1. 语法错误
2. 编码问题
3. 性能问题
4. 内存问题

# 介绍

代码调试是**论如何排查问题**的一个过程，根据一切蛛丝马迹，推断出问题所在，并消灭之

# Agenda

1. 介绍
2. **基础调试**
3. 进阶调试

# 基础调试

## 土法调试

1. Print
   - *dir, type, __file__，。。。*
2. Log
   - 4W， 15-05-25 19:50:10 INFO {9817--1218664768 processor.create_spider_task_msg:485} create a spider task
   - *DEBUG、INFO、WARNING、ERROR、CRITICAL*
3. Assert
   - *开关__debug__*
   - *关键位置/状态/条件*
4. 二分法/替代法
5. Google

<demo>

# 基础调试

PDB的两种使用方式

## 1. 修改源码

```
import pdb
pdb.set_trace()
```

## 2. 不改源码

```
Python –m pdb xxx.py
```

# 基础调试

## PDB的能与不能

√ 在调试的时候可以动态改变变量的值

× 不支持多线程和远程调试

× 不支持段错误调试

× 不支持运行中的daemon程序调试

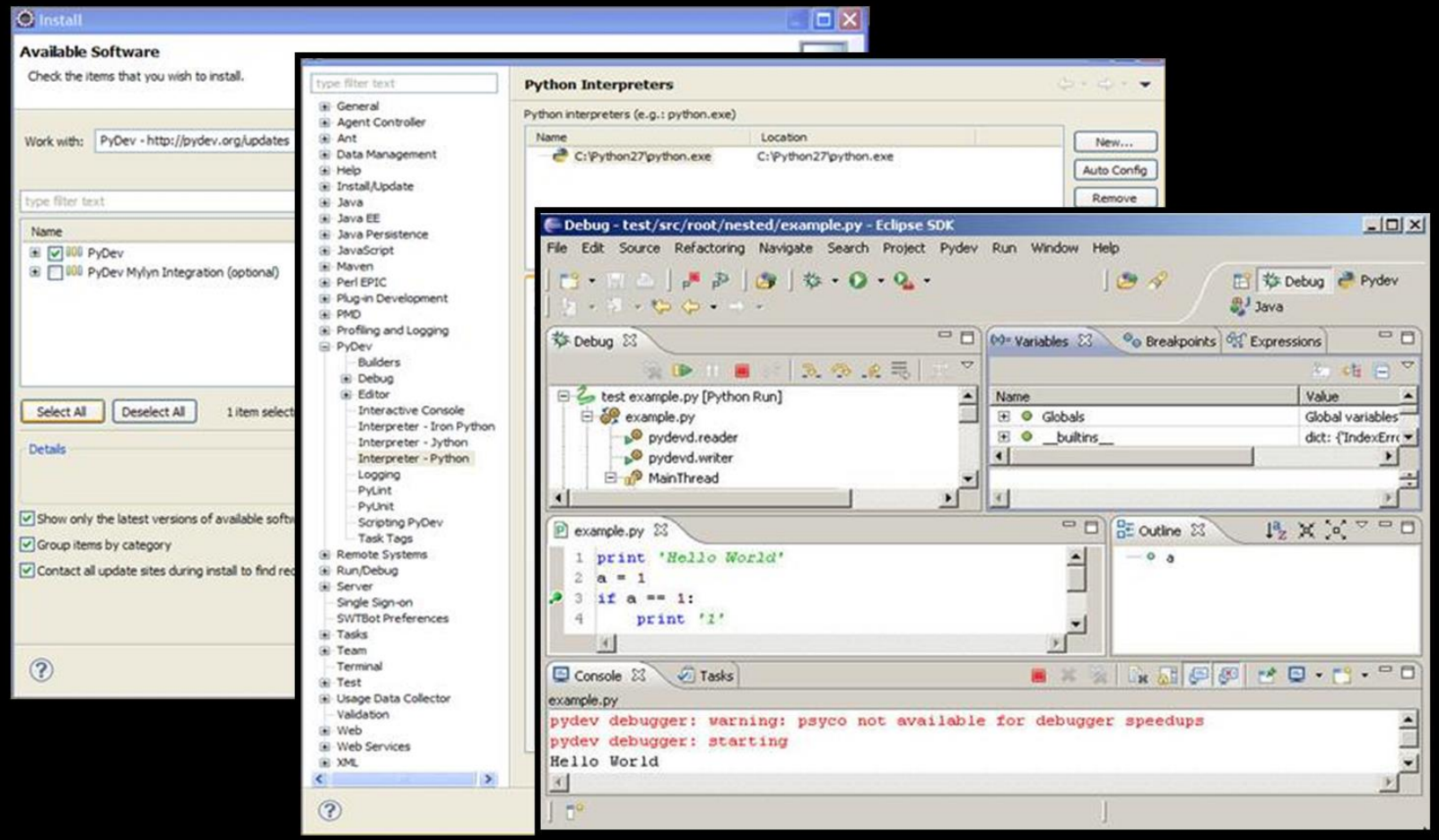
| 命令 | 解释 |
| --- | --- |
| break 或 b 设置断点 | 设置断点 |
| continue 或 c | 继续执行程序 |
| list 或 l | 查看当前行的代码段 |
| step 或 s | 进入函数 |
| return 或 r | 执行代码直到从当前函数返回 |
| exit 或 q | 中止并退出 |
| next 或 n | 执行下一行 |
| pp | 打印变量的值 |
| help | 帮助 |

<demo>

# 基础调试

## Pydev+Eclipse

- 安装、配置(略)




<demo>

# Agenda

1. 介绍
2. 基础调试
3. **进阶调试**

# 进阶调试

## 内存问题

1.GC

- **引用计数不为零**
  - *交叉引用*
  - *全局变量和类变量*

- **gc.collect()**
  - *分代回收* gc.set_threshold(threshold0[, threshold1[, threshold2]])
  - *频繁做垃圾回收可能会影响性能*

<demo>

2. 底层的C/C++模块引起的内存泄露

# 进阶调试

## 内存问题调试工具

1. **定性**
   - sys.getrefcount — 查看当前对象的引用数
   - gc.set_debug(gc.DEBUG_LEAK) — 进行循环引用导致的内存泄露的检查 <demo>
2. **可视化**

   **Objgraph**（需要安装graphviz）
   - objgraph.show_most_common_types(limit=50) — 程序内存的前 N 个对象
   - objgraph.show_growth() — 哪些对象被增加了 <demo>
   - objgraph.show_backref() — 泄漏对象的引用是什么
3. **定量**

   **memory profiler**
   - python -m memory_profiler xxx.py <demo>

# 进阶调试

## 多线程

1. pydev+eclipse

   <demo>

2. gdb+pstree

   <demo>

```
(gdb) info threads
  Id   Target Id         Frame
  5    Thread 0xb6740b70 (LWP 10585) "python" 0xb77d0424 in __kernel_vsyscall ()
  4    Thread 0xb5f3fb70 (LWP 10586) "python" 0xb77d0424 in __kernel_vsyscall ()
  3    Thread 0xb573eb70 (LWP 10587) "python" 0xb77d0424 in __kernel_vsyscall ()
  2    Thread 0xb4f3db70 (LWP 10588) "python" 0xb77d0424 in __kernel_vsyscall ()
* 1    Thread 0xb76136c0 (LWP 10564) "python" 0xb77d0424 in __kernel_vsyscall ()
(gdb) thread 2
[Switching to thread 2 (Thread 0xb4f3db70 (LWP 10588))]
#0  0xb77d0424 in __kernel_vsyscall ()
(gdb)
```

# 进阶调试

## 段错误

Python代码一般不会出现段错误，往往是底层的C模块引起的。

- dmesg
- ulimit -c

  generate-core-file生成coredump文件用gdb –core来进行debug

<demo>

- gdb+libpython.py
  - GNU的gdb

```
(gdb) python
> import sys
>sys.path.insert(0, '/path/to/libpython.py' )
> import libpython
>end
(gdb)
```


  - gdb有—python选项（redhat或fedora修改过的）*$ gdb --python  /path/to/libpython.py -p $pid*
  - 其它安装python2.7-dbg https://wiki.python.org/moin/DebuggingWithGdb

# 进阶调试

## 死锁

1. strace

   *$ strace -p $pid*

2. gdb

   *$ gdb -p 16188*

   *(gdb) info threads*

   *Id Target Id Frame*

   *2 Thread 0xb7300b70 (LWP 16189) "python" 0xb7723424 in __kernel_vsyscall ()*

   *1 Thread 0xb75666c0 (LWP 16188) "python" 0xb7723424 in __kernel_vsyscall ()*

<demo>

$ gdb -p 16190

(gdb) py-bt

#5 Frame 0x96a136c, for file /home/zhanglong/ref_dev/tool/autobuild/wse/env/python/lib/python2.7/threading.py, line 127, in acquire (self=<_RLock(_Verbose__verbose=False, **_RLock__owner=-1221588112**, _RLock__block=<thread.lock at remote 0xb75570a0>, _RLock__count=1) at remote 0x96097cc>, blocking=1, me=-1219074368)

rc = self.__block.acquire(blocking)

# Q&A