ROVM Client 문서

정 원교

2006.3.20

지구 남쪽에서

이메일 : weongyo@gmail.org

요 약

이 문서는 ROVM Client 를 이용하여 ROVM Server 와 어떻게 연동이 되는지를 설명한 문서입니다. ROVM Client 를 이용하여 사용할 수 있는 명령어들과 각각의 기능에 대해서 설명하였습니다.

차 례

제 1 절 소개

ROVM Client 는 ROVM 의 한 구성요소로써 순수히 ROVM Server 와의 통신을 목적으로 만들어졌으며 또한 명령어의 디버그 과정과 Opcode 의 디버그 과정을 보다 쉽게 하기 만들어졌습니다.

ROVM Client 는 사용자가 입력한 명령어를 해석하여 ROVM Server 가 인식할 수 있는 네트워크 바이트를 구성하여 ROVM Server 에 전달하는 것이 목적입니다. 또한 사용자가 요청한 request 에 대해 서버 측이 응답한 결과를 해석하여 사용자가 쉽게 이해할 수 있도록 화면에 뿌려주는 것이 목적입니다.

ROVM Client 는 현재 Python 코드로 작성되어 있습니다. 실제 실행 과정 또한 약간 python interpreter 과 비슷하게 구성되어 있습니다.

제 2 절 실행하기

ROVM Client 는 현재 하나의 Python 코드로 구성되어 있으며 실행시 Shell 과 같이 프롬프트가 뜨게 됩니다.

```
C:\rovmclient> python rovmclient.py
ROVM Client 0.0.1a
Type "help" for more information.
>>>
```

내부에서 사용되는 명령어에 대해 자세히 살펴보기 위해서 'help' 명령어를 입력함으로써 사용법에 대한 설명을 보실 수 있습니다.

만약 각 명령어 대한 자세한 설명을 원한다면 'help <command>' 를 입력함으로써 세부적으로 살펴볼 수 있습니다.

```
>>> help

Normal mode commands
------------------------------------------------------------------
 help         Show this message.
 opcode       Entering OPCODE mode.
 printopcode  Print saved opcodes.
 quit         Exit rovmclient.
 req <URL>    Get Ticket.
 reqend       Remove Ticket.
 send         Send opcodes.
 stack        Print remote stack status.
------------------------------------------------------------------
If you want to see a detailed explanation, Type 'help <command>'.

>>> help req

REQ - 티켓을 끊습니다.

ROVM 에서는 어떤 명령어를 원격에서 실행시키기 위해서는 해당 서버의 허락을
받아야 합니다.  그러한 허락을 받을 수 있는 방법이 바로 서버에서 티켓을
끊는 것입니다.  이 명령어를 통해서 서버로 부터 Unique 티켓을 획득할 수
있으며 다음 명령어 부터는 이 티켓을 사용해서 명령어를 실행할 수 있습니다.
즉, 실행하고자 하는 모든 명령어에는 이 티켓이 함께 동봉됩니다.  이를 통해
서버에서는 해당 티켓의 유효성을 테스트하게 됩니다.

형식 : req <URL 주소>
예제 :
        >>> req e://192.168.59.128:8889

>>>
```

ROVM Client 의 프롬프트는 Python 과 같은 '>>>' 입니다. 이 모습과는 다른 '...' 프롬프트도 존재하는데 이에 대한 자세한 설명은 3.2 절에서 자세하게 하게 될 것입니다.

제 3 절 실행 모드

ROVM Client 의 경우, 현재 두가지 모드를 가지고 있으며, 각 모드마다 그 용도가 다릅니다. 일반적으로 ROVM Client 를 실행하였을 때, 기본적으로 화면에 나타나게 되는 것은 '명령어 모드'이고 'opcode' 명령을 통해서 진입할 수 있는 'Opcode 모드'라는 것이 또한 존재하게 됩니다.

'명령어 모드'는 ROVM 의 실제 operation (여기서 operation 이란 오직 Virtual Machine 상에서의 작동을 뜻합니다.)

을 제외한 부분을 처리하기 위한 모드입니다. 즉, 티켓을 끊는다거나, opcode 를 서버에 전송한다거나 Server 의 상태 확인한다거나 그런 종류의 명령들은 모두 '명령어 모드'에 해당하게 됩니다.

'Opcode 모드'는 실제 Virtual Machine 상에서의 operation 과 관련된 명령어를 입력할 때 사용합니다. 예를 들면, 실행할 opcode 를 입력하는 과정이 그 예가 되겠습니다. 자세한 내용은 아래에서 할 것입니다.

3.1 명령어 모드

3.1.1 HELP

도움말을 출력하는 명령어입니다.

형식 : help
예제 :

```
>>> help

Normal mode commands
----------------------------------------------------------------
 help         Show this message.
 opcode       Entering OPCODE mode.
 printopcode  Print saved opcodes.
 quit         Exit rovmclient.
 req <URL>    Get Ticket.
 reqend       Remove Ticket.
 send         Send opcodes.
 stack        Print remote stack status.
----------------------------------------------------------------
If you want to see a detailed explanation, Type 'help <command>'.

>>>
```

3.1.2 OPCODE

Opcode 모드로 진입합니다.

ROVM Client 는 현재 두가지 모드를 가지고 있습니다. 첫번째는 '>>>' 프롬프트로 시작하는 '일반 모드'와 '. . .' 프롬프트로 시작하느 'Opcode 모드'가 그것입니다. 일반 모드에서는 티켓을 끊거나 서버의 스택을 확인하거나 등등의 여러 실행, 디버깅, 기타 일들을 위한 명령어를 제공합니다. 하지만 'Opcode 모드' 에서는 실제 ROVM 에서 실행되는 opcode 들을 중심으로 구성되어 있습니다. Opcode 모드를 빠져나오기 위해서는 '. . .' 프롬프트 상태에서 단순히 <ENTER> 키만 눌러주면 자동으로 빠져나오게 됩니다. 사용자가 입력한 opcode 들이 자동으로 해석되어 저장되어 있다가 '일반 모드' 의 send 명령어가 실행되었을 때 Server 측에 보내게 됩니다. 즉, opcode 가 완성된

형태일 필요는 전혀 없습니다. 디버깅을 위해서 하나의 opcode 만 넣고 send 명령어를 실행시켜도 됩니다. '일반 모드'의 stack 명령어를 통해서 *항상* 티켓에 할당된 스택 상태를 확인할 수 있습니다.

형식 : opcode
예제 :

```
>>> opcode
... <현재 opcode 모드 상태에 진입해 있습니다.>
```

3.1.3 PRINTOPCODE

사용자가 현재까지 입력한 opcode 목록을 보여줍니다.

사용자가 'Opcode 모드' 에서 입력한 opcode 가 실제로 어떻게 변환되었는지를 확인할 수 있으며, 자신이 입력한 opcode 가 어떤 것들을 입력하였는지를 보여주는 명령어입니다.

형식 : printopcode
예제 :

```
>>> opcode
... cpush 1
... cpush 2
...
>>> printopcode
<입력한 내용이 출력됨>
```

3.1.4 QUIT

ROVM Client 프로그램을 종료할 때 사용합니다.

3.1.5 REQ

티켓을 끊습니다.

ROVM 에서는 어떤 명령어를 원격에서 실행시키기 위해서는 해당 서버의 허락을 받아야 합니다. 그러한 허락을 받을 수 있는 방법이 바로 서버에서 티켓을 끊는 것입니다. 이 명령어를 통해서 서버로 부터 Unique 티켓을 획득할 수 있으며 다음 명령어 부터는 이 티켓을 사용해서 명령어를 실행할 수 있습니다. 즉, 실행하고자 하는 모든 명령어에는 이 티켓이 함께 동봉됩니다. 이를 통해 서버에서는 해당 티켓의 유효성을 테스트하게 됩니다.

'URL 주소'의 경우, e://<hostname>:<port> 형식을 반드시 지켜야 합니다. '<hostname>' 의 경우 *192.168.23.234* 혹은 *www.blahblah.org* 형식 또한 상관없습니다. '<port>' 번호의 경우 해당 서버가 현재 열고 있는 포트 번호를 입력해 주면 됩니다.

만약 "인증"을 통한 ROVM Server 에 접근하고자 한다면, e://<userid>@<hostname>:<port> 형식을 사용하시기 바랍니다.

형식 : req <URL 주소>
예제 :

```
>>> req e://192.168.59.128:8889
>>> req e://rovm@192.168.58.129:4390
```

만약 "인증" 방식을 사용한다면, 사용자의 패스워드를 물어 올 것입니다.

```
C:\>python rovmclient.py
ROVM Client 0.0.1h
Type "help" for more information.
>>> req e://rovm@192.168.58.129:4390
Enter password:
```

3.1.6 REQEND

티켓을 버립니다.

req 명령어를 통해서 구입한 티켓을 더 이상 사용할 필요가 없어서 이를 버립니다. 이 명령어를 실행할 경우, ROVM 서버 상에서의 티켓과 관련된 모든 resource 들을 해제하게 되며, 더 이상 해당 티켓을 사용할 수 없게 됩니다.

형식 : reqend
예제 :

```
>>> req e://192.168.59.128:8889
>>> reqend
```

3.1.7 SEND

Opcode 를 보냅니다.

사용자가 'Opcode 모드'에서 입력해 둔 명령어들을 실제 서버측에 보냅니다. 이 명령어가 실행되기 전에 반드시 서버로부터 티켓을 발급받아야 합니다. 이 명령어가 실행된 후, 사용자가 입력한 opcode 들은 지워지게 되며, 다시 'Opcode 모드'로 명령어를 입력하고, send 명령어를 실행했을 때는, 이제 막 'Opcode 모드'에서 입력했던 opcode 들만 전송되게 됩니다.

형식 : send
예제 :

```
>>> req e://192.168.59.128:8889
>>> opcode
... <실행할 opcode 를 입력합니다.>
... <ENTER>
>>> send
```

3.1.8 STACK

스택 상태를 출력합니다.

현재 티켓에 설정되어 있는 스택 상태를 원격 서버로 부터 받아와 화면에 출력합니다.

이 명령어는 ROVM Server 가 디버깅 상태로 동작할 경우, 가능한 명령어로써 기본적으로 disable 되어 있습니다. Disable 된 Server 이 이 명령을 보낸다면 **ERROR** 가 반환되어 올 것입니다.

형식 : stack
예제 :

```
>>> req e://192.168.59.128:8889
>>> opcode
... <실행할 opcode 를 입력합니다.>
... <ENTER>
>>> send
>>> stack
<스택 내용이 출력된 화면>
```

3.2 Opcode 모드

현재 'Opcode 모드'는 계속 개발 중이며, 앞으로 완성시켜야 할 Opcode 갯수가 매우 많이 남아 있습니다. 그에 따라 이 'Opcode 모드' 절 또한 계속 업데이트 될 것입니다.

'명령어 모드'에서 'Opcode 모드'로 진입하기 위해서 단순히 'opcode' 명령어를 입력함으로써 들어갈 수 있습니다.

```
>>> opcode
...
```

'...' 프롬프트는 현재 ROVM Client 프로그램이 'Opcode 모드'에 들어와 있음을 가르킵니다. 만약 'Opcode 모드'에서 '명령어 모드'로 빠져나가기 위해서는 단순히 <ENTER> 키만 입력해 주면 됩니다.

```
... <ENTER>
>>>
```

'Opcode 모드' 상에서도 '도움말 기능'을 볼 수가 있는데, 'help' 명령어를 입력함으로써 'Opcode 모드'상에서 사용할 수 있는 명령어들을 열람할 수 있습니다.

```
... help

Opcode mode commands
----------------------------------------------------------------
 cpush <Char Number>
              Push 'char type' number.
 iadd         Add int.
 <ENTER>      Escape opcode mode
----------------------------------------------------------------
If you want to see a detailed explanation, Type 'help <command>'.
```

3.2.1 BPUSH

Boolean 을 PUSH 합니다.

Boolean 을 스택상에 PUSH 합니다. 오직 값은 1 혹은 0 만 존재하며, 1 일 경우 TRUE 를, 0 일 경우 FALSE 를 의미합니다.

번호 : 13 (0xd)
형식 : bpush `<1 byte unsigned char>`
스택 : (...) → (..., *<boolean>*)
예제 :

```
... bpush 0
... bpush 1
```

3.2.2 CPUSH

Char (C 언어에서 char 타입) 를 PUSH 합니다.

Char 을 PUSH 합니다. Char 은 1 바이트의 type 으로써 C 언어의 'char' 와 완벽하게 같습니다. 숫자의 범위 -128 127 까지 입니다. 더하기 등등의 연산시 이 값은 integer 로써 계산이 되게 됩니다.

번호 : 16 (0x10)
형식 : cpush <1 byte number>
스택 : (...) → (..., *<byte number>*)
예제 :

```
... cpush -128
... cpush 127
```

3.2.3 HPUSH

Short (C 언어에서 short 타입) 을 PUSH 합니다.

Short 을 PUSH 합니다. C 언어와 마찬가지로 2 바이트의 type 입니다.

번호 : 17 (0x11)
형식 : hpush <2 byte short>
스택 : (...) → (..., <short>)
예제 :

```
... hpush 12332
... hpush -123
```

3.2.4 IPUSH

Integer (C 언어에서 int 타입) 를 PUSH 합니다.

Integer 을 PUSH 합니다. Integer 은 4 바이트의 type 으로써 C 언어의 ‘int 와 완벽하게 같습니다.

번호 : 18 (0x12)
형식 : ipush <4 byte signed number>
스택 : (...) → (..., *<int number>*)
예제 :

```
... ipush -2147483648
... ipush 2147483647
```

3.2.5 FPUSH

Float 를 스택에 PUSH 합니다.

Float 타입으로써 4 바이트 실수를 표현합니다.

번호 : 14 (0xe)
형식 : fpush `<4 byte float>`
스택 : (...) → (..., `<float>`)
예제 :

```
... fpush 123.0
```

3.2.6 DPUSH

Double 를 스택에 PUSH 합니다.

Double 타입으로써 8 바이트 실수를 표현합니다.

번호 : 15 (0xf)
형식 : dpush `<8 byte double>`
스택 : (...) → (..., `<double>`)
예제 :

```
... dpush 123123.0
```

3.2.7 SPUSH

문자열을 stack 상에 PUSH 합니다.

PUSH 된 문자열은 자동으로 StringRef (ObjectRef 의 다른 이름으로써, 문자열과 관련된 점만 강조하기 위해서 지은 이름입니다.) 로 변환되게 되어 저장되게 됩니다. 가변길이를 가지는 opcode 입니다.

번호 : 19 (0x13)
형식 : spush “`<string>`”
스택 : (...) → (..., `<StringRef>`)
예제 :

```
... spush "Hello! ROVM Server"
... spush "I love you."
```

3.2.8 DUP

가장 상위 스택을 복제하여 PUSH 합니다.

스택에 올라와 있는 리스트 중 가장 상위에 있는 스택을 그대로 복사하여 스택에 PUSH 합니다.

번호 : 89 (0x59) 형식 : dup 스택 : (..., <value>) → (..., <value>, <value>) 예제 :

```
... spush "Hello! ROVM Server"
... dup
```

3.2.9 IADD

Integer 를 더합니다.

스택에서 2 개의 integer entry 를 pop 한 후에, 더하고 그에 대한 결과를 다시 push 합니다.

번호 : 90 (0x60)
형식 : iadd
스택 : (..., *<value1>*, *<value2>*) → (..., *<result>*)
예제 :

```
... cpush 1
... cpush 2
... iadd
```

3.2.10 CALL

Class 내부에 선언되어 있는 메쏘드를 호출하는데, 사용됩니다. 이 opcode 가 사용되기 전에 반드시 스택상에 objectref 가 존재해야 하며, 메쏘드 type 에 맞는 argument 또한 먼저 push 되어 있어야 합니다.

번호 : 182 (0xb6)
형식 : call <name> <type>
스택 : (..., *objectref*, [*arg1*, [*arg2* ...]]) -¿ (...)
예제 :

```
... call __init__ (S)V
... call abc (SII)I
```

3.2.11 NEW

지정한 Class 에 대한 Object Reference 를 PUSH 합니다.

지정한 Class 에 대한 Object Reference 를 구해서 Stack 에 PUSH 하게 됩니다. 만약 해당 Class 를 찾지 못하였거나, 성공적으로 Load 하지 못하였거나, 해당 Object Reference 를 생성하지 못하였다면, 'ERROR' 메세지를 반환하게 됩니다.

번호 : 187 (0xbb)
형식 : new `e://<hostname>[:<port>]<path>`
스택 : (...) → (..., *<objectref>*)
예제 :

```
... new e://192.168.58.129:4390/testsuite/insert
... new e://envlang.kldp.net/testsuite/insert
```

3.2.12 NEWARRAY

지정한 Type 에 대한 Array Ref 를 생성합니다.

n 개의 배열을 만들기 위해서는 이 명령어가 실행되기 전에 해당 정보를 스택에 PUSH 해 놓아야 합니다. 해당 정보를 바탕으로 배열을 생성하여 그에 맞는 ArrayRef 를 반환하게 됩니다. `<type number>` 에 대한 자세한 정보는 "ROVM 문서"에서의 'NEWARRAY' opcode 를 참조하시기 바랍니다.

번호 : 188 (0xbc)
형식 : newarray `<type number>`
스택 : (..., *<count>*) → (..., *<arrayref>*)
예제 :

```
... ipush 256
... newarray 3
```