# JAVA面向对象

## java对象的初始化顺序

先初始化对象的属性，再进行构造函数的调用。

## static修饰的属性/方法

static修饰的属性是属于整个类的，所有该类生成的对象的该属性都指向同一块内存。应该使用类名来调用static属性。

static所申明的变量和static修饰的方法 会首先被初始化。

static修饰的仅仅只会初始化一次。

## extends继承(单继承)

继承的初始化顺序，子类继承父类，当创建子类时，先创建父类对象，再创建子类对象。

创建父类时，默认调用不带参数的构造方法。若父类没有无参数构造方法，应在子类的构造函数使用super(xxx)来调用父类的构造方法。并且 super语句必须放在构造函数第一行执行。如果子类中有多个构造函数，均需要在这些构造函数中使用super(xxx)手动构造父类。

## abstract 抽象类/抽象方法

抽象方法存在的意义就是用来被子类覆盖。

有一个抽象方法的类就是抽象类，抽象类不能用来创建对象，抽象类存在的意义，就是用来被继承。抽象类等于为开发人员确定了相应的约束。

## Exception异常 异常全部都是类

异常分为运行时异常，和非运行时异常，运行异常JVM会自动处理，如果发现没有被处理，会自动抛出；非运行异常需要程序员手动处理，不会自动抛出，如不处理则会直接编译报错。

异常的标准写法，继承Exception，覆盖父类的构造方法。

代码抛出 异常对象 使用throw，方法抛出为 异常类型 使用 throws。

一般建议使用**RuntimeException**

异常的抛出和捕获都支持多态，但是如果捕获顺序先捕获了父类异常就无法再捕获子类异常了。

## Main函数的参数

Args用来存储运行时的参数，可以在运行时指定相应的参数，使用空格分隔参数。

## String内存分析

**String** 是一种字符串常量。定义**String**类型 变量时，会首先从 **String Pool** 中找是否有相等的常量，如果没有则会在 String Pool中新建常量，如果有，则直接指向**String Pool**中对应的值，所以此时2个String变量的指向为同一常量， **str1 == str2** ；当使用 **new String(“X”)；**时，会分配不同的内存，此时**str1 !=str2** ,但是其内存又指向相同的**String Pool**值。

## IO-File类

**File.**[**mkdir**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/File.html#mkdir())**()**创建文件夹，若指定的目录中的父节点不存在，不会创建。

**File.**[**mkdirs**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/File.html#mkdirs())**()** 创建文件夹，包括所有路径必须但不存在的父节点目录。

**File.**[**renameTo**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/File.html#renameTo(java.io.File))**(f)**将File文件移动到f(f不存在)，且名称更为f的名称。

**FilenameFilter** 文件过滤器 接口。

## (匿名)内部类的使用

由于类的初始化顺序 会先初始化static 所以static 方法无法访问内部类。

匿名内部类，在代码返回之前实现接口。只有一个方法会涉及到该类时建议使用。

## IO-FileStream(字节 节点流)

**int length=** [**FilterInputStream**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/FilterInputStream.html)**.**[**read**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/FilterInputStream.html#read(byte[]))**(byte[] b)**

文件输入流 将读取到的内容放入b 中,读取到的长度为length;

**FileOutputStream.write(buff, 0, length);**

文件输出流 将buff中从[ 0到length长度 ]的内容 输出。

## IO-BufferedStream(字节 过滤流) 装饰模式

**extends FilterInputStream**

[**BufferedInputStream**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/BufferedInputStream.html#BufferedInputStream(java.io.InputStream))**(**[**InputStream**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/InputStream.html) **in)**

**int length=** [**BufferedInputStream**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/BufferedInputStream.html#BufferedInputStream(java.io.InputStream))**.**[**read**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/FilterInputStream.html#read(byte[]))**(byte[] b)**

**BufferedOutputStream.write(buff, 0, length);**

**BufferedOutputStream.flush()**(将缓冲输出流中的 缓存 内容 输出 出去) 必须，关闭流时自动flush();

## IO-DataStream(字节 数据 过滤流) 装饰模式

**extends FilterInputStream**

**DataInputStream.**[**readInt**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/DataInputStream.html#readInt())**()** 可以将文件中的字节数据方式读取为int ,每4字节读取为一个int值。

**DataOutputStream.**[**writeInt**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/DataOutputStream.html#writeInt(int))**(int v)** 将int数据以 4字节值写出到输出流中。

还可以操作其他基本数据类型。

## IO-BufferedRead(字符 缓冲流)

使用字符流来读取 文件 内容， 和 写入 文件内容

**BufferedReader.**[**readLine**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/BufferedReader.html#readLine())**();** 读取一行文本，如果没有则为**null**；

使用字符输出流来 写入内容。

**new PrintWriter(new BufferedWriter(new FileWriter()))；**

**PrintWriter.**[**println**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/io/PrintWriter.html#println(java.lang.String))**(**[**String**](mk:@MSITStore:E:\API\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) **x);** 打印文本 并换行。

## IO-StreamReader(字节转字符 转换流)

转换流 将 一个字节流 转换为 字符流 **InputStreamReader**

**new BufferedReader(new InputStreamReader(new InputStream));**

## 多线程的创建

多线程就是 我创建的 这个对象 不会 阻塞 程序的运行。

创建一个多线程方法

创建类继承**Thread**类，覆盖**run()**方法，使用**start()**启动线程。

创建类实现**Runnale**接口 实现**run()**方法，即该线程要做的事情。(主要)，启动时将该类包装为**Thread**,使用**start()**启动。包装为**Thread**时，可设置线程**name**

使用**Thread**包装线程时，所有包装后的线程共享同一对象及其内部对象

线程的状态 **new –block start -runnable execute –runing**

## 多线程的同步

多线程的同步 解决 多线程访问同一对象的统一性

加入 **synchronized (key)**可以让多线程同步，使用任意对象来作为此线程的**key**,

单个线程的多线程可以使用**this**作为线程锁， 相当于在方法使用**synchronized**

使用多个**key**线程同步时，此时有可能发生死锁的问题。

避免死锁不要出现同步的嵌套，让同步块尽可能的大(会降低效率)。

## 多个不同线程的多线程通讯问题

多个不同线程对同一对象的修改 需使用线程的同步，线程同步时 使用2个线程共同的修改的对象作为线程同步的**key，线程的调度主要使用**

**key.notify() 随机唤醒一个使用key作为同步锁的线程**

**key.wait() 将使用key同步锁的当前线程挂起等待**

**线程的结束 将Thread使用setDaemon(true);** 作为后台线程，当所有线程结束后，后台线程自动结束。

## Java网络编程 (基于TCP)

1. 创建服务端 **ServerSocket = new ServerSocket(int port);** 端口号
2. 创建客户端 **Socket = new Socket("127.0.0.1", port);**
3. **服务端接受客户端请求 ServerSocket.accept();**
4. **两端的通信，通过获取2端scoket的inputStream 和 outputStream来进行通信。**
5. **服务端应 持续开启 监听客户端的请求。**

## UDP网络编程

## 正则表达式

## XML 基于java处理XML

1. JAVA类运行时获取src下的文件，Java代码编译时会将src下的所有文件同步至bin目录下，可以使用该类运行时的文件路径来进行获取。

**URL url = XXX.class.getClassLoader().getResource("menu.xml");**

**File f3 = new File(url.getPath());**

**File f = new File(url.getFile());**

1. **dom4j基本操作**

**SAXReader reader = new SAXReader();**创建读取对象

**Document document = reader.read(File file);**读取文件至Document

**Element rootElement = document.getRootElement();**获取根节点

**rootElement.elementIterator();**循环跟节点下所有element

**e.attributeValue("id");**根据e的属性名获取e的属性值

**e.elementTextTrim("name");**根据e的子节点名称来获取子节点的文本值

**3. 基于**XPath表达式语法的搜索

**List<Element> nodes = document.selectNodes("/books/book[]");**

**基于Xpath可以很方便的对XML进行搜索，Xpath支持元素,属性,函数等的查找，语法请参考相关文档。**

1. **dom4j写入基本对象至XML文件**

**创建XML写对象**

**XMLWriter xmlWriter = new XMLWriter(new PrintWriter(out), OutputFormat.createPrettyPrint());**

**创建要写入的Doucment**

**Document d = DocumentHelper.createDocument();**

**Doucment填写内容**

**Element food = rt.addElement("food");**

**food.addAttribute("id", id);**

**food.addElement("name").addText(name);**

**用写对象将doucment写出**

**xmlWriter.write(d);**