

知识点列表

编号	名称	描述	级别
1	Java 中表示时间的方式	了解 Java 中 3 中表示时间的方法	*
2	时间的类型转换	了解时间的类型转换会出现的问题以及解决办法	*
3	日期的输入与输出	熟练掌握 Java 中的日期处理	***
4	异常处理模块 try catch finally	掌握 Java 中的异常处理方式，抛出和 try catch	***
5	异常的分类	理解 Java 中的 Error 和 Exception，以及 Excpetion 的分类	*
6	自定义异常	掌握并能够运用自定义异常	***

注： **"理解级别 ***"掌握级别 ****"应用级别

目录

1. 时间日期	3
1.1. Java 中表示时间的方式 *	3
1.2. 时间的类型转换 *	4
1.3. 日期的输入与输出 ***	6
2. 异常 (Exception)	9
2.1. 捕获异常 try catch finally ***	13
2.2. 异常的分类 *	16
2.3. 自定义异常 **	17

1. 时间日期

- 1) Java 中的时间类有：Date 和 Calendar
- 2) Java 中时间类的本质
 - Date = long + 操作
 - Calendar = long + 操作
- 3) 时间的标准有两种
 - (1970) GMT long
 - (0) UTC long
- 4) java 中基本的时间表示 **GMT long**
- 5) Java 提供了 **Date** 类型表示时间和日期
 - Date 是 long 类型的包装
- 6) **Calendar** 是历法的的抽象
 - 历法：公历、农历、太阳历 ...
- 7) **GregorianCalendar** 是历法的实现，采用公历（太阳历）算法实现的
Calender cal = Calender.getInstance();
- 8) Date 和 Calendar 的默认值就是系统当前时间

1.1. Java 中表示时间的方式 *

【案例】long、Date、Calendar 演示

```
LongDemo.java X
1 package corejava.day07.ch12;
2 import java.util.Calendar;
5
6 /** GMT Long Demo */
7 public class LongDemo {
8     public static void main(String[] args) {
9         //1. 从格林威治时间到当前系统时间的毫秒数
10         long now = System.currentTimeMillis();
11     }
```

```

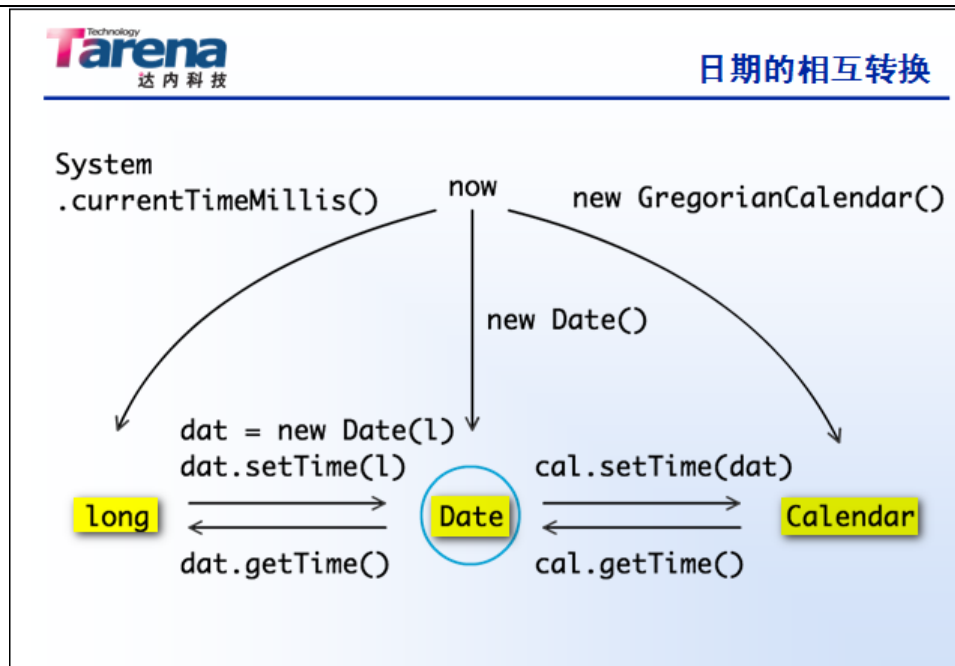
12    //2. 计算现在的年份(不考虑闰年)
13    long year = 1970+now/1000/60/60/24/365;
14    System.out.println(year); //约等于2011
15
16    //3. 当前时间(公历)
17    Date date = new Date(now);
18    //3.1 getYear() 已过时, 有"千年虫"问题
19    int y = date.getYear() + 1900; //1900 偏移量
20    //3.2 getMonth() 已过时, 根据历法规则计算当前时间月份
21    int month = date.getMonth()+1; //以0为第一个月
22    System.out.println(y);
23    System.out.println(month);
24
25    //4. Calendar 抽象历法
26    //4.1 GregorianCalendar 格里高历(就是公元历法)
27    //GregorianCalendar = GMT long +公历算法
28    Calendar cal = new GregorianCalendar();
29    //4.1.1 long 表示1970.1.1
30    long birthday1 = 0;
31    System.out.println(birthday1); //0
32    //4.1.2 Date 表示1970.1.1
33    Date birthday2 = new Date(0);
34    System.out.println(
35        (birthday2.getYear()+1900)+", "
36        +(birthday2.getMonth()+1));
37    //4.1.3 Calendar 表示1970.1.1
38    // Calendar没有"千年虫问题", 不用加"偏移量"1900
39    Calendar birthday3 = new GregorianCalendar(1970,
40        Calendar.JANUARY, 1);
41    System.out.println(
42        (birthday3.get(Calendar.YEAR)) + ", "
43        + (birthday3.get(Calendar.MONTH)+1));
44    }
45 }

```

注:

- ✓ 格林威治时间 (GMT) : 1970 年 1 月 1 日 00:00:00
- ✓ 公元元年 (UTC) : 0 年
- ✓ 目前 Java 中 3 种日期表达方式 long、Date、Calendar 都在用

1.2. 时间的类型转换 *



注：

✓ **now** 表示当前时间

【案例】时间类型转换

```

1 package corejava.day07.ch12;
2 import java.util.Calendar;
3
4
5
6 /** 时间类型转换 */
7 public class DateConvertDemo {
8     public static void main(String[] args) {
9         //1. 演示long -> Date -> Calendar
10        //1.1 GMT 原点, 1970.1.1 0:0:0
11        long time = 0;
12        //1.2 现在时间
13        Date date = new Date();
14        //1.3 long->Date 更改为time的时间 (1970.1.1 0:0:0)
15        date.setTime(time);
16        //结论: 会出现"东八区现象"(多8个小时)
17        System.out.println(date); //Thu Jan 01 08:00:00 CST 1970
18        //1.4 现在时间
19        Calendar cal = new GregorianCalendar();
20        //1.5 Date->Calendar 更改为time的时间
21        cal.setTime(date);
    }
}
    
```

```

22
23 //1.6 减一天(变为1969年最后1天)
24 //本质上: GMT long - 1000*60*60*24
25 cal.add(Calendar.DAY_OF_YEAR, -1);
26
27
28 //2. 演示Calendar -> Date -> long
29 //2.1 1天的毫秒数 86400000
30 System.out.println(1000*60*60*24);
30 System.out.println(1000*60*60*24);
31 //2.2 Calendar->Date
32 Date d = cal.getTime();
33 //2.3 Date->long
34 long l = d.getTime();
35 System.out.println(l); //1970.1.1的前1天
36 }
37 }

```

Console X

<terminated> DateConvertDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 4, 2011 1:32:41 P

Thu Jan 01 08:00:00 CST 1970

86400000

-86400000

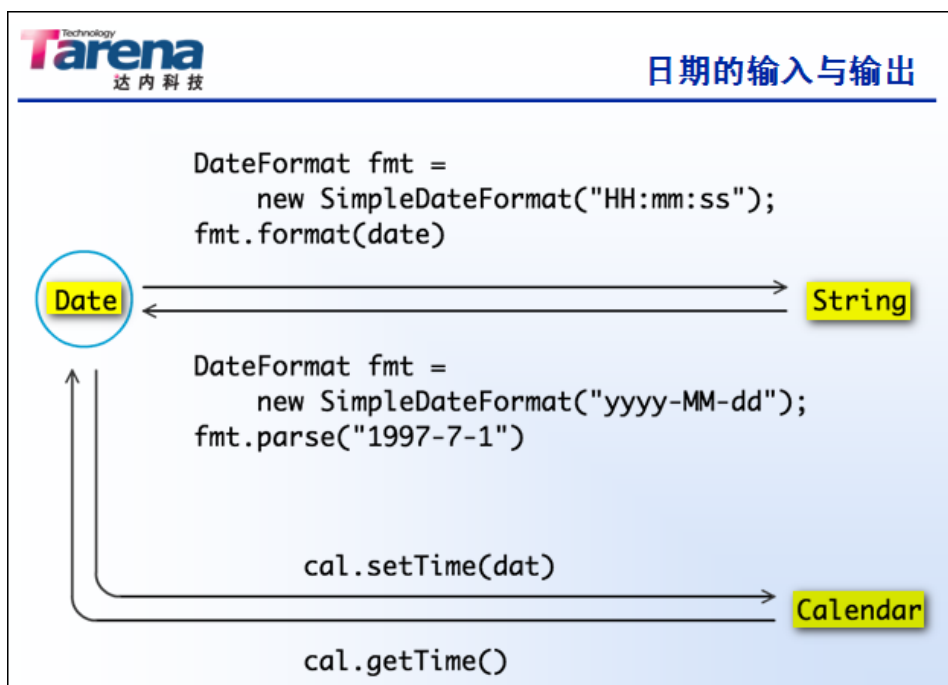
1.3. 日期的输入与输出 ***

- 1) 日期输出的本质是将 Date 转换为格式化的 String
- 2) 日期输入的本质是将格式化的 String 转换为 Date
- 3) `java.text.SimpleDateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");`
 - **日期的输入与输出**

日期输出的本质是 Date 转换为格式化的 String

日期输入的本质是 将格式化的 String 转换为 Date

`java.text.SimpleDateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");`
 - **构建 SimpleDateFormat** 一般提供日期的格式"yyyy-MM-dd" 具体参看 javadoc
 - ◆ 如: "yyyy-MM-dd HH:mm:ss"
 - "yyyyMMdd"
 - "dd/MM/yyyy"
 - "MM/dd/yyyy"
 - **fmt.parse(String)**可以实现将(合法)字符串解析为日期类型, 经常用于处理日期输入
 - fmt.format(Date)**可以把日期格式化为字符串用于输出处理



注：

- ✓ Java 没有提供 String 直接转换为 Calendar 的 API 方法，但第三方提供的 API 有这样的方法

【案例 1】商品促销日期计算

```

SpecialDateDemo.java
1 package corejava.day07.ch13;
2 import java.text.SimpleDateFormat;
6
7 /** 商品促销日期计算 */
8 public class SpecialDateDemo {
9     public static void main(String[] args)
10         throws Exception{
11
    
```

```

12    //1. 日期字符串
13    String d = "2011-05-17";
14    //2. 规定日期格式为"年-月-日"
15    //    常用格式为"yyyy-MM-dd HH:mm:ss"
16    SimpleDateFormat fmt =
17        new SimpleDateFormat("yyyy-MM-dd");// HH:mm:ss
18    //3. 按指定格式将字符串转换为Date(解析)
19    Date date = fmt.parse(d);
20    Date spec = specialDay(date, 3);//计算促销日期
21    //4. 按指定格式将Date转换为String(格式化)
22    String s = fmt.format(spec);//Date -> "yyyy-MM-dd"
23    System.out.println("促销日期:"+s);
24 }
25 /**
26  * 商品促销日期计算, 规则:商品过期的前两周的周五促销
27  * @param proc 商品生产日期
28  * @param exp 保质期月数
29  * @return 促销日期
30  */
31 public static Date specialDay(Date proc, int exp){
32     Calendar cal = new GregorianCalendar();
33     cal.setTime(proc); //生产日期
34     cal.add(Calendar.MONTH, exp); //过期日
35     cal.add(Calendar.WEEK_OF_YEAR, -2); //回调2周
36     //调整到当前周的星期五
37     cal.set(Calendar.DAY_OF_WEEK, Calendar.FRIDAY);
38     return cal.getTime();
39 }
40 }

```

注：

- ✓ yyyy-MM-dd HH:mm:ss 常用的日期格式
- ✓ 解析时有异常 (Exception)，必须处理，本例中为“抛出”

【案例 2】格式化的案例：将数字格式化


```

1 package com.java.day07.ch10;
2 import java.text.DecimalFormat;
3 /** 数字格式化 输入与输出 */
4 public class NumberFormatDemo {
5     public static void main(String[] args)
6         throws Exception{
7         //1. 将"5.6%"转换为数字
8         String s = "5.6%";
9         //1.1 指定数字格式
10        //    "#"表示必须有，0表示可有可无
11        DecimalFormat fmt = new DecimalFormat("0.##%");
12        double d = fmt.parse(s).doubleValue();
13        System.out.println(d);
14
15        //2. 将数字转换为指定格式字符串
16        double x = 0.07555555;
17        System.out.println(fmt.format(x));
18    }
19 }

```

Console

```

<terminated> NumberFormatDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 4, 201
0.055999999999999994
7.56%

```

注：

- ✓ 类似的还有 **NumberFormat**，请查看 API 帮助文档学习
- ✓ **DecimalFormat** 的经典格式：
 - New DecimalFormat("0.##%") 百分比（显示 2 位小数）
 - 000,000,000.## 每 3 位显示 1 个逗号，显示 2 位小数
 如 34523.54 会格式化为 345,23.54

2. 异常 (Exception)

- 1) 什么是异常？异常是行为（方法、过程）的意外结果
- 2) 一个方法如果抛出了异常，这个方法就必须声明异常的抛出
- 3) 异常的声明：在方法上声明方法的意外结果

如：

```

User reg( String pwd, String email) throws UserExistException;
User login(String email, String pwd) throws NameOrPwdException;

```

- 4) 异常类一般继承于 Exception
- 5) 调用抛出异常的方法，必须处理异常，有两种方式

- 使用 try catch finally 捕获
 - 直接再抛出异常
 - 处理异常的方式依赖于具体业务逻辑，可灵活处理
- 6) 如果代码有异常发生，异常以后的代码将不再执行

【案例】用户登录及异常处理

● User.java

```

1 package corejava.day08.ch01;
2 public class User{
3     int id;
4     String email;
5     String pwd;
6     public User() {}
7     public User(int id, String email, String pwd) {
8         super();
9         this.id = id;
10        this.email = email;
11        this.pwd = pwd;
12    }
13    public String toString() { return id+","+email; }
14    public boolean equals(Object obj) {
15        if(obj==null){
16            return false;
17        }
18        if(this==obj)
19            return true;
20        if (obj instanceof User) {
21            User o = (User) obj;
22            return id==o.id;
23        }
24        return false;
25    }
26    public int hashCode() { return id; }
27 }

```

● UserManager.java

```

1 package corejava.day08.ch01;
2 import java.util.HashMap;
3
4
5 /** 该类提供用户的管理功能：注册、登录 */
6 public class UserManager{
7     /** 存储User对象的集合
8      * key 是email, Value是用户对象 */
9     private Map<String, User> users =
10         new HashMap<String, User>();
11     private int id = 1;
12
13     /** 该方法根据email和pwd注册用户
14      * 如果成功，则返回注册的用户对象
15      * 如果email重复就抛出异常，表示已经注册过
16      */
17     public User reg(String email, String pwd)
18         throws UserExistException{ //异常的声明
19         if(users.containsKey(email)){
20             //抛出异常实例，将提前结束方法执行，返回异常实例
21             throw new UserExistException("注册过了!" + email);
22         }
23         User newguy = new User(id++, email, pwd);
24         users.put(email, newguy);
25         return newguy;
26     }
27
28     /** 登录方法，用户名或密码错误时抛出异常 */
29     public User login(String email, String pwd)
30         throws EmailOrPwdException{
31         if(! users.containsKey(email)){
32             throw new EmailOrPwdException("无此用户!");
33         }
34         User u = users.get(email);
35         if(!u.pwd.equals(pwd)){
36             throw new EmailOrPwdException("密码不对!");
37         }
38         return u;
39     }
40 }

```

```

41
42 /** 该类表示"用户注册过异常", 一定继承于Exception类
43  * 用以表示"注册时可能已经注册的异常"
44  */
45 class UserExistException extends Exception{
46     public UserExistException(String message) {
47         super(message);
48     }
49 }
50
51 /** 该类表示"用户名或密码错误异常", 一定继承于Exception类
52  * 用以表示"输入的用户名或密码错误的异常"
53  */
54 class EmailOrPwdException extends Exception{
55     public EmailOrPwdException(String message) {
56         super(message);
57     }
58 }

```

● ExceptionDemo.java

```

ExceptionDemo.java
1 package corejava.day08.ch01;
2
3 public class ExceptionDemo {
4     public static void main(String[] args)
5         throws Exception{
6         UserManager mgr = new UserManager();
7         User u = mgr.reg("abc@sina.com", "123");
8         System.out.println("注册成功");
9
10        //测试1: 重复注册, 出现异常"注册过了!abc@sina.com"
11        // u = mgr.reg("abc@sina.com", "abc");
12        // System.out.println("注册成功");
13
14        //测试2: 登录密码错误, 出现异常"密码不对!"
15        // User someone = mgr.login("abc@sina.com", "123456");
16        // System.out.println(someone);
17
18        //测试3: OK, 登录成功
19        User someone2 = mgr.login("abc@sina.com", "123");
20        System.out.println(someone2);
21    }
22 }

```

注:

- ✓ 代码第 7 行、第 19 行调用抛出异常的方法, 必须处理异常, 有两种方式
 - 使用 try catch finally 捕获
 - 直接再抛出异常

- ✓ 处理异常的方式依赖于具体业务逻辑，可灵活处理

2.1. 捕获异常 try catch finally ***

- 1) **try** 是**尝试运行代码块**，如果有异常会被随后的 catch 捕获，异常发生以后代码不执行
- 2) **catch** 代码块是**异常处理代码**，需要提供合理的处理
 - 异常的处理是与具体业务逻辑有关
 - 可以写多个 catch 处理一系列异常，但是要注意：异常的大小关系，大类型的放到后面处理。
- 3) 有的时候直接 **catch (Exception)** 粗粒度处理异常，**代码简洁**，语义含糊。根据业务逻辑适当选用
- 4) **finally** 代码块，不管是否出现异常，**总会执行的代码块**
 - finally 经常用来处理现场的清理，比如：可靠的数据库连接关闭
- 5) 处理异常有一个**基本原则**：能够底层处理的尽量处理，但是如果不能处理，必须抛出到调用（方法）。不应该简单的抛弃
- 6) 异常捕获再抛出，是一种把底层异常进行封装,转换为另外一种异常类型
- 7) *** 建议在捕获到异常时候使用 **e.printStackTrace()**，打印到控制台
 - 输出内容是：出现异常时候的方法调用堆栈
 - 一般情况下，凡是捕获异常代码都输出：e.printStackTrace()

者

【案例 1】用户登录及异常处理 try catch 捕捉异常

- 版本 01
ExceptionDemo.java

```
ExceptionDemo01.java
1 package corejava.day08.ch01;
2
3 public class ExceptionDemo01 {
4     public static void main(String[] args) {
5         UserManager mgr = new UserManager();
6         try {
7             User u = mgr.reg("abc@sina.com", "123");
8             System.out.println("注册成功");
9             User someone2 = mgr.login("abc@sina.com", "123");
10            System.out.println(someone2);
11        } catch (UserExistException e) {
12            e.printStackTrace();
13        } catch (EmailOrPwdException e) {
14            e.printStackTrace();
15        }
16    }
17 }
```

- 版本 02 循环接收用户信息输入

ExceptionDemo.java

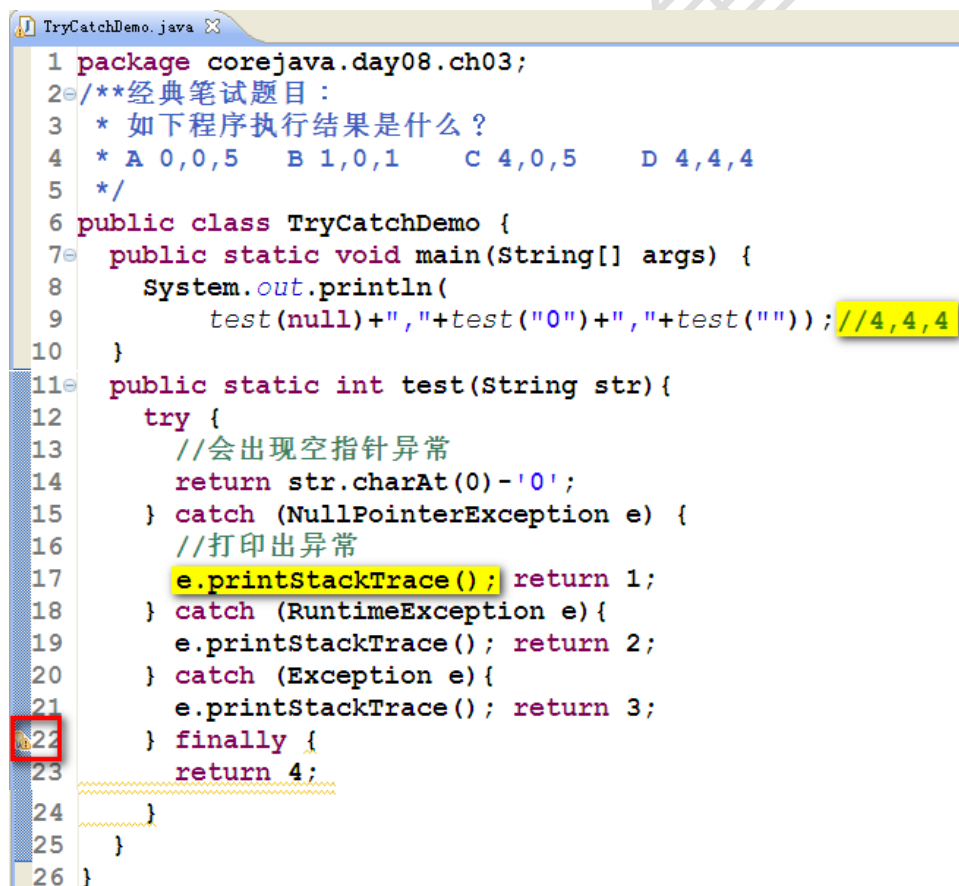
```
ExceptionDemo02.java
1 package corejava.day08.ch01;
2 import java.util.Scanner;
3 public class ExceptionDemo02 {
4     public static void main(String[] args) {
5         UserManager mgr = new UserManager();
6         Scanner s = new Scanner(System.in);
7         while(true) {
8             try {
9                 System.out.print("1 注册, 2 登录:");
10                String cmd = s.nextLine();
11                if ("1".equals(cmd)) {
12                    //注册用户
13                    System.out.print("e-mail:");
14                    String email = s.nextLine();
15                    System.out.print("密码:");
16                    String pwd = s.nextLine();
17                    User u = mgr.reg(email, pwd);
18                    System.out.println("成功注册"+u);
19                } else if ("2".equals(cmd)) {
20                    System.out.print("e-mail:");
21                    String email = s.nextLine();
22                    System.out.print("密码:");
23                    String pwd = s.nextLine();
24                    User u = mgr.login(email, pwd);
```

```

25         System.out.println("成功登录"+u);
26     }else{
27         System.out.println("不可识别命令!");
28     }
29     }catch(UserExistException e){
30         e.printStackTrace();
31     }catch (EmailOrPwdException e) {
32         e.printStackTrace();
33     }
34 }
35 }
36 }
37

```

【案例 2】finally 演示



```

TryCatchDemo.java X
1 package corejava.day08.ch03;
2 /**经典笔试题目：
3  * 如下程序执行结果是什么？
4  * A 0,0,5   B 1,0,1   C 4,0,5   D 4,4,4
5  */
6 public class TryCatchDemo {
7     public static void main(String[] args) {
8         System.out.println(
9             test(null)+" "+test("0")+" "+test("")); //4,4,4
10    }
11    public static int test(String str){
12        try {
13            //会出现空指针异常
14            return str.charAt(0)-'0';
15        } catch (NullPointerException e) {
16            //打印出异常
17            e.printStackTrace(); return 1;
18        } catch (RuntimeException e){
19            e.printStackTrace(); return 2;
20        } catch (Exception e){
21            e.printStackTrace(); return 3;
22        } finally {
23            return 4;
24        }
25    }
26 }

```

```

Console
<terminated> TryCatchDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 7, 2011 10:
java.lang.NullPointerException
    at corejava.day08.ch03.TryCatchDemo.test(TryCatchDe
    at corejava.day08.ch03.TryCatchDemo.main(TryCatchDe
java.lang.StringIndexOutOfBoundsException: String index out
    at java.lang.String.charAt(String.java:687)
    at corejava.day08.ch03.TryCatchDemo.test(TryCatchDe
    at corejava.day08.ch03.TryCatchDemo.main(TryCatchDe
4, 4, 4
    
```

注：

- ✓ 本案例在语法上允许（只是警告），但工作中不会出现，仅作为演示
 - 案例 return 了 2 次（catch 一次，finally 一次）
- ✓ **finally 永远会被执行**
- ✓ 捕获（catch）异常有顺序，异常“由小到大”，否则会出编译错误
- ✓ **NullPointerException** 空指针异常
- ✓ **StringIndexOutOfBoundsException** 字符串下标越界

2.2. 异常的分类 *

Throwable

--Error	是系统不可恢复的错误，JVM 发生的错误
--OutOfMemoryError	堆内存溢出
--StackOverflowError	栈内存溢出
--Exception	程序可以检查处理的异常,常见的异常继承根
--java.text.ParseException format	解析对象时候发生
如：Date d = dateformat.parse("2010-5-5");	
--RuntimeException	非检查异常，Javac 忽略对这类异常的语法检查
--IllegalArgumentException	
--NullPointerException *	
--ArrayIndexOutOfBoundsException *	
--ClassCastException *	
--NumberFormatException * Integer.parseInt(S)	

关于异常的分类：

- 1) Error 是 JVM（Java 虚拟机）中出现的不可恢复的错误
- 2) Exception 是类（Class）发生的异常
 - 检查异常 编译期发生
 - 非检查异常（RuntimeException） 运行时发生

3) 请记住常见的几种 RuntimeException

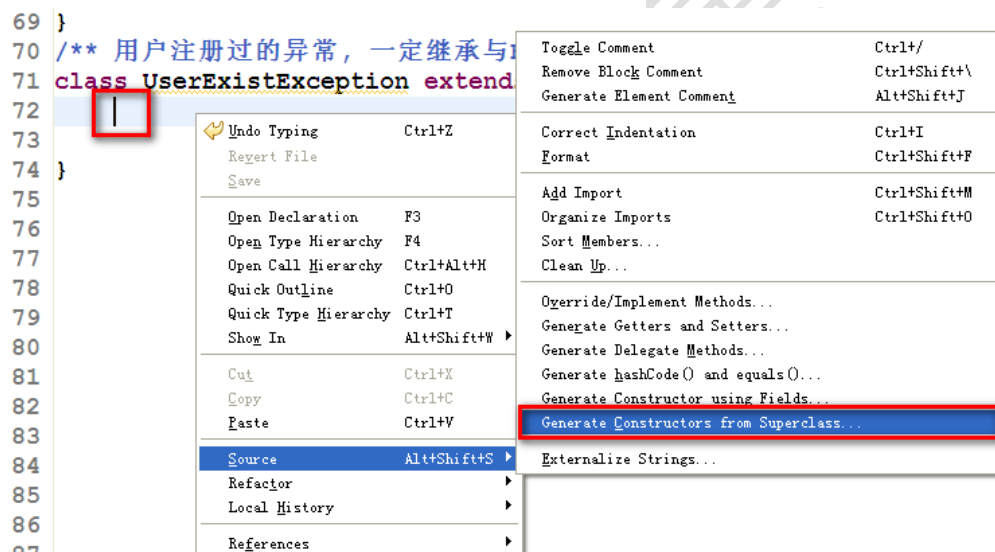
2.3. 自定义异常 **

软件中会大量使用自定义异常，一般从 Exception 继承异常类命名要有实际意义，一般都手工继承父类的构造器。

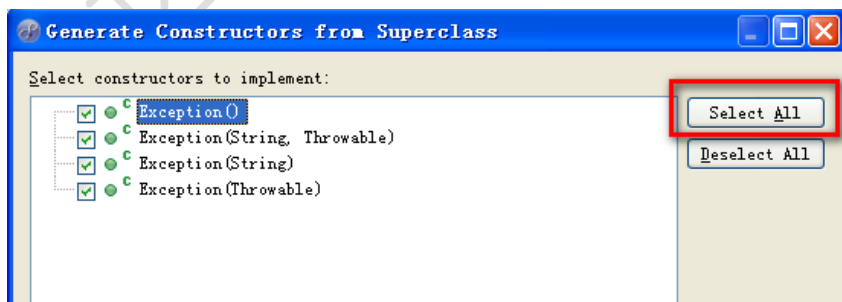
【案例 1】 请参照“用户登录及异常处理”的案例

【案例 2】 使用 Eclipse 工具从父类创建构造器

第 1 步：在类中点击【右键】Source -> Generate Constructors from Superclass



第 2 步



执行结果

```

70 /** 用户注册过的异常，一定继承与Exception类 */
71 class UserExistException extends Exception{
72
73     public UserExistException() {
74         super();
75         // TODO Auto-generated constructor stub
76     }
77
78     public UserExistException(String message, Throwable
79         super(message, cause);
80         // TODO Auto-generated constructor stub
81     }
82
83     public UserExistException(String message) {
84         super(message);
85         // TODO Auto-generated constructor stub
86     }

```