1 Sử dụng debugger jdb và pdb cho Java và Python

Các ngôn ngữ lập trình thường được cung cấp các bộ công cụ hỗ trợ debug. Các chức năng thường thấy trong các bộ công cụ này bao gồm:

- Breakpoint (điểm dừng): Cho phép lập trình viên dừng chương trình tại điểm xác định và theo dõi các giá trị tại điểm dừng hiện thời, từ đó suy luận ra lỗi. Điểm dừng có thể có hoặc không đi kèm điều kiện kích hoạt (conditional breakpoint).
- Stepping (thực hiện từng bước): Cho phép lập trình viên chạy chương trình theo từng bước một thay vì liên tục, cũng như theo dõi các giá trị ở từng bước.
- Stack Trace (truy vết ngăn xếp): Cho phép lập trình viên theo dõi call stack (ngăn xếp lệnh) để biết được lệnh nào đang gọi lệnh nào vào một thời điểm xác định.

Hai ngôn ngữ Java và Python được cung cấp sẵn hai bộ công cụ debug mặc định là jdb và pdb.

1.1 jdb - Java Debugger

jdb là một bộ công cụ debug sử dụng command line dành cho Java. Nó hỗ trợ khả năng debug code Java chạy trên máy ảo nội bộ hoặc từ xa.

1.1.1 Khởi động một tiến trình jdb

Cú pháp chung để khởi động jdb trên giao diện command line là

```
jdb [ options ] [ class ] [ arguments ]
```

Trong đó:

- options: các cài đặt cho tiến trình hiện tại.
- class: class để thực hiện debug.
- arguments: các tham số truyền vào chương trình, giống như khi khởi tạo máy ảo Java bình thường.

Có hai cách thông dụng để tạo tiến trình jdb. Cách thứ nhất là yêu cầu jdb khởi tạo một máy ảo mới để chạy class cần được debug. Ví dụ:

```
jdb MyClass
```

Khi được khởi tạo theo cách này, máy ảo Java sẽ tạm dừng trước khi bắt đầu thực hiện chỉ thị đầu tiên của chương trình.

Cách thứ hai là kết nối jdb với một máy ảo đang chạy. Máy ảo đang chạy cần phải sử dụng cài đặt:

```
-agentlib:jdwp=transport=dt_shmem,server=y,suspend=n
```

để chuẩn bị các thư viện debug và chỉ thị cụ thể cách thức kết nối cho jdb. Ví dụ: để khởi chạy class MyClass cho việc debug:

```
java -agentlib:jdwp=transport=dt_shmem,
    address=jdbconn,server=y,suspend=n MyClass
```

Khi đó ta có thể kết nối với jdb bằng lệnh:

```
jdb -attach jdbconn
```

Chú ý rằng class MyClass không được truyền vào jdb, vì ta đang không khởi tạo máy ảo mới.

1.1.2 Các lệnh jdb cơ bản

Nếu ta bắt đầu tiến trình jdb bằng cách khởi tạo máy ảo mới, ta có thể thiết lập các breakpoint cần thiết trước khi bắt đầu chạy chương trình bằng lệnh run.

Để sử dụng breakpoint, ta có một số cú pháp:

- stop at MyClass:22 (Đặt breakpoint ở dòng thứ 22 của file MyClass).
- stop in java.lang.String.length (Đặt breakpoint ở đầu của hàm java.lang.String.length).
- stop in MyClass.<init> (Đặt breakpoint tại đầu hàm khởi tạo của class MyClass).
- stop in MyClass.<clinit> (Đặt breakpoint tại đầu hàm khởi tạo static của class MyClass).

Nếu như một hàm được overload (viết đè), ta phải viết rõ kiểu dữ liệu của các biến truyền vào của overload mục tiêu. Ví dụ:

```
stop in MyClass.myMethod(int,java.lang.String)
```

Lệnh clear cho phép ta xóa bỏ một breakpoint cụ thể (khi truyền vào tham số), hoặc xóa tất cả breakpoint (khi không truyền vào tham số).

Khi gặp breakpoint, lệnh cont yêu cầu chương trình tiếp tục chạy. jdb không hỗ trợ conditional breakpoint.

jdb hỗ trợ stepping thông qua hai lệnh step (chạy tiếp dòng lệnh tiếp theo trong chương trình) và next (chạy tới hết hàm hiện tại trong call stack, nhảy sang hàm tiếp theo).

Để theo dõi các giá trị trong chương trình, jdb cung cấp hai lệnh print và dump. Ngoài việc in ra các biến có trong chương trình, lập trình viên có thể kiểm tra giá trị tính toán bất kì giữa chúng.

Để theo dõi các thread đang chạy, jdb cung cấp lệnh threads để liệt kê danh sách, và thread để đổi thread đang thao tác lên.

Để theo dõi call stack, jdb cung cấp lệnh where.

1.2 pdb - Python Debugger

pdb là một bộ công cụ debug trên nền command line dành cho Python. pdb có thể được chạy trên nền command line giống như jdb, hoặc import trực tiếp vào mã nguồn Python để sử dụng và mở rộng.

1.2.1 Khởi động một tiến trình pdb

Có một số cách để sử dụng pdb:

1) Thông qua Python shell:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
```

2) Bằng cách chạy script pdb.py:

```
python3 -m pdb myscript.py
```

3) Bằng cách kêu gọi từ trong mã nguồn cần debug:

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

Hoặc ở trong Python phiên bản 3.7 trở lên:

```
breakpoint()
```

1.2.2 Các lệnh pdb cơ bản

Khi sử dụng pdb dưới dạng command line (cách 1 hoặc cách 2), tương tự như với jdb, chương trình sẽ được tạm dừng trước khi thực hiện chỉ thị đầu tiên, và ta có thể sắp xếp các breakpoint cần thiết trước khi bắt đầu chạy bằng lệnh run.

Cú pháp để thiết lập breakpoint trong pdb là:

```
b(reak) [([filename:]lineno | function) [, condition]]
```

Nếu ta sử dụng tham số lineno, breakpoint được đặt tại dòng được chỉ định. Nếu ta sử dụng tham số function, breakpoint được đặt trước chỉ thị đầu tiên của hàm tương ứng. Filename có thể được sử dụng để chỉ thị rõ ràng tên file đặt breakpoint.

Nếu condition được truyền vào, breakpoint sẽ chỉ được kích hoạt nếu condition trả về true.

Nếu như không có tham số nào được truyền vào, lệnh này sẽ liệt kê tất cả các breakpoint, cùng với số lần breakpoint này đã được kích hoạt, số lần bỏ qua và điều kiện đi kèm (nếu có).

pdb còn hỗ trợ một loại breakpoint đặc biệt là temporary breakpoint (breakpoint tạm thời). Temporary breakpoint sẽ bị loại bỏ ngay vào lần đầu tiên nó được kích hoạt. Cú pháp của temporary breakpoint là:

```
tbreak [([filename:]lineno | function) [, condition]]
```

Các breakpoint có thể bị vô hiệu hóa bằng lệnh disable, hoạt hiệu hóa trở lại bằng lệnh enable, hoặc bị loại bỏ hoàn toàn bằng lệnh clear. Chúng cũng có thể được chỉ thị để bỏ qua một số lần nhất định bằng lệnh ignore, hoặc thay đổi điều kiện kích hoạt bằng lệnh condition.

Tương tự như jdb, khi gặp một breakpoint, pdb sẽ dừng chương trình lại cho tới khi gặp lệnh continue.

Cũng tương tự như jdb, pdb hỗ trợ các lệnh step, next và where để thực hiện chương trình theo từng bước một và để theo dõi call stack.

Để in ra một giá trị trong quá trình debug, ta sử dụng cú pháp:

```
p expression
```

expression ở đây có thể là một giá trị, một biến, một biểu thức hoặc một chương trình con.