Scala

セットアップ

ランタイム

■ Java ランタイム

・http://www.oracle.com/technetwork/jp/java/javase/downloads/index.html

■ Scala ランタイム

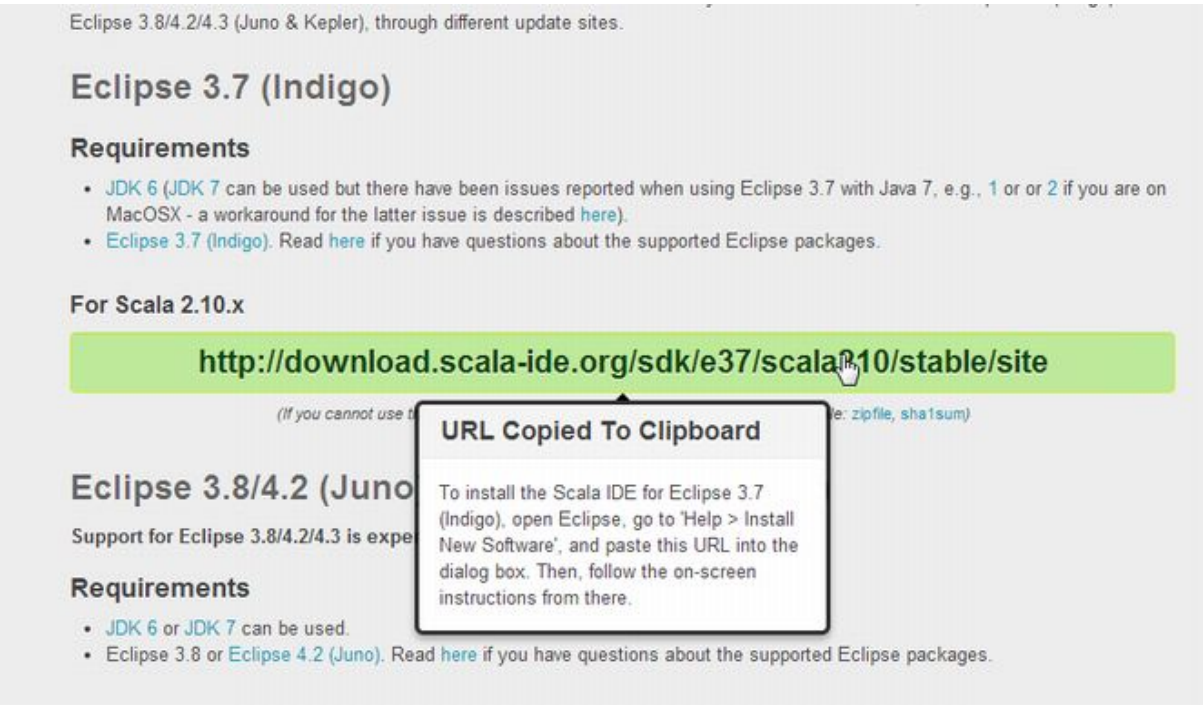
・http://www.scala-lang.org/downloads

Eclipse

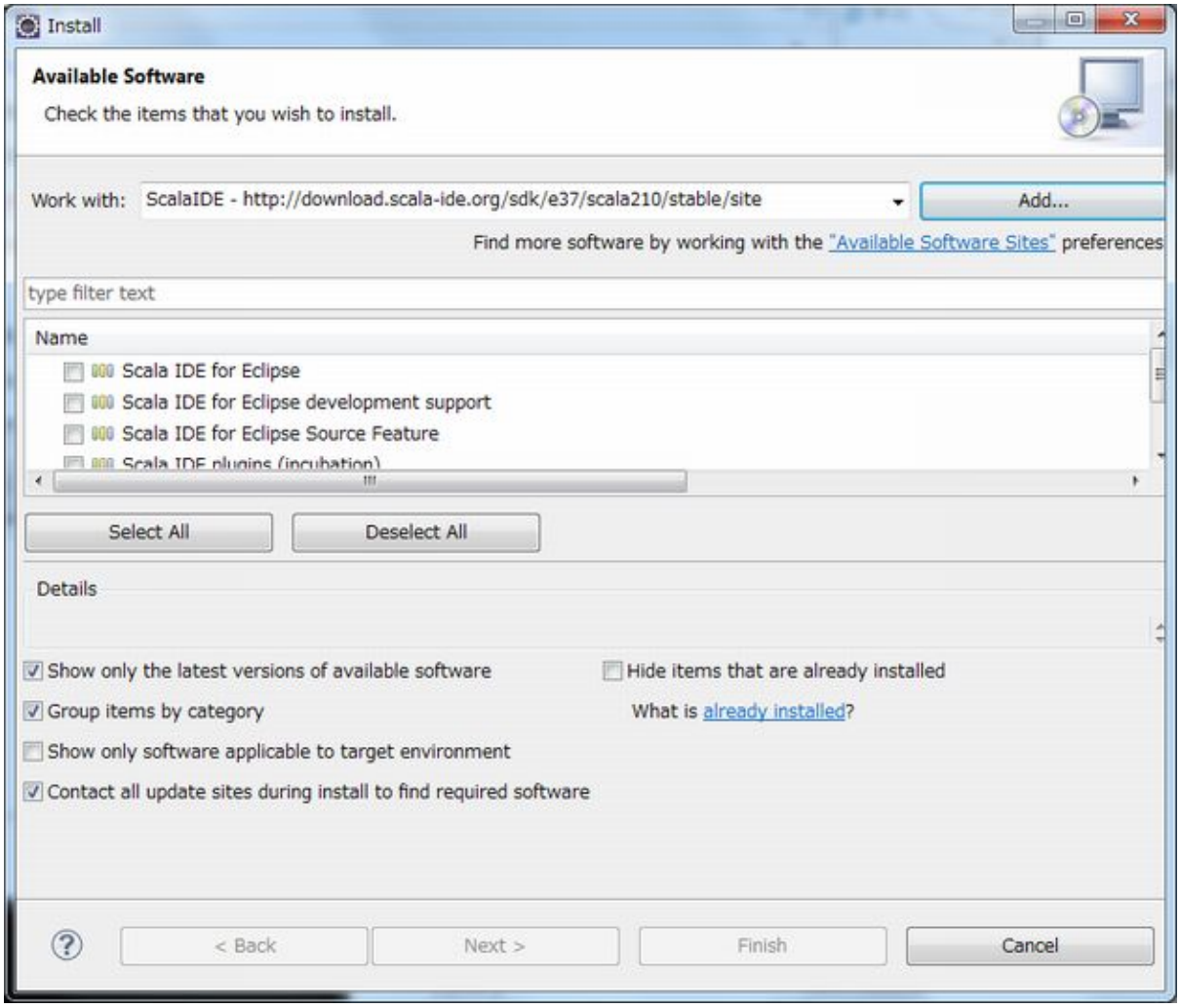
■ Scala IDE for Eclipse

・http://scala-ide.org/

アップデートサイトの URL をコピー



Help - Install New Software に上記 URL を指定してインストール



チュートリアル

Scala Java プログラマ向けチュートリアル

コマンド

実行

■ scala

HelloScala.scala

```
object HelloScala {
    def main(args:Array[String]):Unit = {
        println("hello Scala")
    }
}
```

・ソースコードを指定すると、Java クラスファイルにコンパイルしてから実行する

```
>scala HelloScala.scala
hello Scala
```

・指定しないと対話型シェル (REPL) が起動する
　・:quit で終了

```
>scala
Welcome to Scala version 2.10.2 (Java HotSpot(TM) Client VM, Java 1.7.0_04).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("hello")
hello

scala> :quit
```

タブキーを押すと、補完される

```
scala> import java.io.Buffered
BufferedInputStream    BufferedOutputStream   BufferedReader          BufferedWriter
```

コンパイル

■ scalac

・Java VM で動作するクラスファイルが生成される

```
>scala HelloScala.scala
```

変数

val

・val で宣言した変数は、再代入不可

```
scala> val i = 1
i: Int = 1

scala> i=2
<console>:8: error: reassignment to val
       i=2
```

Scala で関数型スタイルのプログラミングを行う場合の助けとなる。変数をどのタイミングで参照しても同じ値であることが保証される (参照透過性)。同様にイミュータブル (変更不可) のコレクションも提供

var

・var で宣言した変数は再代入可能

```
scala> var i = 1
i: Int = 1

scala> i  =2
i: Int = 2
```

■ lazy var (**遅延評価**)

・実際に必要になるまで、計算を遅延させる

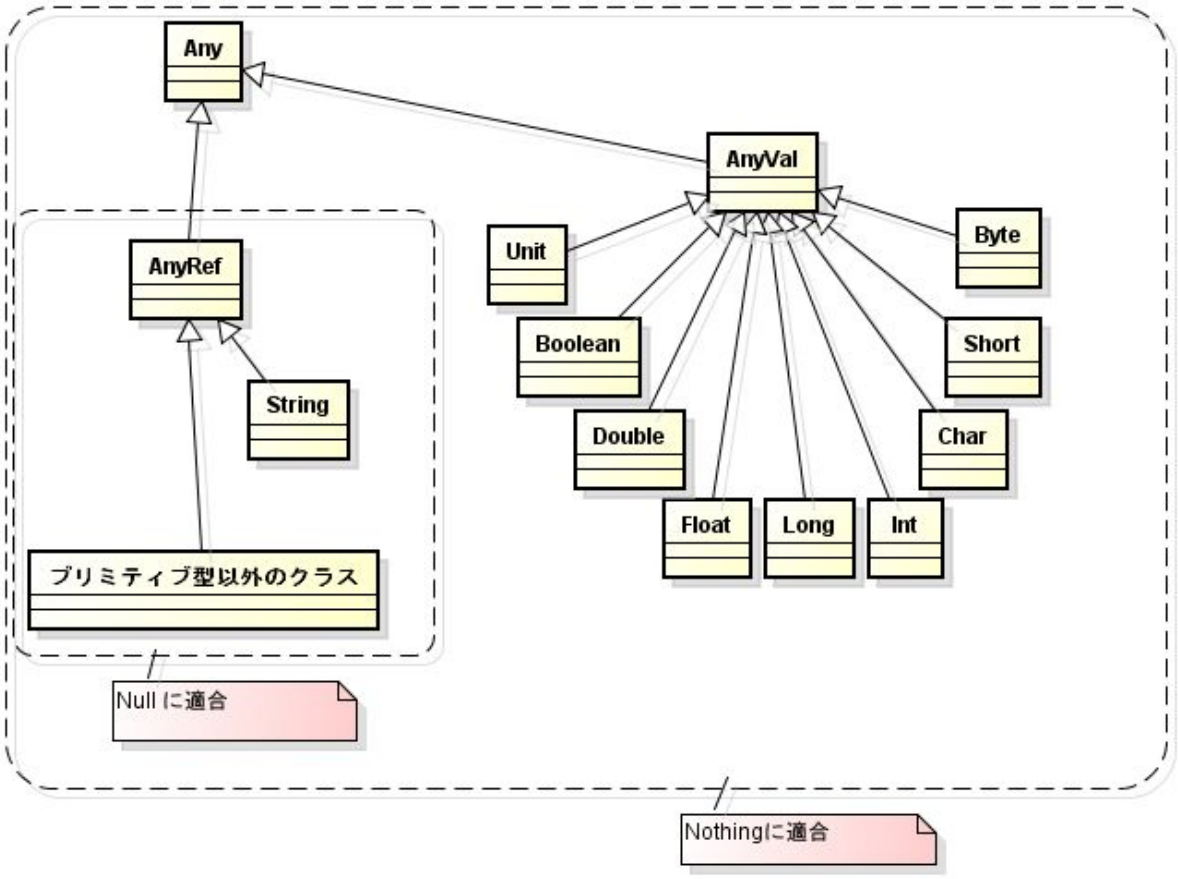
```
scala> val x = 2
x: Int = 2

scala> lazy val y = x * 2
y: Int = <lazy>

scala> print(y)  // ここで計算される
4
```

型

種類



■ Any

・全ての型の基底クラス

■ AnyRef

・参照型の基底クラス

■ AnyVal

・値型の基底クラス
　・プリミティブ型ラッパークラスの基底クラス

■ Unit 型

・Java の void に対応

■ Null

・参照型に適合

■ Nothing

・全ての型に適合

型推論

・Scala では、変数やフィールド、メソッド定義において型の記述を省略できるケースがある
・メソッドの戻り値の型も省略できるが、すべての型を同じにしないと、戻り値の型は、Any になってしまう

すべての型を合わせた場合

```
scala> def test(p:Int) = if(p==0) { 0 } else { 1 }
test: (p: Int)Int
```

戻り値の型が異なる場合、戻り値が Any と判断される

```
scala> def test2(p:Int) = if(p==0) { "zero" } else { 1 }
test2: (p: Int)Any
```

暗黙の型変換

■変換メソッドを用意

・implicit def メソッド名を定義しておくと、自動的に型変換を行うことができる
・必要に応じ自動変換される

```
implicit def メソッド名 ( 引き数名 : 変換前の型 ): 変換後の型
{
    変換処理
}
```

java.util.Date を Calendar に自動変換する例

```
scala> import java.util.{Date,Calendar}
import java.util.{Date, Calendar}

scala> implicit def date2calendar(date:Date):Calendar =
     | {
     |     val cal = Calendar.getInstance()
     |     cal.setTime(date)
     |     cal
     | }

scala> val cal:Calendar = new Date()
cal: java.util.Calendar = java.util.GregorianCalendar[time=1382275606352,areFieldsSet=true,areAllFie
ldsSet=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Tokyo",offset=32400000,dstSavings=
0,useDaylight=false,transitions=10,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YE
AR=2013,MONTH=9,WEEK_OF_YEAR=43,WEEK_OF_MONTH=4,DAY_OF_MONTH=20,DAY_OF_YEAR=293,DAY_OF_WEEK=1,DAY_OF
_WEEK_IN_MONTH=3,AM_PM=1,HOUR=10,HOUR_OF_DAY=22,MINUTE=26,SECOND=46,MILLISECOND=352,ZONE_OFFSET=3240
0000,DST_OFFSET=0]
```

暗黙の型変換を行うことで、継承を行わずに型を拡張できる (final クラスも拡張できる) が、該当部分のソースを見ただけでは処理がわかりにくくなるので、濫用は避ける

■ Predef

・Predef では、以下の様な暗黙の型変換が事前定義されている

数値同士

・ただし、縮小変換は未定義
・明示的に変換する

```
scala> val l:Long = 123
l: Long = 123

scala> val i:Int = l
<console>:12: error: type mismatch;
 found   : Long
 required: Int
       val i:Int = l

scala> val i:Int = l.toInt
i: Int = 123
```

Java ラッパー型と Scala 型

制御

if

・式なので値を返す

ブロック内で最後に評価された値が、if の戻り値となる

```
scala> val i = 1
i: Int = 1

scala> val s = if(i==0){"zero"} else if(i==1){"one"} else {"other"}
s: String = one
```

・値を返さないブロックが有る場合、戻り値は Any となり、そのブロックが呼ばれた場合、Unit が返される

```
scala> def test(p:Int) = if(p==0){"zero"}
test: (p: Int)Any

scala> test(1)
res7: Any = ()
```

for

```
for ( 変数名 <- コレクション )
```

指定した範囲の例

```
scala> for (i <- 1 to 10) {
     |     println(i)
     | }
1
2
3
4
5
6
7
8
9
10
```

scala.collection.immutable.Range の例

```
scala> for (i <- Range(1,3))
     |     println(i)
1
2
```

List の例

```
scala> val l = List("a","b","c")
l: List[String] = List(a, b, c)
```

```
scala> for (e <-l) {
     |     println(e)
     | }
a
b
c
```

■フィルタ

```
for ( 変数名 <- コレクション if  条件 )
```

例

```
scala> for (o <- Range(1,10) if o % 2 == 0) {
     |     println(o)
     | }
2
4
6
```

```
8
```

■コレクションを返す

```
var 変数 =for( 変数名 <- コレクション )yield {
  処理
}
```

例

```
scala> for (o <- Range(1,10) if o % 2 == 0) yield {
     |     o
     | }
res4: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 4, 6, 8)
```

■ネスト

```
for ( 変数名 <- コレクション ; 変数名 <- コレクション )
```

match 式

他言語の switch にあたる

```
式 match {
case パターン =>{ 処理 }
case _ =>{ 処理 }
}
```

・_ はワイルドカード

マッチしない場合

・scala.MatchError

型でのマッチ

・case 変数 : 型

```
case x:String
case _:Array[String]
```

ケースクラスのパターンマッチ

ケースクラスは、case class キーワードで定義する、主にパターンマッチに利用するクラス

```
case class CodeAndValue(code:Int, value:String);

def matchCodeAndValue(cv:CodeAndValue) = {
    cv match {
        case CodeAndValue(_,"Test1") => {
            println("this is test1")
        }
        case CodeAndValue(2,_) => {
```

```
            println("this code is two")
        }
        case CodeAndValue(code, value) => {
            println(cv.code + " " + cv.value)
        }
    }
}


scala> matchCodeAndValue(CodeAndValue(1,"Test1"))
this is test1

scala> matchCodeAndValue(CodeAndValue(2,"Test2"))
this code is two

scala>  matchCodeAndValue(CodeAndValue(3,"Test3"))
3 Test3
```

List や配列のパターンマッチ

タプルのパターンマッチ

正規表現のパターンマッチ

XML のパターンマッチ

パターンマッチに条件

```
case (x,y) if x == y =>{
}! エラー処理
```

scala.Option

・値があるかないかわからない状態を表すための型
  ・null を使わずに表現
  ・値を取得 (null でない)

・option.get

  ・値もしくは null 取得

・option.orNull

  ・値がない場合デフォルトを返す

・option.getOrElse

・サブクラス
  ・Some
  ・None
・Option(x)
  ・値が null

・None

  ・値が null 以外

・Some

例外処理

- ・Java 同様
  - ・try - catch - finally

・式なので値を返せる

- ・catch ブロックは match 式で型毎の処理
- ・例外処理を強制する仕組みはない

scala.Either

- ・メソッドの戻り値に使う
- ・コンテナ
  - ・型パラメータで指定したいずれかを保持

・正常終了
・・期待した戻り値
・異常終了
・・例外

```
def readFile(fileName:String):Either[Throwable,String] = {
}
```

・呼び出し元ではパターンマッチで結果を取り出せる

```
var ret:Either[Throwable,Srting] = readFile("test".txt)
ret match {
case
}
```

関数

■定義

無名関数

- ・引数と本体を =>(関数矢印) で結ぶ

```
( 引数名 : 型 ,...)=>{
}* 戻り値は最後の式の値
```

- ・省略
  - ・式が一つの場合、中括弧
  - ・引数の型が推論できる場合、型

・引数が一つの場合、丸括弧

■実体

トレイト

・引数の数に応じ、Function0 - 22
　・引数 23 以上でエラー
・関数の実体はトレイトのインスタンス

ファーストクラスオブジェクト

・値として扱うことができる

■高階関数

引数に関数を渡す

・def メソッド名 (引数関数名 (関数の引数型)=> 引数関数戻り値)
・def execute(f:(Int)=>Boolean)=f(10)
　・引数 f には、「引数が Int で Boolean を返す関数」を渡すことができる

関数を生成して返す

```
def hoge:Int => Int = {
}** 戻り値が Int=>Int の関数
```

　・戻り値を生成して返す

・関数リテラルなど

■ネストした関数

ローカル関数

```
def foo(a:Srting)  = {
  def bar(b:String) = {}
}:: 外側の関数からしかアクセスできない
```

■クロージャ

関数の生成時に外部の変数を取り込んで動作する

取り込む変数を自由変数という

def multi(times:Int) = (i:Int) => i * times

・Int 引数の無名関数部分がクロージャ

```
var tenTimes = multi(10)
tenTimes(3)  // 30
```

```
multi(100)(3)  // 300!! 正規表現
```

■ scala.util.matching.Regex

■生成

```
var r = """\d{3}""".r
```

文字列

■ raw 文字列

・"3 つで囲む
・| でインデントをあわせられる

```
scala> val sql = """select
     | *
     | from
     | test"""
sql: String =
select
*
from
test
```

■ 1 行ずつ処理

lines,linesWithSeparatos

■連結

+

StringBuilder

■比較

==,!=,equals

・値の比較

eq,ne

・参照の比較

構成

■ 1 ソースファイル

複数パッケージ

・宣言に中括弧

リファレンス

■ Api

http://www.scala-lang.org/api/current/index.html

■コンソール

入力

・readLine,readf,readXxx

出力

・print,printf,println
・Xxx は、Int などの型

■パッケージ

複数行に分けられる

中括弧で名前空間

■インポート

パッケージ

クラス

別名をつけて

パッケージメンバー

シングルトンクラスメンバー

■ Predef

scala.Predef

・暗黙的にインポートされる