

复旦大学计算机科学技术学院



编程方法与技术

8.1. 接口、泛型复习

周扬帆

2021-2022第一学期

接口

```
interface Comparable {  
    String name = "Hello, World";  
    boolean compare(Comparable b);  
}  
  
class MyInteger implements Comparable {  
    public boolean compare(MyInteger b) {  
        ...  
    }  
}
```

- ❑ 把需实现的方法和共有常量定义在接口里
 - 接口的变量默认为 **public final static** : 常量
 - 接口的方法默认为 **public abstract**
- ❑ 实现接口的类，必须定义接口的方法
 - 接口类似只有 **abstract** 方法和常量的类

接口

- ❑ 和class一样, 有public和default两种接口
- ❑ 和class一样, 可以有内部接口
 - 但没有局部接口
- ❑ 同样可以用匿名类实现接口

```
interface Printable {  
    void print();  
}
```

```
new Printable() {  
    public void print() { ...  
    }  
}.print();
```

- ❑ 接口可以extends其他接口, 组成新接口
 - 可以extends多个其他接口
- ❑ 一个类可以实现多个接口

接口的默认方法

- Java 1.8让接口可以写具体实现
- 二义问题怎么解决?

```
interface OldInterface {  
    void a();  
    default void b() {  
        System.out.println("Hello!");  
    }  
}  
  
interface NewInterface {  
    void a();  
    default void b() {  
        System.out.println("Hello Again!");  
    }  
}  
  
public class Test implements OldInterface, NewInterface {  
    //...  
}
```

需要定义b()的实现! → 如何调用默认定义?

显式调用

OldInterface.**super**.b();
NewInterface.**super**.b();

Java泛型类

□ 类的泛型（模板类）

- 类的实现中，把某些用到的数据类型抽象为泛型（模板）
- 在类**创建**的时候才指定类型
- 此模板可以接受合适类型的对象
- 目的？

→ 写代码方便

→ 方便理解

→ 方便编译器查错

```
public class Print <T> {  
    public void print(T [] a) {  
        for(T i: a) {  
            System.out.print(i + " ");  
        }  
        System.out.println();  
    }  
    public static void main(String args[]) {  
        Character [] ca = {'1', '2'};  
        new Print<Character>().print(ca);  
    }  
}
```

Java泛型类

```
class Print <T> {
```

```
    public void print(T [] a) {
```

```
        for(T i: a) {
```

```
            System.out.print(i + " ");
```

```
        }
```

```
        System.out.println();
```

```
    }
```

```
}
```

```
public class Test {
```

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

```
        Character [] ca = {'1', '2'};
```

```
        Print<Character> printer = new Print <Character>();
```

```
        printer.print(ca);
```

```
    }
```

```
}
```

定义这个类里有一种类型叫做T,
具体是什么, new的时候才指定

把T作为一种类型,

new的时候指定T是什么

所以只能用在实例化方法中

Print <Object>不是Print <String>的父类

Java泛型

```
class Data <T, S> {  
    private T var1;  
    private S var2;  
    ...  
}
```

← 可定义多个泛型

```
Data <String, String> data1 = new Data <String, String> ();  
Data <String, Integer> data2 = new Data <String, Integer > ();
```

} 不同类型实例

Java泛型接口

```
interface DatumInterface <S> {  
    public S getVar();  
    public void setVar(S var);  
}
```

和泛型类相似

```
class Datum <T> implements DatumInterface <T/S??> {  
    private T var;  
    public T getVar() {  
        return var;  
    }  
    public void setVar(T var2) {  
        var = var2;  
    }  
}
```

支持泛型实现

Java泛型方法

```
public class Test {  
    public <T> void print(T a) {  
        System.out.println(a);  
    }  
    public static void main(String args[]) {  
        Test test= new Test();  
        test.print("Hello");  
        test.print(12);  
    }  
}
```

定义这个方法有一种类型叫做T，具体是什么，调用的时候指定

在调用时，指定方法里的T为String

在调用时，指定方法里的T为Integer

不能创建(new)泛型变量和数组

没有泛型对象数组

```
class Datum <T> {  
     private T a1 [] = new T [10];  
    private T a2 = new T();  
}
```

```
class Datum <T> {  
    ...  
}
```

```
Datum <String> a [] = new Datum <String> [10]; 
```

为了代码安全而禁止

编译器不知道如何初始化
万一T需要带参数初始化?

- Fact: Java泛型只在编译时处理，运行时是擦除的
 - 为了兼容性
 - 为了方便
- 数组有可能运行时被装进去不符合泛型要求的对象

复旦大学计算机科学技术学院



编程方法与技术

8.2. Java的this引用

周扬帆

2021-2022第一学期

this关键字

□ 对象自己的引用: this

```
ClassA objectA = new ClassA();  
classA.method();
```

...

```
class ClassA {  
    public void method() {  
        ClassB objectB = new ClassB(this);  
        ...  
    }  
}
```

当前对象的引用

```
}  
class ClassB {  
    ClassA objectA;  
    ClassB(ClassA objectA) {  
        this.objectA = objectA;  
    }  
}
```

默认作用域: 本block内
→ 要访问本类的同名成员对象,
用this.XXX

- 用于把引用传给其他对象
- 用于作用域控制
- ...

this关键字

```
ClassA objectA = new ClassA();
```

```
...
```

```
class ClassA {  
    public ClassA() {  
        this(1);  
        ...  
    }  
    public ClassA(int i) {  
        ...  
    }  
}
```

调用另一个构造方法

1. 第一行

2. 只能调用一次

- 构造方法里调用另一个构造方法

- ...

this关键字

```
ClassA objectA = new ClassA();
```

```
...
```

```
class ClassA {  
    public ClassA init () {  
        ...  
        return this;  
    }  
    public ClassA connect () {  
        ...  
        return this;  
    }  
    public ClassA disconnect () {  
        ...  
        return this;  
    }  
    public ClassA setTimeout(int) { ...}  
    public ClassA setLogLevel(int) { ...}  
}
```

```
objectA.init()  
    .connect()  
    .disconnect();
```

```
objectA.init()  
    .setTimeout(1)  
    .connect()  
    .disconnect();
```

```
objectA.init()  
    .setTimeout(1)  
    .setLogLevel(0)  
    .connect()  
    .disconnect();
```

```
objectA.init()  
    .setLogLevel(0)  
    .setTimeout(1)  
    .connect()  
    .disconnect();
```

■ 实现Fluent Interface

this关键字

□ 内部类访问外部类的引用

```
ClassA objectA = new ClassA();  
classA.method();  
...  
class ClassA {  
    public void method() {  
        ClassB objectB = new ClassB();  
        ...  
    }  
    class ClassB {  
        ClassA objectA = ClassA.this;  
    }  
}
```

□ 外部类的外部类的引用?

- 一样: 类名.this

□ 静态内部类?

复旦大学计算机科学技术学院



编程方法与技术

8.3. JavaScript的this引用

周扬帆


2021-2022第一学期

函数的this

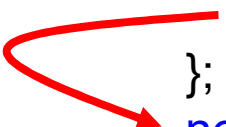
□ this引用的指向

```
value = 0;  
function func() {  
  this.value = 1;  
};  
func();  
console.log(value);
```

global

A red arrow originates from the word 'this' in the function definition and points to the word 'global'.

```
value = 0;  
function func() {  
  this.value = 1;  
};  
new func();  
console.log(value);
```

A red arrow originates from the word 'this' in the function definition and points to the 'new' keyword in the function call.


函数的this

- **this引用的指向和函数的调用方式有关**
- **函数调用方式**
 - 普通的函数调用
 - 对象的方法调用
 - 构造函数调用
 - apply/call调用

普通的函数调用

□ this指向全局对象global

```
value = 0;  
function func() {  
  this.value = 1;  
};  
func();  
console.log(value);
```

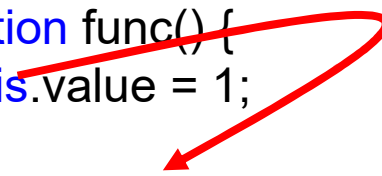


A red arrow points from the `this` property access inside the `func()` function to the word `global`, indicating that `this` refers to the global object.

对象的方法调用

□ this指针指向对象本身

```
value = 0;  
function func() {  
    this.value = 1;  
};  
var obj = new Object();  
obj.value = 2;  
obj.fn = func;  
obj.fn();  
console.log(value);  
console.log(obj.value);
```



0
1

小练习

```
value = 0;  
function func() {  
  this.value = 1;  
  function func2() {  
    this.value = 3;  
  }  
  return func2;  
};  
var obj = new Object();  
obj.value = 2;  
obj.fn = func;  
var func2 = obj.fn();  
func2();
```



```
obj.fn2 = func2;  
obj.fn2();
```

3
3


```
console.log(value);  
console.log(obj.value);
```

3
1

构造函数调用

□ this指针指向new出的新对象

```
value = 0;  
function func() {  
  this.value = 1;  
};  
var obj = new func();  
console.log(obj.value);  
console.log(value);
```



1
0

apply/call调用

□ this指针指向绑定的对象

```
value = 0;  
function func() {  
    this.value = 1;  
};  
var obj = new Object();  
obj.value = 2;  
func.apply(obj);  
console.log(obj.value);  
console.log(value);
```

□ 函数对象的**apply(obj, args)**方法

- 将函数绑定在参数obj指定的上下文执行
- args是参数数组
- this当然就是obj了

□ 函数对象的**call(obj, arg1, arg2, ...)**方法

- 将函数绑定在参数obj指定的上下文执行
- 第二个参数开始是参数列表
- this当然就是obj了

apply/call 用处：对象继承方法

□ 通过构造函数实现继承的方法

匿名

```
1 function Animal(name, sex) {  
2     this.name = name;  
3     this.sex = sex;  
4 }  
5  
6 Animal.prototype.getName = function () {  
7     return this.name;  
8 }  
9  
10 var cat = new Animal('white', 'male');  
11 cat.getName(); // white  
12  
13 function People(name, sex) {  
14     Animal.call(this, name, sex);  
15 }  
16  
17 People.prototype = new Animal('people', null);  
18  
19 var Chris = new People('Chris', 'male');  
20 Chris.getName(); // Chris
```


复旦大学计算机科学技术学院



编程方法与技术

8.4. Java文件操作

周扬帆

2021-2022第一学期

Java File类

□ 源文件

- test.java: <http://y-droid.com/test.java>
- main方法里
 - test1()
 - test2()
 - ...

Java File类

□ File类

- `import java.io.File;`
- 用来与操作系统交互，实现各种文件操作
 - 删除、重命名等等

□ 构造方法

- `File file = new File(String pathName);`

`File file = new File(filePath+ "\\\" + fileName);`

`File file = new File(filePath+ "/" + fileName);`

`File file = new File(filePath+ File.separator + fileName);`

另： *File.pathSeparator*: 分开两个路径名，如windows的 ; 符号

Java File类

□ File类判断方法： boolean返回值

- canExecute(): 判断文件是否可执行
- canRead(): 判断文件是否可读
- canWrite(): 判断文件是否可写
- exists(): 判断文件是否存在
- isDirectory(): 判断是否是目录
- isFile(): 判断是否是普通文件
- isHidden(): 判断是否隐藏
- isAbsolute(): 判断文件路径是否是绝对路径

例程里，main里执行test1()方法

Java File类

□ File类信息获取方法

- String getName()
 - String getPath()
 - String getAbsolutePath()
 - String getParent()
 - long lastModified()
 - long length()
 - File[] listFiles()
 - String[] list()
- 路径名: 路径+文件名
- 路径(没有文件名)
- long型时间
- 列出目录下的文件和目录

例程里, main里执行test2()方法

Java File类

□ File类文件控制方法

- boolean createNewFile()
 - boolean renameTo(File f)
 - boolean delete()
 - boolean mkdir()
 - boolean mkdirs()
- 创建新文件
- } 操作文件、目录
- } 创建新目录

设想有C:/Documents/

mkdir: C:/Documents/abc/d 失败

mkdirs: C:/Documents/abc/d 成功, 连abc都创建好

Java File类

□ 问题

■ 如何遍历一个目录下的所有文件

```
void findFilebyName (File filePath, String fileName) {  
    for (File fileDir : filePath.listFiles()) {  
        if (fileDir.isDirectory()) {  
            findFilebyName (fileDir, fileName);  
        }  
        else if (fileDir.isFile()) {  
            if (fileDir.getName().contains(fileName)) {  
                System.out.println(fileDir.getName());  
            }  
        }  
    }  
}
```

例程里, main里执行test3()方法

复旦大学计算机科学技术学院



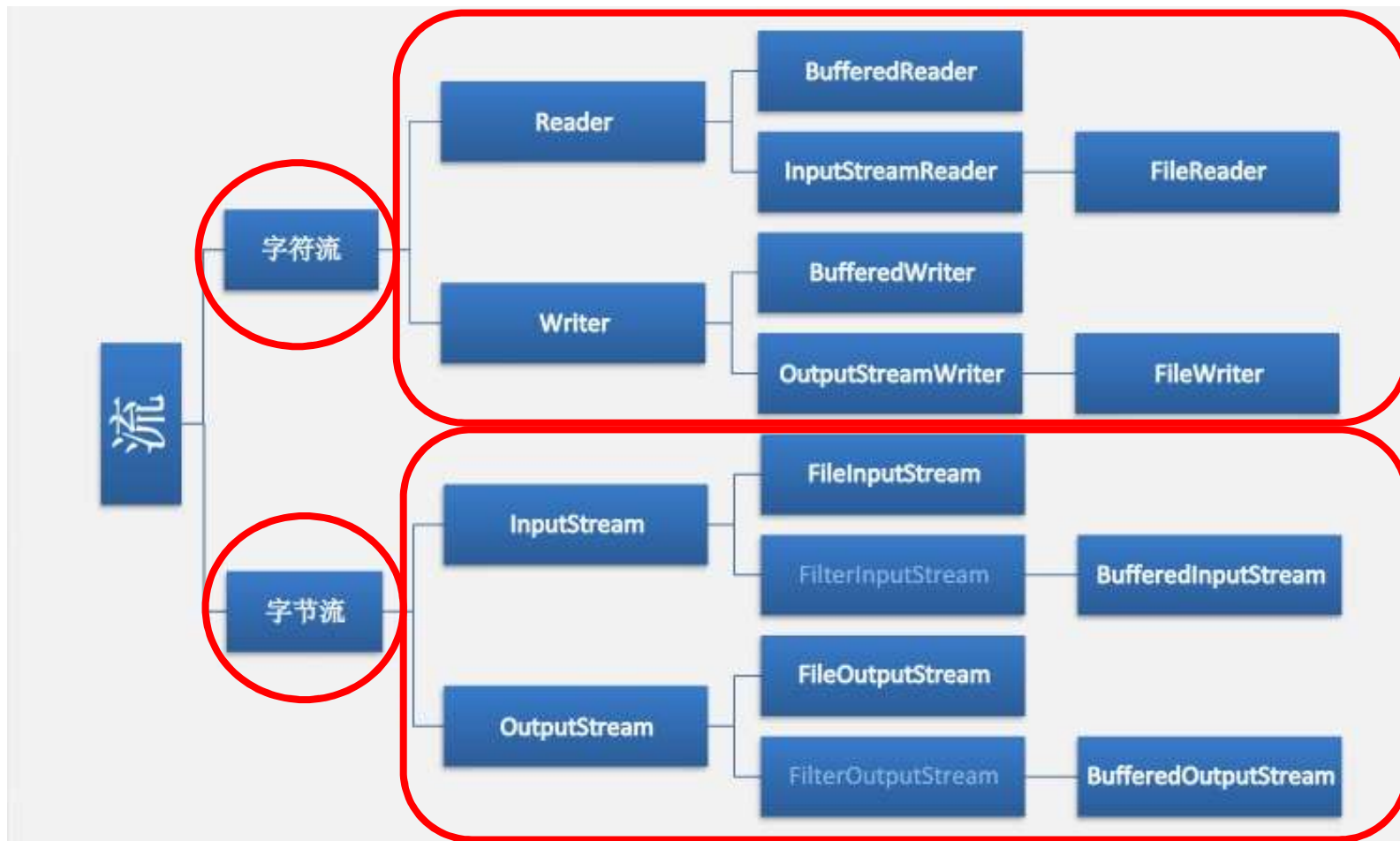
编程方法与技术

8.5. Java流I/O

周扬帆

2021-2022第一学期

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java字节流

□ FileInputStream类/FileOutputStream类

■ 以字节的形式从流中读/写数据

■ 构造方法

FileInputStream(File file)

FileOutputStream(File file)

FileInputStream(String name)

FileOutputStream(String name)

■ 用法示例

```
FileInputStream fin = new FileInputStream(srcFile);
FileOutputStream fout = new FileOutputStream(destFile);
byte[] bytes = new byte[1024];
while (fin.read(bytes) != -1) {
    fout.write(bytes);
    fout.flush();
}
fin.close();
fout.close();
```

例程里，main里执行test4()方法

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java字节流

□ BufferedInputStream类/ BufferedOutputStream类

■ 缓冲的读/写数据

■ 构造方法

BufferedInputStream(FileInputStream fis)

BufferedOutputStream(FileOutputStream fos)

■ 用法示例

```
BufferedInputStream bis = new BufferedInputStream(new FileInputStream(srcFile));
BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(destFile));
byte[] bytes = new byte[16];
while (true) {
    int size = 0;
    if((size = bis.read(bytes)) <= 0) {
        break;
    }
    bos.write(bytes, 0, size);
    bos.flush();
}
bis.close();
bos.close();
```

哪个性能增益大: 大大块读, 还是碎碎读

例程里, main里执行test5()方法

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java文件字符流

□ InputStreamReader类

- 字节流 → 字符流
- 编码可指定，默认为平台默认字符编码
- 构造方法
 - 字节流，编码

```
InputStreamReader isr = new InputStreamReader(InputStream in);
```

```
InputStreamReader isr = new InputStreamReader(InputStream in, String charsetName);
```

□ OutputStreamWriter类

- 类似

Java文件字符流

□ InputStreamReader类/ OutputStreamWriter类

■ 使用示例

```
InputStreamReader isr= new InputStreamReader(new FileInputStream(srcFile));
OutputStreamWriter osw= new OutputStreamWriter (new FileOutputStream(destFile));
char[] bytes = new char[1024];
while (isr.read(bytes) != -1) {
    osw.write(bytes);
    osw.flush();
}
isr.close();
osw.close();
```

例程里，main里执行test6()方法

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java字符流

□ FileReader类/FileWriter类

■ 以字符的形式从文件中读/写数据

■ 构造方法

FileReader(File file)

FileReader(String name)

FileWriter(File file)

FileWriter(String name)

■ 用法示例

```
fr = new FileReader(srcFile);
fw = new FileWriter(destFile);
char buf [] = new char [1024];
int len = 0;
while((len = fr.read(buf)) != -1){
    fw.write(buf, 0, len);
}
fr.close();
fw.close();
```

例程里，main里执行test7()方法

Java流的继承关系



字节(byte)流：处理声音或者图片等二进制的数据的流
字符(char)流：处理文本数据（如txt文件）的流

Java字符流

□ BufferedReader类/BufferedWriter类

■ 带缓冲的读写

■ 构造方法

BufferedReader(InputStreamReader isr)

BufferedWriter(OutputStreamWriter osw)

■ 用法示例

```
bufReader = new BufferedReader(new InputStreamReader(new FileInputStream(sFile)));
bufWriter = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(dFile)));
String input = null;
while((input = bufReader.readLine()) != null)
{
    bufWriter.write(input);
    bufWriter.newLine();
}
bufReader.close();
bufWriter.close();
```

例程里，main里执行test8()方法

思考

- ❑ 理解字符流和字节流的区别
- ❑ 思考字符流和字节流是否可以互相转换
- ❑ 思考流（特别是输入流）用完是否需要关闭，思考流之间互相调用（输入流连输出流、输入流嵌套封装）如何关闭
- ❑ 思考字符编码的意义，理解其实现

复旦大学计算机科学技术学院



编程方法与技术

8.6. Java异常

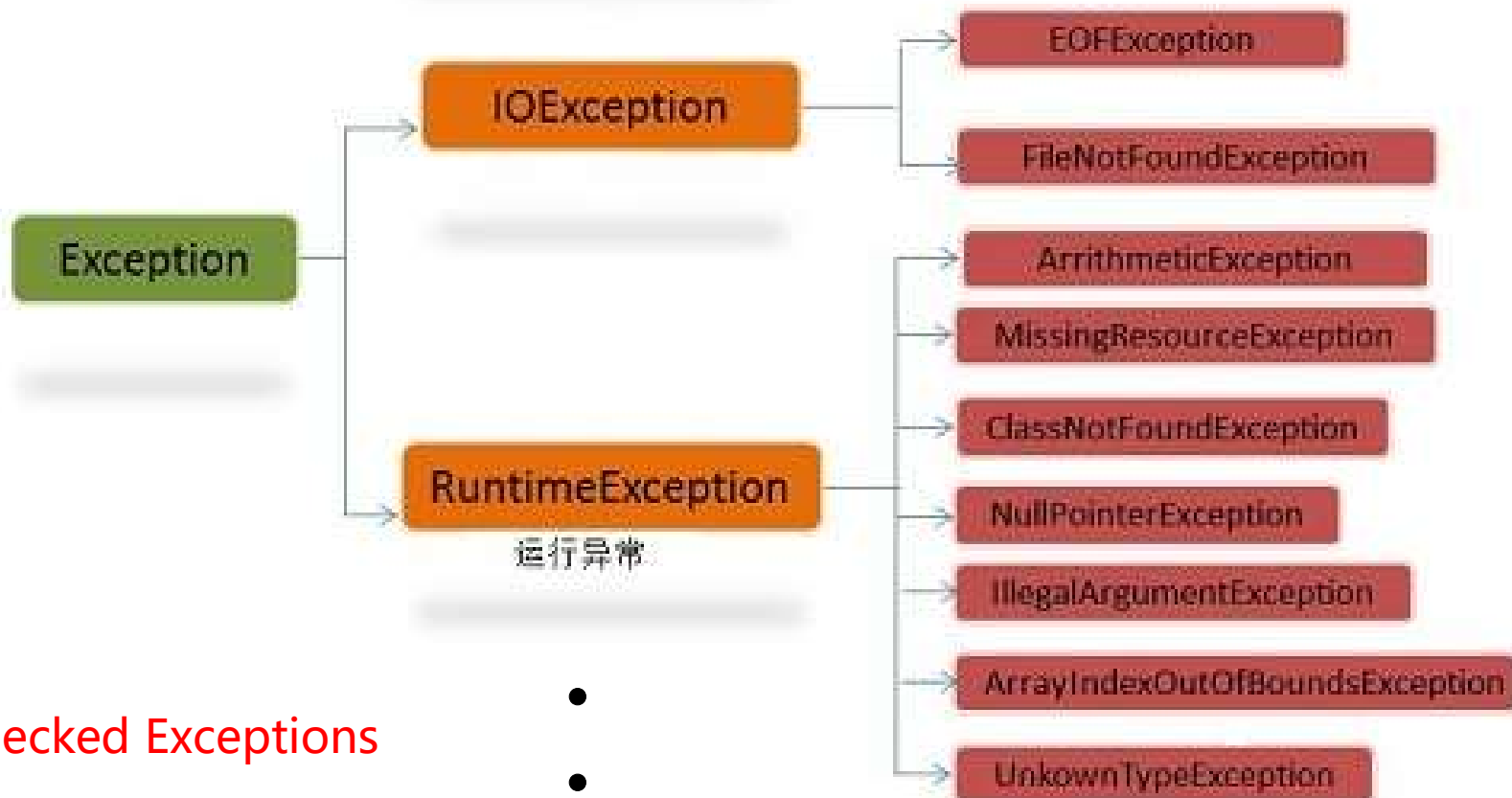
周扬帆

2021-2022第一学期

Java异常

- **程序运行时会遇到很多异常**
 - 文件找不到
 - 读写文件时发生IO错误
 - 网络连接失败
 - 参数非法
 - 空引用
 - 数组越界
- **Java是一种安全、鲁棒的语言**
 - 通过异常的捕捉处理、防止程序崩溃

Java异常



checked Exceptions

unchecked Exception

-
-
-

Java异常

❑ Checked exceptions

- 提供机制，强制程序员写异常处理
- 什么是需要强制的？

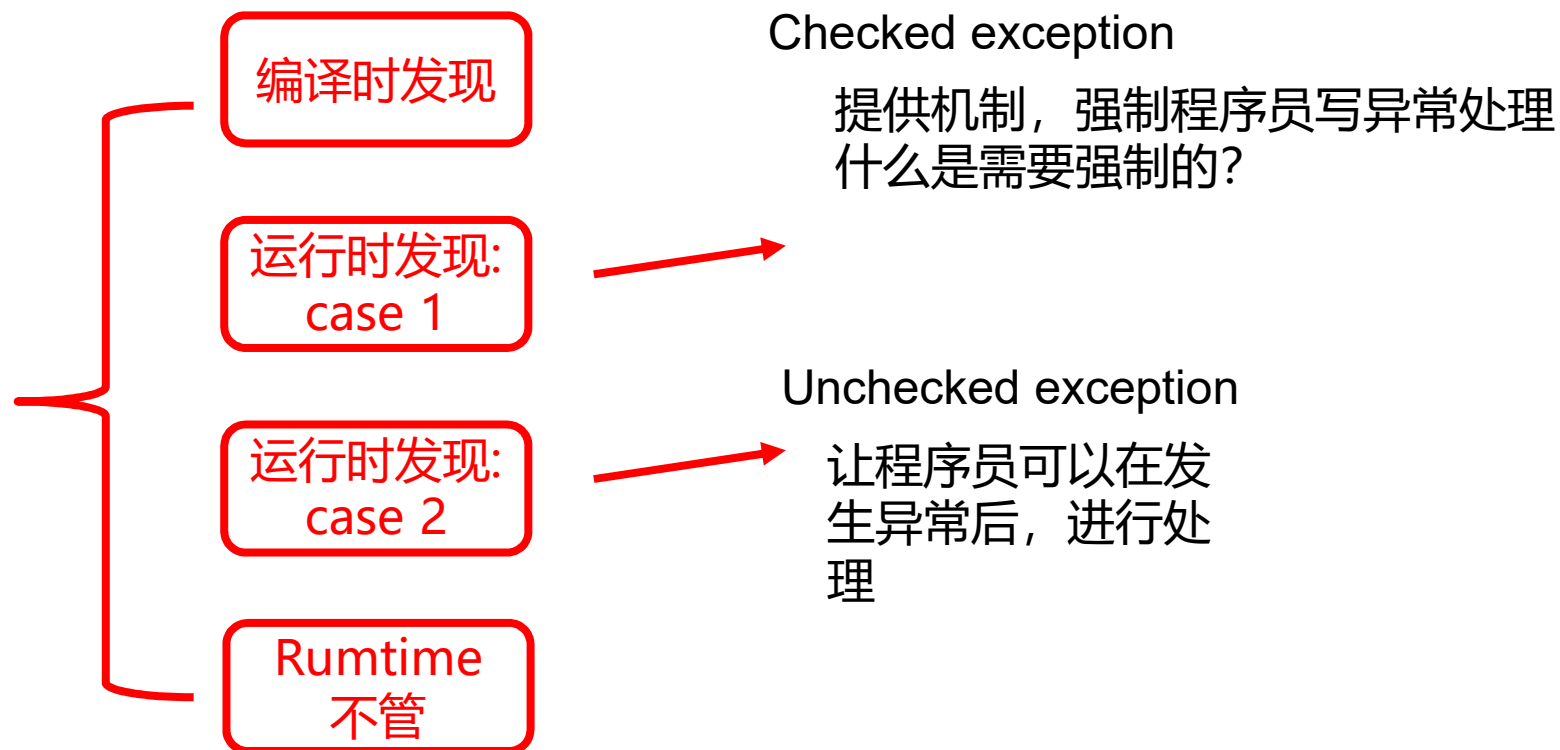
程序员

❑ Unchecked exceptions

环境

- 提供机制，让程序员可以在发生异常后，进行处理

Safety威胁的处理



Java异常

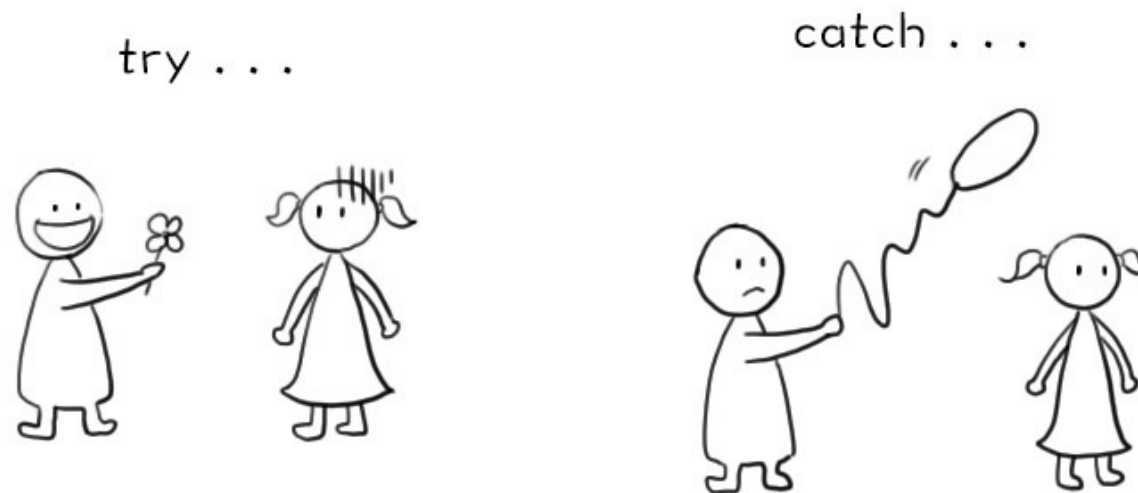
- ❑ 异常的捕捉处理
- ❑ 异常的抛出
- ❑ 自定义异常
- ❑ 异常处理过程

Java异常

- 异常的捕捉处理
- 异常的抛出
- 自定义异常
- 异常处理过程

Java异常的捕捉

□ try/catch/finally



Java异常的捕捉

❑ try/catch/finally

❑ 用法

```
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
finally {  
    //最后需要执行的语句  
}
```

异常捕捉示例

```
public static void copy(String sFile, String dFile) {  
    File srcFile = new File(sFile);  
    File destFile = new File(dFile);  
    FileInputStream fin = null;  
    FileOutputStream fout = null;  
    try {  
        fin = new FileInputStream(srcFile);  
        if (!destFile.exists()) {  
            destFile.createNewFile();  
        }  
        fout = new FileOutputStream(destFile);  
        byte[] bytes = new byte[1024];  
        while (fin.read(bytes) != -1) {  
            fout.write(bytes);  
            fout.flush();  
        }  
    } catch (FileNotFoundException e) {  
        System.out.println("Can find the source file: " + sFile);  
    } catch (IOException e) {  
        System.out.println("IO Exception caught.");  
    }  
    ...  
}
```

异常捕捉示例

```
public static void copy(String sFile, String dFile) {  
    ...  
    finally {  
        try {  
            if (fin != null) {  
                fin.close();  
            }  
        } catch (IOException e) {  
            System.out.println("Cannot close: " + sFile);  
        }  
        try {  
            if (fout != null) {  
                fout.close();  
            }  
        } catch (IOException e) {  
            System.out.println("Cannot close: " + dFile);  
        }  
    }  
}
```


Java异常

- 异常的捕捉处理
- 异常的抛出处理
- 自定义异常
- 异常处理过程

Java异常的抛出

❑ 方法可以不处理异常，而将异常抛出给调用者

```
public static void copy(String sFile, String dFile) throws FileNotFoundException, IOException {  
    File srcFile = new File(sFile);  
    File destFile = new File(dFile);  
    FileInputStream fin = null;  
    FileOutputStream fout = null;  
    fin = new FileInputStream(srcFile);  
    if (!destFile.exists()) {  
        destFile.createNewFile();  
    }  
    fout = new FileOutputStream(destFile);  
    byte[] bytes = new byte[1024];  
    while (fin.read(bytes) != -1) {  
        fout.write(bytes);  
        fout.flush();  
    }  
}
```

异常捕捉示例

□ 调用者的处理1: try/catch/finally

```
...  
// src dest  
try {  
    copy(src, dest);  
} catch (FileNotFoundException e) {  
    System.out.println("Can find the source file: " + src);  
} catch (IOException e) {  
    System.out.println("IO Exception caught.");  
}
```

□ 调用者的处理2: throws给调用者的调用者

- 直到main方法
- main方法也能throws → 给谁

Java异常

- 异常的捕捉处理
- 异常的抛出处理
- 自定义异常
- 异常处理过程

自定义异常

- ❑ 异常也是一种Java类
- ❑ 可自定义自己的异常类

```
public class MyException extends Exception {  
    String message;  
    public MyException(String exceptionMessage) {  
        message = exceptionMessage;  
    }  
    public String getMessage() {  
        return message;  
    }  
}
```

自定义异常

□ 编程实现自定义异常的抛出

```
void testException(Integer i) throws MyException {  
    if(i == null) {  
        MyException e = new MyException("Input i: null");  
        throw e;  
    }  
    System.out.println("Input i: " + i);  
}
```

Java异常

- ❑ 异常的捕捉处理
- ❑ 异常的抛出处理
- ❑ 自定义异常
- ❑ 异常处理过程

Java异常的捕捉

```
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
finally {  
    //最后需要执行的语句  
}
```

遇到异常，按顺序查下来



永远会执行！不管有没有异常

Java异常的捕捉

```
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
finally {  
    //最后需要执行的语句  
}
```

多态特性:

**如果捕捉到子类的异常
会进入父类的catch**

**因此父类的catch永远不能
在子类的catch前面, 否则报错**

Java异常的捕捉

```
try {  
    //可能会抛出异常的语句  
    return i = 0;  
}  
catch (XXException e) {  
    //异常处理的语句  
}
```

```
catch (XXException e) {  
    //异常处理的语句  
}
```

```
finally {  
    return i = i + 1;  
    //最后需要执行的语句  
}
```

永远会执行！ 不管有没有异常

Java异常的捕捉：讨论

```
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}
```

比较

```
catch (Exception e) {  
    //异常处理的语句  
}
```

Java异常的捕捉：讨论

```
try {  
    //其他代码  
    //可能会抛出异常的语句  
    //其他代码  
}  
catch (XXException e) {  
    //异常处理的语句  
}
```

比较

```
//其他代码  
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
//其他代码
```

Java异常的捕捉：讨论

```
if (a == null) {  
    throw new XXException(...);  
}  
...  
try...catch...
```

比较

```
if (a == null) {  
    return -1;  
}  
...  
if(ret == -1) {  
}
```

Java异常的捕捉：讨论

```
try {  
    everything  
}  
catch (Exception e) {  
    //do nothing  
}
```

思考

- ❑ 自行了解、理解Error和Exception
- ❑ try-catch-finally-return执行顺序
- ❑ 思考、理解NullPointerException和ArrayIndexOutOfBoundsException
- ❑ 理解“异常链”
- ❑ 如果在try或者finally的代码块中调用System.exit(), finally里的代码会执行么?

复旦大学计算机科学技术学院



编程方法与技术

8.7. JavaScript异常

周扬帆

2021-2022第一学期

异常

□ 变量未赋值

```
var value;  
console.log(value);
```

value的值是undefined, 程序无异常

```
var value;  
  
//...  
if (value === undefined) {  
    console.log("value undefined");  
}
```

异常

□ 变量不存在

```
var value;  
console.log(value1);
```

```
ReferenceError: value1 is not defined  
    at Object.<anonymous> (C:\Users\simple\WebstormProjects\untitled\test2.js:22:28)  
    at Module._compile (module.js:570:32)  
    at Object.Module._extensions..js (module.js:579:10)  
    at Module.load (module.js:487:32)  
    at tryModuleLoad (module.js:446:12)  
    at Function.Module._load (module.js:438:3)  
    at Module.runMain (module.js:604:10)  
    at run (bootstrap node.js:383:7)  
    at startup (bootstrap node.js:149:9)  
    at bootstrap node.js:496:3
```

□ 异常为什么需要编程处理？

- 更友好
- 容错处理
- ...

try/catch/finally

```
try {  
    var value;  
    console.log(value1);  
}  
catch (err) {  
    var txt = "Error description: " + err.message;  
    console.log(txt);  
}
```

Error description: value1 is not defined

try/catch/finally

```
try {  
    var value;  
    console.log(value1);  
}  
catch (err) {  
    var txt = "Error description: " + err.message;  
    console.log(txt);  
}  
finally {  
    console.log('OK');  
}
```

先try, 有异常就catch, 不管有没有异常, 都finally

finally的作用?

try/catch/finally

```
var func = function() {  
  try {  
    var value;  
    //...  
    console.log(value);  
    return 0;  
  }  
  catch (err) {  
    var txt = "Error description: " + err.message;  
    console.log(txt);  
  }  
  finally {  
    console.log('OK');  
    return 1;  
  }  
}  
console.log(func());
```

0/1 ?

异常throw

□ 自定义抛出异常信息

```
try {  
    console.log('Step 1');  
    throw 'My Exception';  
    console.log('Step 2');  
}  
catch (err) {  
    console.log(err);  
}
```

```
try {  
    console.log('Step 1');  
    throw new Error('My Exception');  
    console.log('Step 2');  
}  
catch (err) {  
    console.log(err);  
    //console.log(err.getMessage);  
}
```

复旦大学计算机科学技术学院



编程方法与技术

8.8.课程练习

周扬帆

2021-2022第一学期

Java文件与异常处理

- 遍历给定路径，查找所有的java文件
- 对于每一个java文件 (调用processJavaFile方法)
 - 找到文件中一行注释，格式为 "//todo:"
 - 如果存在： 输出文件路径名 (path + name)
 - 如果不存在，抛出**自定义的**Exception，定义一个错误信息"todo not found"
 - 抛出所有异常给调用者
- 调用processJavaFile的异常捕捉处理
 - 如果是自定义Exception，输出错误信息，继续查找下一个文件
 - 如果是其他Exception，输出相应错误信息，并做相应的**合理处理**
- DDL: Nov-17-23:59 (下周三晚)