java语法总结

OOP

构造函数可以调用另一个构造函数，使用this

构造函数没显示给变量赋值，系统给一个默认值，数字是0，boolean为假，对象为null。

final关键字修饰对象，只是引用不能变，对象内容可以修改；

所有类都至少有一个构造函数，如果类没有声明构造函数，系统提供一个默认无参的构造函数，如果已经编写了一个构造函数还想要无参的构造函数，必须自己写一个无参构造函数。

System.out就是System类的静态常量，但是它能绕过Java的语言机制，将out设置到不同的流上，这是个例外。

将类放到包中，将包的声明作为源文件第一个声明

package com.test

class Test()

{

}

Test类就在com.test包中

使用jar将多个class文件制作到jar包内，类库就是这么做的；

包作用域：

public可以被所有其他类使用，private只能被声明他的类使用，啥也不加的能被同一个包下的所有类使用

源文件名必须和类名一样，一个文件只能有一个被声明为public

静态导入

import static java.lang.Math.\* 导入Math类的静态方法和静态变量

静态嵌套类和内部类

接口

public interface IntSeq {

boolean hasNext();

int next;

}

接口的方法默认是public的；

实现类必须将接口的方法声明为public的，否则报错；

如果一个类只实现部分接口方法，那它必须用abstract修饰声明；

可以将子类型不需要转换就赋值给父类型：

Parent p = new Child;

父类型赋值给子类型最好先用instanceof判断一下，然后使用转换后赋值

if (p instanceof Child) {

Child test = (Child)p;

}

继承接口

一个接口可以继承另外一个接口，来提供额外的方法，实现次接口必须将两个接口方法都实现；

一个类可以实现多个接口

public class FileSeq implements IntSeq, Closeable {

...

}

接口中任何变量都是自动是public static final,你无法在接口中拥有实例变量，接口指定行为，而不是状态；

继承

子类覆写父类方法注意函数名一致，函数参数的类型必须一致，不一致就是子类定义了一个新的方法，为了避免这种错误，方法前家注解@override,这个时候如果子类方法中把参数类型修改，编译器直接报错

覆盖一个方法，可以将返回值改为父类方法返回类型的子类型：

public Employee getSupervior();

@override public Manager getSupervior();

构造函数

public Manager(String name, double salary) {

super(name, salary);

bonus = 0;

}

super来调用父类的构造函数；如果子类中忘记调父类构造函数，父类必须有一个无参的，被隐式调用的构造函数。

动态查找

将子类实例变量赋值给一个父类变量，调用这个父类变量的方法是，会实际调用子类对应的方法。这么做可以将所有父类。注意这种情况下只能调用父类中存在的方法(子类覆写后就执行子类的)。如果要执行子类中存在的方法要这么做：

if (emp1 instanceof Manager) {

Manager mgr = (Manager)emp1;

mgr.setBonus(100);

}

final

一个方法声明为final的时候，子类不能覆写它；

抽象类

一个类中定义的方法用abstract修饰，称为抽象方法，对应的类叫抽象类，抽象类不能被实例化。抽象类可以有实现好的非抽象方法。继承了抽象类的方法如果没有实现抽象方法，那它也必须定义成抽象的。

抽象类和接口的区别：抽象类可以有实例变量和构造函数，例如

public abstract Person {

private String name; 实例变量；

public Person(String name) {this.name = name;}

public abstract int getId();

}

protected:

允许同一个包内的访问

集合

遍历map:

<http://www.runoob.com/java/java-collections.html>

泛型

泛型通配符

public static void getData(List<?> data) {

System.out.println("data :" + data.get(0));

}

List<?> data ？代表具体的类型

泛型通配符上限

public static void getUperNumber(List<? extends Number> data) {

System.out.println("data :" + data.get(0));

}

List<? extends Number> 表示泛型值只接受Number及其子类型

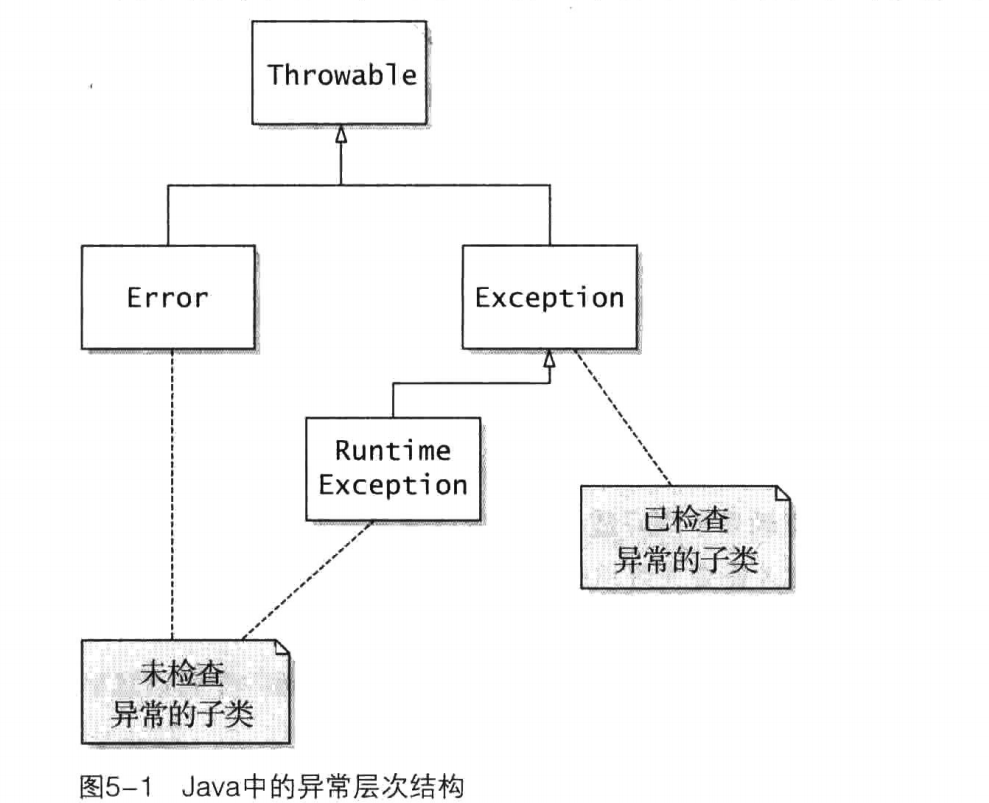
泛型通配符下限

List<? super Number> 表示泛型值只接受Number及其三层父类型

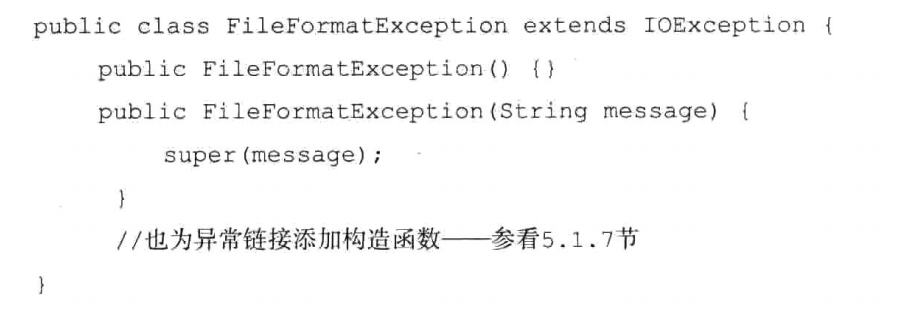
泛型类

泛型方法

异常



实现自己的异常时，最好提供两个构造函数，一个不带参数，一个带消息字符串参数



异常处理的黄金法则：“早抛出，后捕获”；

覆盖一个方法时，子类不能抛出比父类声明中还要多的已检查异常，否则编译器报错；

throw和throws区别：

参考：<http://blog.csdn.net/luoweifu/article/details/10721543>

1、throws出现在方法函数头；而throw出现在函数体。

2、throws表示出现异常的一种可能性，并不一定会发生这些异常；throw则是抛出了异常，执行throw则一定抛出了某种异常对象。

3、两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

好的编程习惯：

1.在写程序时，对可能会出现异常的部分通常要用try{...}catch{...}去捕捉它并对它进行处理；

2.用try{...}catch{...}捕捉了异常之后一定要对在catch{...}中对其进行处理，那怕是最简单的一句输出语句，或栈输入e.printStackTrace();

3.如果是捕捉IO输入输出流中的异常，一定要在try{...}catch{...}后加finally{...}把输入输出流关闭；

4.如果在函数体内用throw抛出了某种异常，最好要在函数名中加throws抛异常声明，然后交给调用它的上层函数进行处理。

JVM原理

JVM调优

Servlet原理

Spring