[**Scala入门到精通——第二十八节 Scala与JAVA互操作**](http://blog.csdn.net/lovehuangjiaju/article/details/47685907)

标签： [scala](http://www.csdn.net/tag/scala)[Scala入门到精通](http://www.csdn.net/tag/Scala%e5%85%a5%e9%97%a8%e5%88%b0%e7%b2%be%e9%80%9a)

2015-08-16 19:54 9279人阅读 [评论](http://blog.csdn.net/lovehuangjiaju/article/details/47685907#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lovehuangjiaju/article/details/47685907#report)

本文章已收录于：

[[](http://lib.csdn.net/base/49?source=blogtop) **Scala知识库**](http://lib.csdn.net/base/49?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Scala入门到精通（30） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg Scala（34） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg Scala编程（28） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/lovehuangjiaju/article/details/47685907)

本节主要内容

1. JAVA中调用Scala类
2. Scala中调用JAVA类
3. Scala类型参数与JAVA泛型互操作
4. Scala与Java间的异常处理互操作

1. JAVA中调用Scala类

[**Java**](http://lib.csdn.net/base/javaee)可以直接操作纵[**Scala**](http://lib.csdn.net/base/scala)类，如同Scala直接使用Java中的类一样，例如：

//在Person.scala文件中定义Scala语法的Person类

package cn.scala.xtwy.scalaToJava

class Person(val name:String,val age:Int)

//伴生对象

object Person{

def getIdentityNo()= {"test"}

}

//ScalaInJava.java文件中定义了ScalaInJava类

//直接调用Scala中的Person类

package cn.scala.xtwy.scalaToJava;

public class ScalaInJava {

public static void main(String[] args) {

Person p=new Person("摇摆少年梦", 27);

System.out.println("name="+p.name()+" age="+p.age());

//伴生对象的方法当做静态方法来使用

System.out.println(Person.getIdentityNo());

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

对！就是这么简单，Java似乎可以无缝操纵Scala语言中定义的类，在trait那一节中我们提到，如果trait中全部是抽象成员，则它与java中的interface是等同的，这时候java可以把它当作接口来使用，但如果trait中定义了具体成员，则它有着自己的内部实现，此时在java中使用的时候需要作相应的调整。我们先看下trait中全部是抽象成员的情况，例如：

//全部是抽象成员，与java的interface等同

trait MySQLDAO{

def delete(id:String):Boolean

def add(o:Any):Boolean

def update(o:Any):Int

def query(id:String):List[Any]

}

//MySQLDAO字节码反编译后的结果

D:\ScalaWorkspace\ScalaChapter28\bin\cn\scala\xtwy\scalaToJava>javap MySQLDAO.cl

ass

Compiled from "MySQLDAO.scala"

public interface cn.scala.xtwy.scalaToJava.MySQLDAO {

public abstract boolean delete(java.lang.String);

public abstract boolean add(java.lang.Object);

public abstract int update(java.lang.Object);

public abstract scala.collection.immutable.List<java.lang.Object> query(java.l

ang.String);

}

//java直接implement，与普通的java接口一样

public class MySQLDAOImpl implements MySQLDAO{

@Override

public boolean delete(String id) {

// TODO Auto-generated method stub

return false;

}

@Override

public boolean add(Object o) {

// TODO Auto-generated method stub

return false;

}

@Override

public int update(Object o) {

// TODO Auto-generated method stub

return 0;

}

@Override

public List<Object> query(String id) {

// TODO Auto-generated method stub

return null;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48

那如果Trait中包括了具体的成员，此时又该怎么使用呢？此时需要作特殊处理，代码如下：

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

trait MySQLDAO {

//具体方法

def delete(id:String):Boolean={ true}

def add(o:Any):Boolean

def update(o:Any):Int

def query(id:String):List[Any]

}

//JAVA语言实现带有具体成员方法的MySQLDAO

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

public class MySQLDAOImpl implements MySQLDAO {

@Override

public boolean delete(String id) {

//调用生成带有具体delete方法实现的MySQLDAO$class

if (MySQLDAO$class.delete(this, id)) return true;

else return false;

}

@Override

public boolean add(Object o) {

return false;

}

@Override

public int update(Object o) {

return 0;

}

@Override

public List<Object> query(String id) {

return null;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40

用javap命令查看带具体成员方法的trait MySQLDAO时，其代码是一样的

D:\ScalaIntellijIDEAWorkSpace\out\production\ScalaChapter28\cn\scala\xtwy\JavaRe

vokeScala>javap MySQLDAO$class.class

Compiled from "MySQLDAO.scala"

public abstract class cn.scala.xtwy.JavaRevokeScala.MySQLDAO$class {

public static boolean delete(cn.scala.xtwy.JavaRevokeScala.MySQLDAO, java.lang

.String);

public static void $init$(cn.scala.xtwy.JavaRevokeScala.MySQLDAO);

}

D:\ScalaIntellijIDEAWorkSpace\out\production\ScalaChapter28\cn\scala\xtwy\JavaRe

vokeScala>javap MySQLDAO.class

Compiled from "MySQLDAO.scala"

public abstract class cn.scala.xtwy.JavaRevokeScala.MySQLDAO$class {

public static boolean delete(cn.scala.xtwy.JavaRevokeScala.MySQLDAO, java.lang

.String);

public static void $init$(cn.scala.xtwy.JavaRevokeScala.MySQLDAO);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

但其实并不是这样的，经本人查阅相关资料发现，可能是scala版本原因导致的，这篇文献中提到的跟实际情况应该是稳合的<http://www.importnew.com/6188.html>   
这篇文章中给出了下面这样一个trait的定义：

trait MyTrait {

def traitName:String

def upperTraitName = traitName.toUpperCase

}

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

它生成下面两个字节码文件MyTrait.class、MyTrait$class

[local ~/projects/interop/target/scala\_2.8.1/classes/com/twitter/interop]$ javap MyTrait

Compiled from "Scalaisms.scala"

public interface com.twitter.interop.MyTrait extends scala.ScalaObject{

public abstract java.lang.String traitName();

public abstract java.lang.String upperTraitName();

}

[local ~/projects/interop/target/scala\_2.8.1/classes/com/twitter/interop]$ javap MyTrait$class

Compiled from "Scalaisms.scala"

public abstract class com.twitter.interop.MyTrait$class extends java.lang.Object{

public static java.lang.String upperTraitName(com.twitter.interop.MyTrait);

public static void $init$(com.twitter.interop.MyTrait);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

这种情况应该是跟实际情况稳合的，trait MyTrait会自动生成一个名为MyTrait的interface，MyTrait$class的抽象类。我们可以看到，该作者的scala版本是2.8.1，而我们的scala版本是2.10.4，至于为什么出现这样的原因，本人暂时还没有弄清楚，但可以肯定的是，<http://www.importnew.com/6188.html>这篇文章讲的内容跟实际是稳合的，因为前面的MySQLDAOImpl仍然是实现MySQLDAO接口方式定义的，但在重写delete方法时采用的是

@Override

public boolean delete(String id) {

//调用生成带有具体delete方法实现的MySQLDAO$class

if (MySQLDAO$class.delete(this, id)) return true;

else return false;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

这种方式进行方法的实现，即MySQLDAO$class是个抽象类，该抽象类中包含了MySQLDAO中实现的方法。也即

trait MySQLDAO {

//具体方法

def delete(id:String):Boolean={ true}

def add(o:Any):Boolean

def update(o:Any):Int

def query(id:String):List[Any]

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

最终反编译后的代码应该具有以下形式：

public cn.scala.xtwy.JavaRevokeScala.MySQLDAO extends scala.ScalaObject{

public abstract boolean delete(java.lang.String);

public abstract boolean add(java.lang.Object);

public abstract int update(java.lang.Object);

public abstract scala.collection.immutable.List<java.lang.Object> query(java.l

ang.String);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

值得注意的是在Scala IDE for Eclipse中不能实现下列代码的调用

@Override

public boolean delete(String id) {

//调用生成带有具体delete方法实现的MySQLDAO$class

if (MySQLDAO$class.delete(this, id)) return true;

else return false;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

只有在Intellij IDEA中才能正确使用，从这点上也说明了Intellij IDEA在编写scala应用程序时更贴近实际。

2. Scala中调用JAVA类

Scala可以直接调用Java实现的任何类，只要符合scala语法就可以，不过某些方法在JAVA类中不存在，但在scala中却存在操作更简便的方法，例如集合的foreach方法，在java中是不存在的，但我们想用的话怎么办呢？这时候可以通过隐式转换来实现，scala已经为我们考虑到实际应用场景了，例如：

import java.util.ArrayList;

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

class RevokeJavaCollections {

def getList={

val list=new ArrayList[String]()

list.add("摇摆少年梦")

list.add("学途无忧网金牌讲师")

list

}

def main(args: Array[String]) {

val list=getList

//因为list是java.util.ArrayList类型，所以下这条语句编译不会通过

list.foreach(println)

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

此时只要引入scala.collection.JavaConversions.\_包就可以了，它会我们自动地进行隐式转换，从而可以使用scala中的一些非常方便的高阶函数，如foreach方法，代码如下：

package cn.scala.xtwy.ScalaRevokeJava

import java.util.ArrayList;

//引入下面这条语句后便可以调用scala集合中的方法，如foreach,map等

import scala.collection.JavaConversions.\_

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

object RevokeJavaCollections{

def getList={

val list=new ArrayList[String]()

list.add("摇摆少年梦")

list.add("学途无忧网金牌讲师")

list

}

def main(args: Array[String]) {

val list=getList

//现在可以调用scala集合中的foreach等方法了

list.foreach(println)

val list2=list.map(x=>x\*2)

println(list2)

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24

前面我们使用的是隐式转换，我们还可以显式地进行转换，例如：

import java.util.ArrayList;

import scala.collection.JavaConversions.\_

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

object RevokeJavaCollections{

def getList={

val list=new ArrayList[String]()

list.add("摇摆少年梦")

list.add("学途无忧网金牌讲师")

list

}

def main(args: Array[String]) {

val list=getList

list.foreach(println)

val list2=list.map(x=>x\*2)

println(list2)

//显式地进行转换

val listStr=asJavaIterable(JavaListString.getListString)

for(i<- listStr)

println(i)

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

下面给出的是Scala集合与Java集合支持的转换操作（取自JavaConversions源码）：

/\*\* A collection of implicit conversions supporting interoperability between

\* Scala and Java collections.

\*

\* The following conversions are supported:

\*{{{ //相互转换

\* scala.collection.Iterable <=> java.lang.Iterable

\* scala.collection.Iterable <=> java.util.Collection

\* scala.collection.Iterator <=> java.util.{ Iterator, Enumeration }

\* scala.collection.mutable.Buffer <=> java.util.List

\* scala.collection.mutable.Set <=> java.util.Set

\* scala.collection.mutable.Map <=> java.util.{ Map, Dictionary }

\* scala.collection.mutable.ConcurrentMap (deprecated since 2.10) <=> java.util.concurrent.ConcurrentMap

\* scala.collection.concurrent.Map <=> java.util.concurrent.ConcurrentMap

\*}}}

\* In all cases, converting from a source type to a target type and back

\* again will return the original source object, eg.

\*

\*{{{//源类型到目标类型转换，再从转换回去，得到的是相同对象

\* import scala.collection.JavaConversions.\_

\*

\* val sl = new scala.collection.mutable.ListBuffer[Int]

\* val jl : java.util.List[Int] = sl

\* val sl2 : scala.collection.mutable.Buffer[Int] = jl

\* assert(sl eq sl2)

\*}}}

\* In addition, the following one way conversions are provided:

\*

\*{{{ //只支持单向转换的类

\* scala.collection.Seq => java.util.List

\* scala.collection.mutable.Seq => java.util.List

\* scala.collection.Set => java.util.Set

\* scala.collection.Map => java.util.Map

\* java.util.Properties => scala.collection.mutable.Map[String, String]

\*}}}

\*

\* @author Miles Sabin

\* @author Martin Odersky

\* @since 2.8

\*/

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39

3. Scala类型参数与JAVA泛型互操作

Java中的泛型可以直接转换成Scala中的泛型，在前面的课程中我们已经有所涉及，例如Java中的Comparator<T> 可以直接转换成 Scala中的Comparator[T]使用方法完全一样，不同的只是语法上的。下列代码给出了其使用方法：

package cn.scala.xtwy.JavaAndScalaGeneric

import java.util.\_

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

case class Person(val name:String,val age:Int)

//在Java中Comparator是这么用的：Comparator<Person>

//而在Scala中，是这么用的：Comparator[Person]

class PersonComparator extends Comparator[Person]{

override def compare(o1: Person, o2: Person): Int = if(o1.age>o2.age) 1 else -1

}

object ScalaUseJavaComparator extends App{

val p1=Person("摇摆少年梦",27)

val p2=Person("李四",29)

val personComparator=new PersonComparator()

if(personComparator.compare(p1,p2)>0) println(p1)

else println(p2)

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

下面的代码演示了Java是如何使用Scala中的泛型的：

package cn.scala.xtwy.JavaAndScalaGeneric

import scala.beans.BeanProperty

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

//Student类用泛型定义，成员变量name及age指定泛型参数

//并且用注解的方式生成JavaBean规范的getter方法

//因为是val的，所以只会生成getter方法

class Student[T,S](@BeanProperty val name:T,@BeanProperty val age:S){

}

package cn.scala.xtwy.JavaAndScalaGeneric;

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

public class JavaUseScalaGeneric {

public static void main(String[] args){

Student<String,Integer> student=new Student<String,Integer>("小李",18);

//Scala版本的getter方法

System.out.println(student.name());

//JavaBean版本的getter方法

System.out.println(student.getName());

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30

通过上述代码，我们已经十分清楚了Scala中的泛型如何与Java中的泛型进行互操作了，但还有一个问题值得去考虑，那就是Java中的通配符的泛型如何与Scala中的泛型进行操作呢？例如：

package cn.scala.xtwy.JavaAndScalaGeneric;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

public class JavaWildcardGeneric {

//Java的通配符类型，要接受任何类型

public static List<?> getList(){

List<String> listStr=new ArrayList<String>();

listStr.add("摇摆少年梦");

listStr.add("学途无忧网金牌讲师");

return listStr;

}

}

package cn.scala.xtwy.JavaAndScalaGeneric

import java.util.List

import scala.collection.JavaConversions.\_

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

class ScalaExistTypeToJavaWildcardGeneric1 {

//采用Scala中的存在类型与Java中的能匹符泛型进行互操作

def printList(list: List[T] forSome {type T}):Unit={

//因为我们引入了import scala.collection.JavaConversions.\_

//所以可以直接调用foreach方法

list.foreach(println)

}

//上面的函数与下面的等同

def printList2(list: List[\_]):Unit={

list.foreach(println)

}

}

object Main extends App{

val s=new ScalaExistTypeToJavaWildcardGeneric1

s.printList(JavaWildcardGeneric.getList)

s.printList2(JavaWildcardGeneric.getList)

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44

4. Scala与Java间的异常处理互操作

Java中的异常处理具有如下形式：

package cn.scala.xtwy.ScalaAndJavaException;

import java.io.File;

import java.io.IOException;

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

public class JavaExceiptionDemo {

public static void main(String[] args) {

File file = new File("a.txt");

if (!file.exists()) {

try {

file.createNewFile();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23

Scala中的异常处理是通过模式匹配来实现的，代码如下：

package cn.scala.xtwy.ScalaAndJavaException

import java.io.File

/\*\*

\* Created by 摇摆少年梦 on 2015/8/16.

\*/

object ScalaExceptionDemo extends App{

val file: File = new File("a.txt")

if (!file.exists) {

try {

file.createNewFile

}

catch {

//通过模式匹配来实现异常处理

case e: IOException => {

e.printStackTrace

}

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22

上面给的例子是Scala如何捕获Java中抛出的异常，下面的例子给出的是Java如何捕获Scala中声明的异常，代码如下：

package cn.scala.xtwy.ScalaAndJavaException

class ScalaThrower {

//Scala利用注解@throws声明抛出异常

@throws(classOf[Exception])

def exceptionThrower {

throw new Exception("Exception!")

}

}

//Java中调用ScalaThrower(Scala类），然后捕获其抛出的异常

public class JavaCatchScalaThrower {

public static void main(String[] args){

ScalaThrower st=new ScalaThrower();

try{

st.exceptionThrower();

}catch (Exception e){

e.printStackTrace();

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

通过本节，我们基本能掌握Scala与Java的互操作，当然这里面还有很多内容没有涉及，但在日常开发工作当中，掌握本节讲的内容便可以应付绝大多数互操作问题。