目录

[Day12.    Java](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416265)

[1     回顾](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416266)

[2     异常（续）](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416267)

[2.1   throws](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416268)

[2.2       处理异常的两种方式](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416269)

[2.3   RuntimeException](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416270)

[练习1   异常](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416271)

[2.4   throw](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416272)

[练习2  throw](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416273)

[2.5       异常包装](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416274)

[练习3   异常包装](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416275)

[2.6       自定义异常](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416276)

[练习4   自定义异常](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416277)

[2.7   RuntimeException 和 其他Exception](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416278)

[2.7.1  RuntimeException](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416279)

[2.7.2   其他异常](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416280)

[2.8   throws 和 try-catch](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416281)

[3     接口](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416282)

[练习5   变形金刚](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416283)

[4     集合（续）](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416284)

[4.1       集合的继承结构](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416285)

[5     HashMap（续）](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416286)

[5.1       哈希运算过程](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416287)

[5.2   hashCode()](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416288)

[练习6   哈希运算测试](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416289)

[6     作业](http://code.tarena.com.cn/CGBCode/cgb1903/01-java/%b1%ca%bc%c7/day12.htm#_Toc6416290)

**Day12.  Java**

**1   回顾**

  Object

  toString()   地址

  equals()     地址

  String

  封装 char[] 数组

  常量池

  不可变，加号连接，效率低

s1 = "aaa";

s2 = "bbb";

s3 = "ccc";

s4 = s1 + s2 + s3;

s5 = "aaa" + "bbb" + "ccc";

编译器编译优化：

s5 = "aaabbbccc";

  StringBuilder

  可变字符序列

  append()

  数组默认 16，存满翻倍+2

  StringBuilder和StringBuffer

  StringBuilder 线程不安全，效率高

  StringBuffer 线程安全

  Integer

  Integer.valueOf()

256个缓存对象，-128到127

  BigDecimal、BigInteger

  精确浮点数运算、超大整数运算

  ArrayList

  封装数组操作

  数组默认长度10，1.5倍增长

  访问任意位置效率高

  增删数据，效率可能降低

  LinkedList

  双向链表

  首尾效率高

  两端操作数据的方法

  队列和栈操作方法

  HashMap

  哈希表、散列表

  存放键值对

  put(k,v) get(k) remove(k) size()

**2   异常（续）**

  错误信息：

  类型

  消息

  行号

  继承结构

  Throwable

  Error

  Exception

  其他异常

  RuntimeException

  ...

  try-catch-finally

**2.1     throws**

在方法上，设置异常的抛出管道

void f() throws X,Y,Z {

}

**2.2     处理异常的两种方式**

  throws

  try-catch

有异常时，两种处理方式，必须二选一，

如果不添加 throws，也不捕获异常，编译失败

**2.3     RuntimeException**

存在默认的抛出管道，RuntimeException和它的子类型异常，都可以从默认管道抛出

**练习1  异常**

项目：day1201\_异常

类：day1201.Test1

**package** day1201;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) **throws** ParseException {

    /\*

     \* 输入生日(yyyy-MM-dd)：1999-9-12

     \*

     \* "1999-9-12" ---解析---> Date实例

     \*

     \* java.util.Date 封装一个毫秒值

     \* java.text.SimpleDateFormat

     \* 日期格式处理工具，

     \* 把Date格式化成字符串

     \* 把日期字符串，解析成Date对象

     \*

     \* 您已经生存了 xxx 天

     \*/

    System.***out***.print("生日(yyyy-MM-dd)：");

    String s = **new** Scanner(System.***in***).nextLine();

    //日期格式处理工具对象

    SimpleDateFormat sdf =

**new** SimpleDateFormat("yyyy-MM-d");

    Date d = sdf.parse(s);//把日期字符串解析成Date实例

    // d.getTime()，访问它内部包含的毫秒值

**long** t = System.*currentTimeMillis*()-d.getTime();

    t = t/1000/60/60/24;

    System.***out***.println("您已经生存了"+t+"天");

}

}

**2.4     throw**

throw 手动抛出异常，执行异常的抛出动作

类比成 return

当程序出现逻辑错误，不自动创建并抛出异常，可以手动判断逻辑错误，手动创建异常对象并抛出

if(...) {

AException e = new AException();

throw e;

}

**练习2  throw**

Test2

**package** day1201;

**import** java.util.Scanner;

**public** **class** Test2 {

**public** **static** **void** main(String[] args) {

*f*();

}

**private** **static** **void** f() {

    /\*

     \* 输入两个浮点数：

     \* 234.23

     \* 0

     \*

     \* 234.23 / 0  Infinity

     \*/

    System.***out***.println("输入两个浮点数：");

**double** a = **new** Scanner(System.***in***).nextDouble();

**double** b = **new** Scanner(System.***in***).nextDouble();

**try** {

**double** c = *divide*(a, b);

        System.***out***.println(c);

    } **catch** (ArithmeticException e) {

        System.***out***.println("不能除零，是我们的错，请鞭笞我们吧！");

        e.printStackTrace();

    }

}

**private** **static** **double** divide(**double** a, **double** b) {

**if** (b == 0) {

        ArithmeticException e =

**new** ArithmeticException("/ by zero");

**throw** e;

    }

**return** a/b;

}

}

**2.5     异常包装**

捕获的异常对象，包装成另一种类型，再抛出

try {

} catch(AException e) {

BException e2 = new BException(e);

throw e2;

}

使用场景：

  不能抛出的异常，包装成能抛出的异常再抛

  多种类型异常，简化成一种类型

**练习3  异常包装**

Date

**package** day1201;

**import** java.text.SimpleDateFormat;

**public** **class** Date **extends** java.util.Date {

@Override

**public** String toString() {

    // "yyyy-MM-dd HH:mm:ss"

    SimpleDateFormat f =

**new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    String s = f.format(**this**);

**return** s;

}

}

Test3

**package** day1201;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

*f*();

}

**private** **static** **void** f() {

    Date d = **new** Date();

    String s = d.toString();

    System.***out***.println(s);

    /\*

     \* 在 d:\ 根目录，创建文件

     \* “2019-04-17 11:09:53.txt”

     \*/

    s = s.replaceAll(":", ".");

    String path = "d:\\"+s+".txt";

    File f = **new** File(path);

    //不想加throws

**try** {

        f.createNewFile();

    } **catch** (IOException e) {

        //也不想在此处修复处理异常，

        //希望把异常再抛到调用位置

        //包装成能抛出的异常再抛

**throw** **new** RuntimeException(e);

    }

}

}

**2.6     自定义异常**

现有的异常类型，不能表示所有业务中的错误情况

A到B转账失败

ZhuanZhangShiBaiException

用户名错误

UsernameNotFoundException

密码错误

WrongPasswordException

自定义异常

  起一个合适的类名

  选择一个合适的父类

  添加合适的构造方法

**练习4  自定义异常**

UsernameNotFoundException

**package** day1201;

**public** **class** UsernameNotFoundException **extends** Exception{

**public** UsernameNotFoundException() {

**super**();

}

**public** UsernameNotFoundException(String message, Throwable cause, **boolean** enableSuppression,

**boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

}

**public** UsernameNotFoundException(String message, Throwable cause) {

**super**(message, cause);

}

**public** UsernameNotFoundException(String message) {

**super**(message);

}

**public** UsernameNotFoundException(Throwable cause) {

**super**(cause);

}

}

WrongPasswordException

**package** day1201;

**public** **class** WrongPasswordException **extends** Exception{

**public** WrongPasswordException() {

**super**();

}

**public** WrongPasswordException(String message, Throwable cause, **boolean** enableSuppression,

**boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

}

**public** WrongPasswordException(String message, Throwable cause) {

**super**(message, cause);

}

**public** WrongPasswordException(String message) {

**super**(message);

}

**public** WrongPasswordException(Throwable cause) {

**super**(cause);

}

}

Test4

**package** day1201;

**import** java.util.Scanner;

**public** **class** Test4 {

**public** **static** **void** main(String[] args) {

    System.***out***.print("用户名：");

    String n = **new** Scanner(System.***in***).nextLine();

    System.***out***.print("密码：");

    String p = **new** Scanner(System.***in***).nextLine();

**try** {

*login*(n, p);

        System.***out***.println("欢迎登录");

    } **catch** (UsernameNotFoundException e) {

        System.***out***.println("用户名错误");

    } **catch** (WrongPasswordException e) {

        System.***out***.println("密码错误");

    }

}

**private** **static** **void** login(String n, String p) **throws** UsernameNotFoundException, WrongPasswordException {

    // "abc"   "123"

**if**(! n.equals("abc")) {

**throw** **new** UsernameNotFoundException();

    }

**if**(! p.equals("123")) {

**throw** **new** WrongPasswordException();

    }

}

}

**2.7     RuntimeException 和 其他Exception**

**2.7.1 RuntimeException**

非检查异常

编译器不检查RuntimeException 是否有处理代码

有默认的抛出管道

**2.7.2 其他异常**

检查异常

编译器检查是否有处理代码，不处理，不能编译

强制开发人员事先考虑错误处理

**2.8     throws 和 try-catch**

不能随便 catch

底层的异常，一般要向上层抛，在上层处理

当经验不足时，不知道该catch还是throws，就选throws

**3   接口**

极端的抽象类

作用：结构设计工具，用来解耦合、隔离实现

用 interface 代替了 class 关键字

用 implements 代替了 extends 关键字

接口中只能定义：

  公开的抽象方法

  公开的常量

  公开的内部类、内部接口

类可以同时继承多个接口

class A implements X,Y,Z {

}

class A extends B implements X,Y,Z {

}

接口和接口之间继承

interface A extends X,Y,Z {

}

**练习5  变形金刚**

项目：day1202\_变形金刚

类：day1202.Test1

             Weapon

             Sword

             AK47

             Lyb

             Transformer

Weapon

**package** day1202;

**public** **interface** Weapon {

/\* public static final \*/

**int** ***TYPE\_COLD***      =0;

**int** ***TYPE\_HOT***    =1;

**int** ***TYPE\_NUCLEAR***   =2;

/\* public abstract \*/

**void** kill();

String getName();

**int** getType();

}

Sword

**package** day1202;

**public** **class** Sword **implements** Weapon {

@Override

**public** **void** kill() {

    System.***out***.println("耍剑");

}

@Override

**public** String getName() {

**return** "倚天剑";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_COLD***;

}

}

AK47

**package** day1202;

**public** **class** AK47 **implements** Weapon {

@Override

**public** **void** kill() {

    System.***out***.println("耍枪");

}

@Override

**public** String getName() {

**return** "AK47";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_HOT***;

}

}

Lyb

**package** day1202;

**public** **class** Lyb **implements** Weapon {

@Override

**public** **void** kill() {

    System.***out***.println("耍棒");

}

@Override

**public** String getName() {

**return** "狼牙棒";

}

@Override

**public** **int** getType() {

**return** Weapon.***TYPE\_NUCLEAR***;

}

}

Transformer

**package** day1202;

**public** **class** Transformer {

//包含一个武器接口

**private** Weapon w; //默认 null

//给变量w赋值的方法

**public** **void** setWeapon(Weapon w) {

**this**.w = w;

}

**public** **void** attack() {

    System.***out***.println("变形金刚进攻");

**if** (w == **null**) {

        System.***out***.println("用牙咬");

**return**;

    }

    String t = "";

**switch**(w.getType()) {

**case** Weapon.***TYPE\_COLD***: t="冷兵器";**break**;

**case** Weapon.***TYPE\_HOT***: t="热兵器";**break**;

**case** Weapon.***TYPE\_NUCLEAR***: t="核武器";**break**;

    }

    //使用冷兵器倚天剑进攻

    System.***out***.println("使用"+t+w.getName()+"进攻");

    w.kill();

}

}

Test1

**package** day1202;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

    Transformer t = **new** Transformer();

    Sword s = **new** Sword();

    AK47 a = **new** AK47();

    Lyb l = **new** Lyb();

    System.***out***.println("按回车执行");

**while**(**true**) {

**new** Scanner(System.***in***).nextLine();

**int** r = **new** Random().nextInt(4);

**switch** (r) {

        //setWeapon()方法中，为变形金刚的成员w赋值

**case** 0: t.setWeapon(s); **break**;

**case** 1: t.setWeapon(a); **break**;

**case** 2: t.setWeapon(l); **break**;

**case** 3: t.setWeapon(**null**); **break**;

        }

        t.attack();

    }

}

}

**4   集合（续）**

ArrayList

LinkedList

HashMap

**4.1     集合的继承结构**

Collection 接口

   |- List 接口

      |- ArrayList

      |- LinkedList

   |- Set 接口

      |- HashSet

      |- TreeSet

Map 接口

   |- HashMap

   |- TreeMap

Iterator 接口

Collections 工具类

**5   HashMap（续）**

**5.1     哈希运算过程**

  HashMap内部，使用 Entry[] 数组存放数据

  数组默认初始容量 16

  放满后容量翻倍+2

  key.hashCode() 获得键的哈希值

  用哈希值和数组长度，运算产生一个下标值 i

  新建Entry 对象来封装键值对数据

  Entry对象，放在 i 位置

  如果是空位置，直接放入

  有数据，依次equals()比较是否相等

  找到相等的，覆盖旧值

  没有相等的，链表连接在一起

  负载率、加载因子 0.75

  新建翻倍容量的新数组

  所有数据，重新执行哈希运算，放入新数组

  jdk1.8

  链表长度到8，转成红黑树

  树上的数据减少到6，转回成链表

**5.2     hashCode()**

hashCode() 是 Object 的方法，默认实现是用内存地址作为哈希值

如果希望相同属性，具有相同哈希值，可以重写这个方法

**练习6  哈希运算测试**

项目：day1203\_哈希运算

类：day1203.Test1

             Point

Point

**package** day1203;

**public** **class** Point {

**private** **int** x;

**private** **int** y;

**public** Point() {

**super**();

}

**public** Point(**int** x, **int** y) {

**super**();

**this**.x = x;

**this**.y = y;

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

@Override

**public** String toString() {

**return** "Point [x=" + x + ", y=" + y + "]";

}

@Override

**public** **int** hashCode() {

    /\*

     \* \*) 相同属性，产生相同的哈希值

     \* \*) 不同属性，要尽量产生分散的哈希值

     \*

     \* 数学家发明了一种算法，

     \* 是一种惯用的，有效算法

     \*/

**int** p = 31; //神奇值31

**int** r = 1;  //r起始值任意

    r = r\*p + x;

    r = r\*p + y;

**return** r;

}

@Override

**public** **boolean** equals(Object obj) {

    //参数obj是Point类型

**if**(obj **instanceof** Point) {

        Point p = (Point) obj;

**return** x == p.x && y == p.y;

    }

**return** **false**;

}

}

Test1

**package** day1203;

**import** java.util.HashMap;

**public** **class** Test1 {

**public** **static** **void** main(String[] args) {

    Point a = **new** Point(1, 3);// -- "2.9亿"

    Point b = **new** Point(1, 3);// -- "3.1亿"

    //有相同的哈希值，才能保证计算出相同下标位置

    System.***out***.println(a.hashCode());

    System.***out***.println(b.hashCode());

    //即使计算出相同位置，

    //equals()也要像等，才能覆盖，否则链表连在一起

    System.***out***.println(a.equals(b));

    HashMap<Point, String> map = **new** HashMap<>();

    map.put(a, "2.9亿");

    map.put(b, "3.1亿");

    System.***out***.println(map);

}

}

**6   作业**

  重写

  day1202\_变形金刚

  Weapon

  Sword,AK47,Lyb........

  Transformer

  Test1

  背诵 哈希运算过程

  内部类

  io

  File

  InputStream

  FileInputStream...