



02. 5주차. 9장 GDK 사용하기

Table of Contents

- 02. 5주차. 9장 GDK 사용하기
 - 객체 정보 문자열로 얻기
 - 프로퍼티에 접근하기
 - 동적으로 메서드 호출하기
 - Object의 편리한 메서드들
 - object 편리한 메서드
 - Object되풀이 메서드
 - File의 객체 되풀이 메서드
 - GDK의 파일 및 입출력 메서드
 - 파일시스템 탐색하기
 - 파일읽기
 - 파일쓰기
 - 필터와변환기
 - 필터와 변환기 사용
 - base64 Encode,Decode 하는것을 보자.유용하다.
 - 직렬화한 객체 전송하기
 - 객체 직렬화 저장, 불러오기
 - 그루비 쓰레드
 - 간단한 쓰레드 실행하는 문법을 살펴보자
 - 쓰레드 예제 한번 분석해보자
 - sleep값을 바꿔가면서 반응을 확인해보자
 - 외부프로세스와 연결하기
 - 외부프로세스 사용해보자.
 - 프로세서와 대화하기.

그루비 아키텍처에서 GDK위치

API: 빌더, 템플릿, SQL...	그루비 라이브러리
GDK	
API: 입출력, 코어, 유틸	JRE 실행환경
자바 가상머신	

그루비는 사용하기 편하도록 JRE를 확장하고(수정도함) 새로운 동적 특징을 추가하고 언어의 표현 방식에 맞게 API도 변형했다. 이러한 확장이나 변형을 통틀어 GDK라고한다

그루비는 컬렉션 관련 메서드들을 Object에 편의 메서드들을 몇 개 추가했다(어드든 쓸수있도록)

객체 정보 문자열로 얻기

자바에서는 toString으로 객체의 정보를 문자열로 받는다. 그루비는 여기에 2개의 또다른 메서드가있다.

1. dump : 객체의 상태, 즉 필드들의 이름과 값을 리턴한다

여기서 중요한거는dump의 결과는 JVM에 따라 달라지기도 한다

2. inspect : 최대한 소스코드와 비슷한 형태의 문자열로 리턴 (표현하기 어려울땐 toString값을 리턴)

```

1 package _C1
2
3 newline = "n"
4
5 println 'newline.toString(); : ' + newline.toString();
6 println 'newline.dump(); : ' + newline.dump(); //객체의 상태 즉 필드들의 이름과 값을 리턴한다.
7 println 'newline.inspect(); : ' + newline.inspect(); //최대한 소스코드와비슷한 형태로 리턴
8
9 println ";
10 class C1_1{
11     def def_val=1;
12     String string_val=1;
13     public public_val=1;
14     private private_val=1;
15 }
16 C1 1 c = new C1 1():

```



```

17 println C1_1.dump();           //객체의 상태 즉 필드들의 이름과 값을 리턴한다.
18 println C1_1.inspect();
19 println C1_1.toString();
20 println "
21 println c.dump();             //객체의 상태 즉 필드들의 이름과 값을 리턴한다.
22 println c.inspect();
23 println c.toString();
24
25 //프로퍼티 접근////////
26 println "
27 println c.properties;
28
29
30 /*결과
31 newline.toString(); :
32
33 newline.dump();           : <java.lang.String@a value=
34   offset=0 count=1 hash=10>
35 newline.inspect();       : "n"
36
37 <java.lang.Class@ae533a cachedConstructor=null newInstanceCallerCache=null name=_C1.C1_1 declaredFields=java.lang.ref.SoftRefer
38 class _C1.C1_1
39 class _C1.C1_1
40
41 <_C1.C1_1@7eb366 def_val=1 string_val=1 public_val=1 private_val=1>
42 _C1.C1_1@7eb366
43 _C1.C1_1@7eb366
44
45 [class: class _C1.C1_1, string_val:1, def_val:1, metaClass: org.codehaus.groovy.runtime.HandleMetaClass@941db6[groovy.lang.MetaClass]
46
47 */

```

프로퍼티에 접근하기

getProperties나 properties를 통해서 프로퍼티를 들여다볼수있다. 읽기전용 맵으로 만들어서 리턴한다.
 ['name'] 처럼 배열기호 연산자 , getAt 함수를 쓰면 getAt 함수를 호출한다.

```

1 package _C2
2 class MyClass {
3     def first = 1           // read-write property
4     def getSecond() { first * 2 } // read-only property
5     public third = 3       // public field property
6     private private_val='private_val';
7     protected protected_val='protected_val';
8     final protected final_protected_val='final_protected_val';
9     static String static_string_val='static_string_val';
10
11     public def getFirst() {
12         println '    call getFirst';
13         return first;
14     }
15     public def getAt(String name){ //getAt을 오버라이딩하면 배열접근,getAt을 호출하면 이것을 타고 이메서드가 없을시 프로퍼티 접근
16         println '    call getAt name:' + name
17         return this."$name"
18     }
19     // return this."$name"
20 }
21
22     public void setFirst(def value){
23         println '    call setFirst value:' + value;
24         this.@first = value;
25     }
26     public void putAt(String name,def value){ //putAt 오버라이딩하면 배열접근,putAt 호출하면 이것을 타고 이메서드가 없을시 프로퍼티 접근
27         println '    call putAt name:' + name + ' value:' + value
28         this."$name" = value;
29     }
30 }
31
32 obj = new MyClass()
33
34
35 println 'obj.properties : ' + obj.properties
36 println 'obj.properties['first'] : ' + obj.properties['first']
37 println 'obj.properties.first : ' + obj.properties.first
38 println 'obj.first : ' + obj.first
39 println 'obj['first'] : ' + obj['first']
40 println 'obj.getAt('first') : ' + obj.getAt('first')
41
42 println '-----';
43 one = 'first'
44 two = 'second'
45 obj[one] = obj[two] // putAt(one) // #3
46 obj.putAt(one,'first2');
47 obj[one] = 'second--';
48
49 println '-----';
50 println obj.dump(); //필드 내용중 static 안보인다.
51 //assert obj.dump() =~ 'first=2' // #4
52
53 /*결과
54 call getFirst
55 obj.properties : [class: class _C2.MyClass, first:1, second:2, static_string_val:static_string_val, metaClass: org.codehaus.groovy.runtime.HandleMetaClass@941db6[groovy.lang.MetaClass]
56 call getFirst
57 obj.properties['first'] : 1
58 call getFirst
59 obj.properties.first : 1

```

```

59     call getFirest
60 obj.first : 1
61     call getAt name:first
62     call getFirest
63 obj['first'] : 1
64     call getAt name:first
65     call getFirest
66 obj.getAt('first') : 1
67 -----
68     call getAt name:second
69     call putAt name:first value:2
70     call putAt name:first value:first2
71     call putAt name:first value:second--
72 -----
73 <_C2.MyClass@149eb9f first=second-- third=3 private_val=private_val protected_val=protected_val final_protected_val=final_protected_val
74 _C2.MyClass@149eb9f
75
76
77 */

```

동적으로 메서드 호출하기

- 어떤 객체의 메서드나 필드에 대한 정보가 필요하면 다음GPath사용한다
obj.class.methods.name
obj.class.fields.name
- 그리고 GDK에서 동적으로 추가된 메서드들은 MetaClass에서 ㅈ정보를 제공한다
obj.metaClass.metaMethods.name
중복없애려면.unique나 sort를 사용해라.
- 동적으로 메서드를 호출하려면 앞에서 설명한 invokeMethod를 사용하면된다.
object.invokeMethod(name,params);

```

1 package _C3
2
3 class PersonDAO {
4     public findAll () {
5         'findAll value'
6     }
7     public findAll(def name,def value) {
8         return name + ' ' + value;
9     }
10
11     // 클로저로 처리할수도있다.
12     public a_All = {
13         'a_All_val'
14     }
15     public b_All = { name,value->
16         'b_All name : '+name + ' value:'+value;
17     }
18 }
19
20
21
22
23 def action = 'findAll' // some external input
24 Object[] params = [];
25
26 dao = new PersonDAO()
27 println dao.invokeMethod (action,params);
28
29 params = ['findAllName','findValue']
30 println dao.invokeMethod (action,params);
31
32
33 // 클로저로 처리할수도있다.
34 println dao['a_All']()
35 println dao['b_All'](*['bname','bvalue'])
36
37 /*결과
38 findAll value
39 findAllName findValue
40 a_All_val
41 b_All name : bname value:bvalue
42 */

```

Object의 편리한 메서드들

println은 사실 this.println을 줄여서 쓴것이다. GDK에서 Object에 println을 추가되었기때문이다.
소스어디에서나 쓸수있는것이다.
==를 개체의 일치가 아닌 값의 일치(즉,등가)로 사용하기 때문에 ==에 대응방법으로 is가 있다.

is(other)	객체의 일치검사
isCase(caseValue,switchValue)	비교메서드
obj.identity(closure)	객체 자신을 클로저의 수신자로 지정(위임)
print(),print(value)	출력
printf(formatter,value)	
printf(formaterStr,value[])	포맷출력
sleep(millis),sleep(millis){onInterrupt}	static Thread.curretThread.sleep(millis)
us(categoryClass(closure), use(categoryClassList) {closure})	클로저 내부에서 categoryClass로 지정된 메서드 사용

object 편리한 메서드

```

1  package _C4
2
3  class C_4{
4      def g1=044;
5  }
6
7
8  //is
9  def c1 = new Date(1);
10 def c2 = new Date(2);
11 def c3 = new Date(2);
12 def c4 = c3;
13 println 'c1==c2 : '+(c1==c2)
14 println 'c2==c3 : '+(c2==c3)
15 println 'c3==c4 : '+(c3==c4)
16 println 'c3.is(c2) : '+(c3.is(c2))
17
18
19
20 //객체를 그 클로저의 수신자로 사용한다.
21 new Date().identity{
22     println "$date.$month.$year"
23 }
24 new C_4().identity{
25     println "$g1"
26 }
27
28 printf('hi %s age : %d', 'hkh',29);
29
30 println ""
31 //sleep
32 def text = 'show me the money';
33 for ( c in text){
34     sleep(100);
35     print c;
36 }
37
38
39 println ""
40 //객체 되풀이 메서드
41 def list=[1,2,3,4,5,6,7,8,0];
42 list.each {
43     print it+' ';
44 }
45
46 /*
47 결과
48 c1==c2 : false
49 c2==c3 : true
50 c3==c4 : true
51 c3.is(c2) : false
52 13.1.114
53 36
54 hi hkh age : 29
55 show me the money
56 1 2 3 4 5 6 7 8 0

```

Object되풀이 메서드

리턴	메서드
Boolean	any(closure)
List	collect(closure)
Collection	collect(Collection collection){closure}
(void)	each(closure)

(void)	eachWithIndex{closure}
Boolean	every{closure}
Object	find{closure}
List	findAll{closure}
Integer	findIndexOf{closure}
List	grep{closure}

File의 객체 되돌이 메서드

```

1 package _C5
2
3 file = new File('W:\code.googlegroovy-visualkhhgroovy_studysrc_C5C5.groovy')
4 file.eachLine{println it}
5 println ""
6 println ""
7 println('-----file.readLines()-----');
8 def lines = file.readLines(); //list로 반환한다
9 println lines;
10 println('-----lines.any(it =~/File/)-----');
11 println lines.any{it =~/File/};
12 println('-----lines.grep(it =~/File/)-----');
13 println lines.grep{it =~/File/};
14 /*
15 결과
16 package _C5
17
18 file = new File('W:\code.googlegroovy-visualkhhgroovy_studysrc_C5C5.groovy')
19 file.eachLine{println it}
20 println ""
21 println ""
22 println('-----file.readLines()-----');
23 def lines = file.readLines(); //list로 반환한다
24 println lines;
25 println('-----lines.any(it =~/File/)-----');
26 println lines.any{it =~/File/};
27 println('-----lines.grep(it =~/File/)-----');
28 println lines.grep{it =~/File/};
29
30 -----file.readLines()-----
31 [package _C5, , file = new File('W:\code.googlegroovy-visualkhhgroovy_studysrc_C5C5.groovy'), file.eachLine{println it}, println "", println
32 -----lines.any(it =~/File/)-----
33 true
34 -----lines.grep(it =~/File/)-----
35 [file = new File('W:\code.googlegroovy-visualkhhgroovy_studysrc_C5C5.groovy'), println("-----lines.any(it =~/File/)-----");, p
36
37 */
38

```

GDK의 파일 및 입출력 메서드

메서드	File	InputStream	Reader	URL	OutputStream	BufferedReader	BufferedWriter	ObjectInputStream	Writer
append	2								
asWritable	2								
eachByte	1	1		1					
eachDir	1								
eachFile	1								
eachFileRecurse	1								
eachLine	1	1	1	1					
eachObject	1							1	
filterLine	2	2	2						
getText	2	2	1	2		1			
leftShift <<	1				3				1
newInputStream	1								
newObjectInputStream	1								
newOutputStream	1								
newPrintWriter	2								

newReader	2	1							
newWriter	4								
readBytes	1								
readLine		1	1						
readLines	1	1	1						
splitEachLine	1		1						
transformChar			1						
transformLine			1						
withInputStream	1								
withOutputStream	1								
withReader	1	1	1	1					
withStream		1		1					
withWriter	2				2				
withWriterAppend	1								
write	2								
writeLine							1		

파일시스템 탐색하기

```

1 package _C5
2
3 homedir = new File('src');
4 println homedir;
5 println homedir.name;
6 println homedir.absolutePath;
7 println homedir.canonicalPath;
8 println homedir.directory;
9
10
11
12 //homedir = new File('/java/groovy')
13 dirs = []
14 homedir.eachDir{dirs << it.name } // #1 //디렉터리이름 기록 클로저
15 println dirs
16
17 cvsdir = new File('./img')
18 files = []
19 cvsdir.eachFile{files << it.name} // #2 //파일이름 기록 클로저
20 println files;
21
22 files = []
23 cvsdir.eachFileMatch(~/.swing.*){files << it.name} // #3 //패턴과 일치하는 파일 이름 기록 클로저
24 println files;
25
26 docsdir = new File('src');
27 count = 0
28 files = []
29 docsdir.eachFileRecurse{if (it.directory){files<<it; count++}} // #4 재귀적으로 디렉터리를 세는 클로저
30 println files;
31
32 /*결과
33 src
34 src
35 W:\code\google\groovy-visualkhh\groovy_studysrc
36 W:\code\google\groovy-visualkhh\groovy_studysrc
37 true
38 [.svn, _1, _10, _14, _16, _17, _18, _19, _2, _20, _21, _22, _23, _24, _25, _26, _27, _3, _4, _5, _6, _7, _8, _9, _B1, _B10, _B2, _B3, _B4, _B5,
39 [GroovyHelper.jpg, GroovyInterceptable.jpg, GroovyObject.jpg, GroovyObject_UML.jpg, invokeMethod.jpg, metaDia.jpg, nodebuilder_d.jpg,
40 [swing_b_layout.jpg, swing_b_pwd.jpg, swing_b_s.jpg]
41 [src.svn, src.svnprop-base, src.svnprops,...]
42
43 */

```

파일읽기

```

1 package _C6
2 example = new File('data/text')
3 println example.readLines() //리스트로 반환
4
5 example.eachLine { //한줄씩
6     println 'startWith:' + it.startsWith('show') + ' ('+it+')'
7 }
8

```


필터와 변환기 사용

```
1 package C9
2
3 reader = new StringReader('abc')
4 writer = new StringWriter()
5
6 reader.transformChar(writer) { //reader에 문자 수만큼 클로저 호출된다 , 인자로받은 writer에 리턴값을 쓴다.
7     it.next(); //reader문자수만큼 돈다.
8 } //1
9 println writer.toString()
10
11
12
13 reader = new File('data/text').newReader() //리더를 리턴받는다.
14 writer = new StringWriter()
15
16 reader.transformLine(writer) { //line만큼 돈다. 인자로 받은 writer 에 리턴값을 쓴다.
17     it - 'o'
18 } //2
19 println writer.toString()
20
21
22
23 input = new File('data/text')
24 writer = new StringWriter()
25
26 input.filterLine(writer) { //필터를 건다 true , false로 리턴중 true만 write에 쓴다.
27     it =~ /show/
28 }
29 println writer.toString()
30
31 writer = new StringWriter()
32 writer << input.filterLine { //리턴값이 true면 writer에 쓴다.
33     it.size() > 8
34 } //4
35 println writer.toString()
36 /*결과
37 bcd
38 shw me the money
39 shw
40 me
41 the
42 mney
43
44 show me the money
45 show
46
47 show me the money
48 */
```

base64 Encode,Decode 하는것을 보자.유용하다.

```
1 package C9
2
3 byte[] data = new byte[256]
4 for (i in 0..255) { data[i] = i }
5
6 println data;
7
8 store = data.encodeBase64().toString()
9
10 println store;
11
12 println ' store.startsWith('AAECAwQFBg') : ' + store.startsWith('AAECAwQFBg')
13 println ' store.endsWith('r7/P3+/w==') : ' + store.endsWith('r7/P3+/w==')
14
15 restored = store.decodeBase64()
16
17 assert data.toList() == restored.toList()
```

직렬화한 객체 전송하기

자바에서 직렬화 인터페이스 (Serializable) 구현하면 객체를 특정형태로 저장하여 나중에 쓸수있다.
GDK에서 어떻게하나보자.

객체 직렬화 저장, 불러오기


```

1 package C10
2
3 file = new File('data/objects.dta')
4 out = file.newOutputStream()
5 oos = new ObjectOutputStream(out)
6
7 objects = [1, "Hello Groovy!", new Date()]
8 objects.each { // #1
9     oos.writeObject(it) // #1
10 } // #1
11 oos.close()
12
13 retrieved = []
14 file.eachObject { // 객체별로 뽑아낸다 여기서선 3개씩 돌게 된다. 1, "Hello Groovy!", new Date()
15     retrieved << it
16 } // #2
17
18 println retrieved
19 println objects
20 println retrieved==objects
21
22 /*결과
23 --
24 --
25 --
26 [1, Hello Groovy!, Thu Feb 13 23:37:05 KST 2014]
27 [1, Hello Groovy!, Thu Feb 13 23:37:05 KST 2014]
28 true
29 */

```

그루비 쓰레드

간단한 쓰레드 실행하는 문법을 살펴보자

```

1 package C11
2
3 //쓰레드를 시작한다.
4 def t = new Thread(){ /*...*/}
5 t.start();
6
7 //쓰레드를 시작한다
8 def t1= new Thread().start{println 'start Thread'}
9 println t1;
10
11 def t2 = Thread.start { // new Thread.start와 차이점이 무엇일까?
12     println 'start Thread.start'
13 }
14 println t2;
15
16
17 //데몬쓰레드를 시작한다
18 Thread.startDaemon {println 'start Daemon Thread'}
19
20 //1초후에 시작한다
21 new Timer().runAfter (1000){ println 'After 1000 start Thread' }
22 /*결과
23 start Daemon Thread
24 start Thread
25 After 1000 start Thread
26 */1000 start Thread

```

쓰레드 예제 한번 분석해보자 sleep값을 바꿔가면서 반응을 확인해보자

```

1 package C11
2
3 import java.util.List;
4
5 class Storage {
6     List stack = []
7     synchronized void leftShift(value){ // #1
8         stack << value
9         println "push: $value"
10        notifyAll() // #2
11    }
12    synchronized Object pop() {
13        while (stack.isEmpty()) { // #3
14            try{ wait() } // #3
15            catch (InterruptedException e){}
16        }
17        def value = stack.pop()
18        println "pop : $value"
19        return value
20    }

```

```

21 }
22 storage = new Storage()
23
24 Thread.start {                //#4
25     for (i in 0..9) {        //#4
26         storage << i          //#4
27         sleep 100             //#4
28     }                        //#4
29 }                            //#4
30
31 Thread.start {                //#5
32     10.times {                //#5
33         sleep 200             //#5
34         value = storage.pop()  //#5
35     }                        //#5
36 }                            //#5
37 /*
38 우선 wait는 기다리라는 뜻이며 notify와 notifyAll은 알린다는 의미입니다.
39 */

```

외부프로세스와 연결하기

프로세스를 생성하려면 우선 실행할 명령어의 문자열을 넣어야한다 GDK에서는 문자열의 execute의 메서드를 호출하면 Process객체를 얻을수있다. 문자열 대신 문자열 리스트(혹배열)에 명령문을 담을수 있다 명령문이 따옴표로 표시된 여러문자열이거나 (인자에 따옴표가 있어서) 문자열 치환이 필요한경우 유용 하다

외부프로세스 사용해보자.

```

1 package C12
2 //문자열 실행
3 def e = `taskmgr`;
4 Process proc = e.execute();
5 println proc.text;
6 //문자열 배열 실행
7 def dircmd = ['cmd','/c','dir']
8 def dir = /Program Files;
9 println (dircmd+dir);
10 proc = (dircmd+dir).execute();
11 println proc.text;
12 //println proc.in.newReader().readLine();
13 //프로세스 스트림 받아올수 있다.
14 println proc.in //입력
15 println proc.err //출력
16 println proc.out //아웃
17 //프로세스 out문자열을 보낼수도 있다.
18 //proc.out<<'aaa'
19 //proc.<<'aaa'
20
21 //끝으로 프로세스를 실행하고 나면 간혹 영원히 끝나지 않는경우가 있다. 강제종료하여 해결한다
22 //그루비에서는 waitForKill(1000) 메서드가있다
23 proc.waitForKill(1000);
24 /*결과
25 [cmd, /c, dir, Program Files]
26 W 드라이브의 볼륨: TUBO_WORKSPACE
27 볼륨 일련 번호: 0C11-E2EE
28 W: 디렉터리
29 java.io.BufferedReader@16614e7
30 java.io.FileInputStream@979e8b
31 java.io.BufferedOutputStream@b754b2
32 */

```

```

W:\>cmd /c dir W:\Program Files
W 드라이브의 볼륨: TUBO_WORKSPACE
볼륨 일련 번호: 0C11-E2EE

W:\> 디렉터리

W:\> 디렉터리

```

프로세스 out문자열을 보낼수도 있다.

```

1 proc.out<<'aaa'
2 proc.<<'aaa'

```

끝으로 프로세스를 실행하고 나면 간혹 영원히 끝나지 않는경우가 있다. 강제종료하여 해결한다
//그루비에서는 waitForKill(1000) 메서드가있다

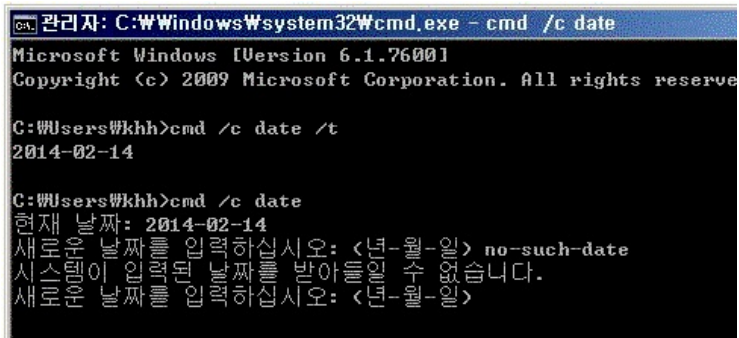
```

1 proc.waitForKill(1000);

```

프로세서와 대화하기.

```
1 package _C13
2
3 println 'cmd /c date /t'.execute().text;
4 today = 'cmd /c date /t'.execute().text.split(/D/)
5
6 println 'cmd /c date'.execute();
7 proc = 'cmd /c date'.execute()
8
9 Thread.start {
10     System.out << proc.in
11 }
12 //Thread.start {
13 //     System.err << proc.err
14 // }
15
16 proc << 'no-such-date' + "\n"
17 println today.join("-")
18 proc << today.join("-") + "\n"
19 println "
20 println "
21 println "
22 proc.out.close()
23 proc.waitForOrKill(0)
24
25 /*
26 2014-02-14
27
28 java.lang.ProcessImpl@1cef4f7
29 2014-02-14
30
31
32
33 현재 날짜: 2014-02-14
34 새로운 날짜를 입력하십시오: (년-월-일) no-such-date
35 시스템이 입력된 날짜를 받아들일 수 없습니다.
36 새로운 날짜를 입력하십시오: (년-월-일)
37 */
```



관리자: C:\Windows\system32\cmd.exe - cmd /c date

Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\wkh>cmd /c date /t
2014-02-14

C:\Users\wkh>cmd /c date
현재 날짜: 2014-02-14
새로운 날짜를 입력하십시오: <년-월-일> no-such-date
시스템이 입력된 날짜를 받아들일 수 없습니다.
새로운 날짜를 입력하십시오: <년-월-일>