第十六章 异常机制和File类

16.1 异常机制（重点）

16.1.1 基本概念

- 异常就是"不正常"的含义，在Java语言中主要指程序执行中发生的不正常情况。
- java.lang.Throwable类是Java语言中错误(Error)和异常(Exception)的超类。
- 其中Error类主要用于描述Java虚拟机无法解决的严重错误，通常无法编码解决，如：JVM挂掉了等。
- 其中Exception类主要用于描述因编程错误或偶然外在因素导致的轻微错误，通常可以编码解决，如：0作为除数等。

16.1.2 异常的分类

- java.lang.Exception类是所有异常的超类，主要分为以下两种：

  RuntimeException - 运行时异常，也叫作非检测性异常

  IOException和其它异常 - 其它异常，也叫作检测性异常，所谓检测性异常就是指在编译阶段都能被编译器检测出来的异常。

- 其中RuntimeException类的主要子类：
  ArithmeticException类　- 算术异常
  ArrayIndexOutOfBoundsException类 - 数组下标越界异常
  NullPointerException - 空指针异常
  ClassCastException - 类型转换异常
  NumberFormatException - 数字格式异常

- 注意：
  当程序执行过程中发生异常但又没有手动处理时，则由Java虚拟机采用默认方式处理异常，而默认处理方式就是：打印异常的名称、异常发生的原因、异常发生的位置以及终止程序。

16.1.3 异常的避免

- 在以后的开发中尽量使用if条件判断来避免异常的发生。
- 但是过多的if条件判断会导致程序的代码加长、臃肿，可读性差。

16.1.4 异常的捕获

- 语法格式

```
try {
  编写可能发生异常的代码;
}
catch(异常类型 引用变量名) {
  编写针对该类异常的处理代码；
}
...
finally {
  编写无论是否发生异常都要执行的代码；
}
```

- 注意事项
  a.当需要编写多个catch分支时，切记小类型应该放在大类型的前面；
  b.懒人的写法：
  catch(Exception e) {}
  c.finally通常用于进行善后处理，如：关闭已经打开的文件等。
- 执行流程
  try {
  a;
  b; - 可能发生异常的语句
  c;
  }catch(Exception ex) {
  d;
  }finally {
  e;
  }

  当没有发生异常时的执行流程：a b c e;
  当发生异常时的执行流程：a b d e;

16.1.5 异常的抛出

- 基本概念
  在某些特殊情况下有些异常不能处理或者不便于处理时，就可以将该异常转移给该方法的调用者，这种方法就叫异常的抛出。当方法执行时出现异常，则底层生成一个异常类对象抛出，此时异常代码后续的代码就不再执行。
- 语法格式
  访问权限 返回值类型 方法名称(形参列表) throws 异常类型1,异常类型2,...{ 方法体; }
  如：
  public void show() throws IOException{}
- 方法重写的原则
  a.要求方法名相同、参数列表相同以及返回值类型相同，从jdk1.5开始支持返回子类类型；
  b.要求方法的访问权限不能变小，可以相同或者变大；
  c.要求方法不能抛出更大的异常；
- 注意：
  子类重写的方法不能抛出更大的异常、不能抛出平级不一样的异常，但可以抛出一样的异常、更小的异常以及不抛出异常。
- 经验分享
  - 若父类中被重写的方法没有抛出异常时，则子类中重写的方法只能进行异常的捕获处理。
  - 若一个方法内部又以递进方式分别调用了好几个其它方法，则建议这些方法内可以使用抛出的方法处理到最后一层进行捕获方式处理。

16.1.6 自定义异常

- 基本概念
  当需要在程序中表达年龄不合理的情况时，而Java官方又没有提供这种针对性的异常，此时就需要程序员自定义异常加以描述。
- 实现流程
  a.自定义xxxException异常类继承Exception类或者其子类。
  b.提供两个版本的构造方法，一个是无参构造方法，另外一个是字符串作为参数的构造方法。
- 异常的产生
  throw new 异常类型(实参);

如：
throw new AgeException("年龄不合理！！！");

- Java采用的异常处理机制是将异常处理的程序代码集中在一起，与正常的程序代码分开，使得程序简洁、优雅，并易于维护。

16.2 File类（重点）

16.2.1 基本概念

- java.io.File类主要用于描述文件或目录路径的抽象表示信息，可以获取文件或目录的特征信息，如：大小等。

16.2.2 常用的方法

方法声明	功能概述
File(String pathname)	根据参数指定的路径名来构造对象
File(String parent, String child)	根据参数指定的父路径和子路径信息构造对象
File(File parent, String child)	根据参数指定的父抽象路径和子路径信息构造对象
boolean exists()	测试此抽象路径名表示的文件或目录是否存在
String getName()	用于获取文件的名称
long length()	返回由此抽象路径名表示的文件的长度
long lastModified()	用于获取文件的最后一次修改时间
String getAbsolutePath()	用于获取绝对路径信息
boolean delete()	用于删除文件，当删除目录时要求是空目录
boolean createNewFile()	用于创建新的空文件
boolean mkdir()	用于创建目录
boolean mkdirs()	用于创建多级目录
File[] listFiles()	获取该目录下的所有内容
boolean isFile()	判断是否为文件
boolean isDirectory()	判断是否为目录
File[] listFiles(FileFilter filter)	获取目录下满足筛选器的所有内容

- 案例题目

  遍历指定目录以及子目录中的所有内容并打印出来。

第十七章 IO流

17.1 IO流的概念

- IO就是Input和Output的简写，也就是输入和输出的含义。
- IO流就是指读写数据时像流水一样从一端流到另外一端，因此得名为“流"。

17.2 基本分类

- 按照读写数据的基本单位不同，分为 字节流 和 字符流。
  其中字节流主要指以字节为单位进行数据读写的流，可以读写任意类型的文件。
  其中字符流主要指以字符(2个字节)为单位进行数据读写的流，只能读写文本文件。
- 按照读写数据的方向不同，分为 输入流 和 输出流（站在程序的角度）。
  其中输入流主要指从文件中读取数据内容输入到程序中，也就是读文件。
  其中输出流主要指将程序中的数据内容输出到文件中，也就是写文件。
- 按照流的角色不同分为节点流和处理流。
  其中节点流主要指直接和输入输出源对接的流。
  其中处理流主要指需要建立在节点流的基础之上的流。

17.3 体系结构

分类	字节输入流	字节输出流	字符输入流	字符输出流
抽象基类	InputStream	OutputStream	Reader	Writer
访问文件	FileInputStream	FileOutputStream	FileReader	FileWriter
访问数组	ByteArrayInputStream	ByteArrayOutpuStream	CharArrayReader	CharArrayWriter
访问管道	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
访问字符串	--	--	StringReader	StringWriter
缓冲流	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
转换流	--	--	InputStreamReader	OutputStreamWriter
对象流	ObjectInputStream	ObjectOutputStream	--	--
	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
打印流	--	PrintStream	--	PrintWriter
推回输入流	PushbackInputStream	--	PushbackReader	--
特殊流	DataInputStream	DataOutputStream	--	--

17.4 相关流的详解

17.4.1 FileWriter类（重点）

（1）基本概念

- java.io.FileWriter类主要用于将文本内容写入到文本文件。

（2）常用的方法

方法声明	功能介绍
FileWriter(String fileName)	根据参数指定的文件名构造对象
FileWriter(String fileName, boolean append)	以追加的方式根据参数指定的文件名来构造对象
void write(int c)	写入单个字符
void write(char[] cbuf, int off, int len)	将指定字符数组中从偏移量off开始的len个字符写入此文件输出流
void write(char[] cbuf)	将cbuf.length个字符从指定字符数组写入此文件输出流中
void flush()	刷新流
void close()	关闭流对象并释放有关的资源

17.4.2 FileReader类（重点）

（1）基本概念

- java.io.FileReader类主要用于从文本文件读取文本数据内容。

（2）常用的方法

方法声明	功能介绍
FileReader(String fileName)	根据参数指定的文件名构造对象
int read()	读取单个字符的数据并返回，返回-1表示读取到末尾
int read(char[] cbuf, int offset, int length)	从输入流中将最多len个字符的数据读入一个字符数组中，返回读取到的字符个数，返回-1表示读取到末尾
int read(char[] cbuf)	从此输入流中将最多 cbuf.length 个字符的数据读入字符数组中，返回读取到的字符个数，返回-1表示读取到末尾
void close()	关闭流对象并释放有关的资源

17.4.3 FileOutputStream类（重点）

（1）基本概念

- java.io.FileOutputStream类主要用于将图像数据之类的原始字节流写入到输出流中。

（2）常用的方法

方法声明	功能介绍
FileOutputStream(String name)	根据参数指定的文件名来构造对象
FileOutputStream(String name, boolean append)	以追加的方式根据参数指定的文件名来构造对象
void write(int b)	将指定字节写入此文件输出流
void write(byte[] b, int off, int len)	将指定字节数组中从偏移量off开始的len个字节写入此文件输出流
void write(byte[] b)	将 b.length 个字节从指定字节数组写入此文件输出流中
void flush()	刷新此输出流并强制写出任何缓冲的输出字节
void close()	关闭流对象并释放有关的资源

17.4.4 FileInputStream类（重点）

（1）基本概念

- java.io.FileInputStream类主要用于从输入流中以字节流的方式读取图像数据等。

（2）常用的方法

方法声明	功能介绍
FileInputStream(String name)	根据参数指定的文件路径名来构造对象
int read()	从输入流中读取单个字节的数据并返回，返回-1表示读取到末尾
int read(byte[] b, int off, int len)	从此输入流中将最多len个字节的数据读入字节数组中，返回读取到的字节个数，返回-1表示读取到末尾
int read(byte[] b)	从此输入流中将最多 b.length 个字节的数据读入字节数组中，返回读取到的字节个数，返回-1表示读取到末尾
void close()	关闭流对象并释放有关的资源
int available()	获取输入流所关联文件的大小

- 案例题目

  编程实现两个文件之间的拷贝功能。

17.4.5 BufferedOutputStream类（重点）

（1）基本概念

- java.io.BufferedOutputStream类主要用于描述缓冲输出流，此时不用为写入的每个字节调用底层系统。

（2）常用的方法

方法声明	功能介绍
BufferedOutputStream(OutputStream out)	根据参数指定的引用来构造对象
BufferedOutputStream(OutputStream out, int size)	根据参数指定的引用和缓冲区大小来构造对象
void write(int b)	写入单个字节
void write(byte[] b, int off, int len)	写入字节数组中的一部分数据
void write(byte[] b)	写入参数指定的整个字节数组
void flush()	刷新流
void close()	关闭流对象并释放有关的资源

17.4.6 BufferedInputStream类（重点）

（1）基本概念

- java.io.BufferedInputStream类主要用于描述缓冲输入流。

（2）常用的方法

方法声明	功能介绍
BufferedInputStream(InputStream in)	根据参数指定的引用构造对象
BufferedInputStream(InputStream in, int size)	根据参数指定的引用和缓冲区大小构造对象
int read()	读取单个字节
int read(byte[] b, int off, int len)	读取len个字节
int read(byte[] b)	读取b.length个字节
void close()	关闭流对象并释放有关的资源

17.4.7 BufferedWriter类（重点）

（1）基本概念

- java.io.BufferedWriter类主要用于写入单个字符、字符数组以及字符串到输出流中。

（2）常用的方法

方法声明	功能介绍
BufferedWriter(Writer out)	根据参数指定的引用来构造对象
BufferedWriter(Writer out, int sz)	根据参数指定的引用和缓冲区大小来构造对象
void write(int c)	写入单个字符到输出流中
void write(char[] cbuf, int off, int len)	将字符数组cbuf中从下标off开始的len个字符写入输出流中
void write(char[] cbuf)	将字符串数组cbuf中所有内容写入输出流中
void write(String s, int off, int len)	将参数s中下标从off开始的len个字符写入输出流中
void write(String str)	将参数指定的字符串内容写入输出流中
void newLine()	用于写入行分隔符到输出流中
void flush()	刷新流
void close()	关闭流对象并释放有关的资源

17.4.8 BufferedReader类（重点）

（1）基本概念

- java.io.BufferedReader类用于从输入流中读取单个字符、字符数组以及字符串。

（2）常用的方法

方法声明	功能介绍
BufferedReader(Reader in)	根据参数指定的引用来构造对象
BufferedReader(Reader in, int sz)	根据参数指定的引用和缓冲区大小来构造对象
int read()	从输入流读取单个字符，读取到末尾则返回-1，否则返回实际读取到的字符内容
int read(char[] cbuf, int off, int len)	从输入流中读取len个字符放入数组cbuf中下标从off开始的位置上，若读取到末尾则返回-1，否则返回实际读取到的字符个数
int read(char[] cbuf)	从输入流中读满整个数组cbuf
String readLine()	读取一行字符串并返回，返回null表示读取到末尾
void close()	关闭流对象并释放有关的资源

17.4.9 PrintStream类

（1）基本概念

- java.io.PrintStream类主要用于更加方便地打印各种数据内容。

（2）常用的方法

方法声明	功能介绍
PrintStream(OutputStream out)	根据参数指定的引用来构造对象
void print(String s)	用于将参数指定的字符串内容打印出来
void println(String x)	用于打印字符串后并终止该行
void flush()	刷新流
void close()	用于关闭输出流并释放有关的资源

17.4.10 PrintWriter类

（1）基本概念

- java.io.PrintWriter类主要用于将对象的格式化形式打印到文本输出流。

（2）常用的方法

方法声明	功能介绍
PrintWriter(Writer out)	根据参数指定的引用来构造对象
void print(String s)	将参数指定的字符串内容打印出来
void println(String x)	打印字符串后并终止该行
void flush()	刷新流
void close()	关闭流对象并释放有关的资源

- 案例题目

  不断地提示用户输入要发送的内容，若发送的内容是"bye"则聊天结束，否则将用户输入的内容写入到文件d:/a.txt中。
  要求使用BufferedReader类来读取键盘的输入　System.in代表键盘输入
  要求使用PrintStream类负责将数据写入文件

17.4.11 OutputStreamWriter类

（1）基本概念

- java.io.OutputStreamWriter类主要用于实现从字符流到字节流的转换。

（2）常用的方法

方法声明	功能介绍
OutputStreamWriter(OutputStream out)	根据参数指定的引用来构造对象
OutputStreamWriter(OutputStream out, String charsetName)	根据参数指定的引用和编码构造对象
void write(String str)	将参数指定的字符串写入
void flush()	刷新流
void close()	用于关闭输出流并释放有关的资源

17.4.12 InputStreamReader类

（1）基本概念

- java.io.InputStreamReader类主要用于实现从字节流到字符流的转换。

（2）常用的方法

方法声明	功能介绍
InputStreamReader(InputStream in)	根据参数指定的引用来构造对象
InputStreamReader(InputStream in, String charsetName)	根据参数指定的引用和编码来构造对象
int read(char[] cbuf)	读取字符数据到参数指定的数组
void close()	用于关闭输出流并释放有关的资源

17.4.13 字符编码

（1）编码表的由来

- 计算机只能识别二进制数据，早期就是电信号。为了方便计算机可以识别各个国家的文字，就需要将各个国家的文字采用数字编号的方式进行描述并建立对应的关系表，该表就叫做编码表。

（2）常见的编码表

- ASCII：美国标准信息交换码， 使用一个字节的低7位二位进制进行表示。
- ISO8859-1：拉丁码表，欧洲码表，使用一个字节的8位二进制进行表示。
- GB2312：中国的中文编码表，最多使用两个字节16位二进制为进行表示。
- GBK：中国的中文编码表升级，融合了更多的中文文字符号，最多使用两个字节16位二进制位表示。
- Unicode：国际标准码，融合了目前人类使用的所有字符，为每个字符分配唯一的字符码。所有的文字都用两个字节16位二进制位来表示。

（3）编码的发展

- 面向传输的众多 UTF（UCS Transfer Format）标准出现了，UTF-8就是每次8个位传输数据，而UTF-16就是每次16个位。这是为传输而设计的编码并使编码无国界，这样就可以显示全世界上所有文化的字符了。

- Unicode只是定义了一个庞大的、全球通用的字符集，并为每个字符规定了唯一确定的编号，具体存储成什么样的字节流，取决于字符编码方案。推荐的Unicode编码是UTF-8和UTF-16。
- UTF-8：变长的编码方式，可用1-4个字节来表示一个字符。

17.4.14 DataOutputStream类（了解）

（1）基本概念

- java.io.DataOutputStream类主要用于以适当的方式将基本数据类型写入输出流中。

（2）常用的方法

方法声明	功能介绍
DataOutputStream(OutputStream out)	根据参数指定的引用构造对象 OutputStream类是个抽象类，实参需要传递子类对象
void writeInt(int v)	用于将参数指定的整数一次性写入输出流，优先写入高字节
void close()	用于关闭文件输出流并释放有关的资源

17.4.15 DataInputStream类（了解）

（1）基本概念

- java.io.DataInputStream类主要用于从输入流中读取基本数据类型的数据。

（2）常用的方法

方法声明	功能介绍
DataInputStream(InputStream in)	根据参数指定的引用来构造对象 InputStream类是抽象类，实参需要传递子类对象
int readInt()	用于从输入流中一次性读取一个整数数据并返回
void close()	用于关闭文件输出流并释放有关的资源

17.4.16 ObjectOutputStream类（重点）

（1）基本概念

- java.io.ObjectOutputStream类主要用于将一个对象的所有内容整体写入到输出流中。
- 只能将支持 java.io.Serializable 接口的对象写入流中。
- 类通过实现 java.io.Serializable 接口以启用其序列化功能。
- 所谓序列化主要指将一个对象需要存储的相关信息有效组织成字节序列的转化过程。

（2）常用的方法

方法声明	功能介绍
ObjectOutputStream(OutputStream out)	根据参数指定的引用来构造对象
void writeObject(Object obj)	用于将参数指定的对象整体写入到输出流中
void close()	用于关闭输出流并释放有关的资源

17.4.17 ObjectInputStream类（重点）

（1）基本概念

- java.io.ObjectInputStream类主要用于从输入流中一次性将对象整体读取出来。
- 所谓反序列化主要指将有效组织的字节序列恢复为一个对象及相关信息的转化过程。

（2）常用的方法

方法声明	功能介绍
ObjectInputStream(InputStream in)	根据参数指定的引用来构造对象
Object readObject()	主要用于从输入流中读取一个对象并返回 无法通过返回值来判断是否读取到文件的末尾
void close()	用于关闭输入流并释放有关的资源

（3）序列化版本号

- 序列化机制是通过在运行时判断类的serialVersionUID来验证版本一致性的。在进行反序列化时，JVM会把传来的字节流中的serialVersionUID与本地相应实体类的serialVersionUID进行比较，如果相同就认为是一致的，可以进行反序列化，否则就会出现序列化版本不一致的异常(InvalidCastException)。

（4）transient关键字

- transient是Java语言的关键字，用来表示一个域不是该对象串行化的一部分。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

（5）经验的分享

- 当希望将多个对象写入文件时，通常建议将多个对象放入一个集合中，然后将集合这个整体看做一个对象写入输出流中，此时只需要调用一次readObject方法就可以将整个集合的数据读取出来，从而避免了通过返回值进行是否达到文件末尾的判断。

17.4.18 RandomAccessFile类

（1）基本概念

- java.io.RandomAccessFile类主要支持对随机访问文件的读写操作。

（2）常用的方法

方法声明	功能介绍
RandomAccessFile(String name, String mode)	根据参数指定的名称和模式构造对象 r: 以只读方式打开 rw：打开以便读取和写入 rwd:打开以便读取和写入，同步文件内容的更新 rws:打开以便读取和写入，同步文件内容和元数据的更新
int read()	读取单个字节的数据
void seek(long pos)	用于设置从此文件的开头开始测量的文件指针偏移量
void write(int b)	将参数指定的单个字节写入
void close()	用于关闭流并释放有关的资源

第十八章 多线程

18.1 基本概念

18.1.1 程序和进程的概念

- 程序 - 数据结构 + 算法，主要指存放在硬盘上的可执行文件。
- 进程 - 主要指运行在内存中的可执行文件。
- 目前主流的操作系统都支持多进程，为了让操作系统同时可以执行多个任务，但进程是重量级的，也就是新建一个进程会消耗CPU和内存空间等系统资源，因此进程的数量比较局限。

18.1.2 线程的概念

- 为了解决上述问题就提出线程的概念，线程就是进程内部的程序流，也就是说操作系统内部支持多进程的，而每个进程的内部又是支持多线程的，线程是轻量的，新建线程会共享所在进程的系统资源，因此目前主流的开发都是采用多线程。
- 多线程是采用时间片轮转法来保证多个线程的并发执行，所谓并发就是指宏观并行微观串行的机制。

18.2 线程的创建（重中之重）

18.2.1 Thread类的概念

- java.lang.Thread类代表线程，任何线程对象都是Thread类（子类）的实例。
- Thread类是线程的模板，封装了复杂的线程开启等操作，封装了操作系统的差异性。

18.2.2 创建方式

- 自定义类继承Thread类并重写run方法，然后创建该类的对象调用start方法。
- 自定义类实现Runnable接口并重写run方法，创建该类的对象作为实参来构造Thread类型的对象，然后使用Thread类型的对象调用start方法。

18.2.3 相关的方法

方法声明	功能介绍
Thread()	使用无参的方式构造对象
Thread(String name)	根据参数指定的名称来构造对象
Thread(Runnable target)	根据参数指定的引用来构造对象，其中Runnable是个接口类型
Thread(Runnable target, String name)	根据参数指定引用和名称来构造对象
void run()	若使用Runnable引用构造了线程对象，调用该方法时最终调用接口中的版本 若没有使用Runnable引用构造线程对象，调用该方法时则啥也不做
void start()	用于启动线程，Java虚拟机会自动调用该线程的run方法

18.2.4 执行流程

- 执行main方法的线程叫做主线程，执行run方法的线程叫做新线程/子线程。
- main方法是程序的入口，对于start方法之前的代码来说，由主线程执行一次，当start方法调用成功后线程的个数由1个变成了2个，新启动的线程去执行run方法的代码，主线程继续向下执行，两个线程各自独立运行互不影响。
- 当run方法执行完毕后子线程结束，当main方法执行完毕后主线程结束。
- 两个线程执行没有明确的先后执行次序，由操作系统调度算法来决定。

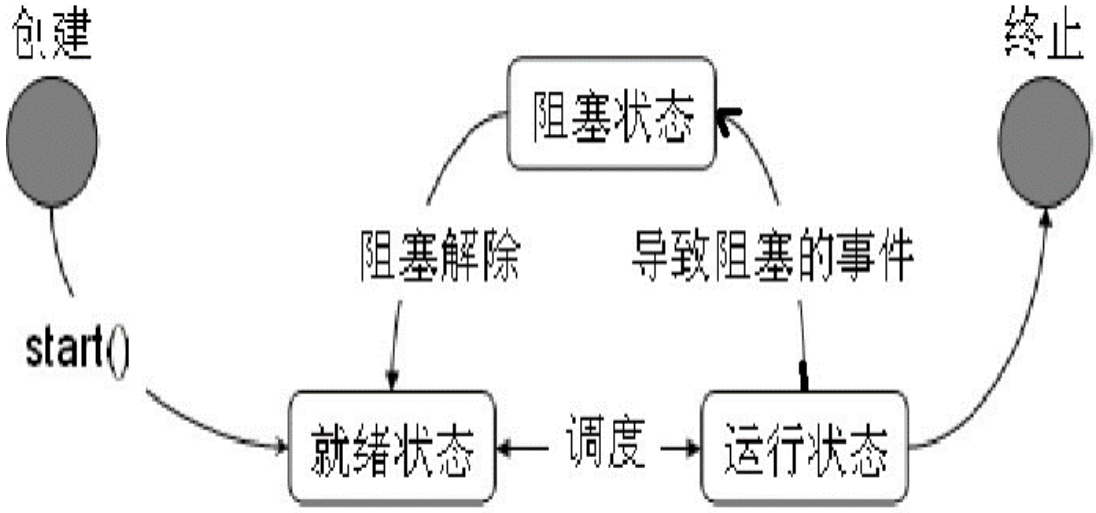
18.2.5 方式的比较

- 继承Thread类的方式代码简单，但是若该类继承Thread类后则无法继承其它类，而实现Runnable接口的方式代码复杂，但不影响该类继承其它类以及实现其它接口，因此以后的开发中推荐使用第二种方式。

18.2.6 匿名内部类的方式

- 使用匿名内部类的方式来创建和启动线程。

18.3 线程的生命周期（熟悉）



- 新建状态 - 使用new关键字创建之后进入的状态，此时线程并没有开始执行。
- 就绪状态 - 调用start方法后进入的状态，此时线程还是没有开始执行。
- 运行状态 - 使用线程调度器调用该线程后进入的状态，此时线程开始执行，当线程的时间片执行完毕后任务没有完成时回到就绪状态。
- 消亡状态 - 当线程的任务执行完成后进入的状态，此时线程已经终止。
- 阻塞状态 - 当线程执行的过程中发生了阻塞事件进入的状态，如：sleep方法。
  阻塞状态解除后进入就绪状态。

18.4 线程的编号和名称（熟悉）

方法声明	功能介绍
long getId()	获取调用对象所表示线程的编号
String getName()	获取调用对象所表示线程的名称
void setName(String name)	设置/修改线程的名称为参数指定的数值
static Thread currentThread()	获取当前正在执行线程的引用

- 案例题目

  自定义类继承Thread类并重写run方法，在run方法中先打印当前线程的编号和名称，然后将线程的名称修改为"zhangfei"后再次打印编号和名称。
  要求在main方法中也要打印主线程的编号和名称。

18.5 常用的方法（重点）

方法声明	功能介绍
static void yield()	当前线程让出处理器（离开Running状态），使当前线程进入Runnable状态等待
static void sleep(times)	使当前线程从 Running 放弃处理器进入Block状态, 休眠times毫秒, 再返回到Runnable如果其他线程打断当前线程的Block(sleep), 就会发生InterruptedException。
int getPriority()	获取线程的优先级
void setPriority(int newPriority)	修改线程的优先级。 优先级越高的线程不一定先执行，但该线程获取到时间片的机会会更多一些
void join()	等待该线程终止
void join(long millis)	等待参数指定的毫秒数
boolean isDaemon()	用于判断是否为守护线程
void setDaemon(boolean on)	用于设置线程为守护线程

- 案例题目

  编程创建两个线程，线程一负责打印1 ~ 100之间的所有奇数，其中线程二负责打印1 ~ 100之间的所有偶数。
  在main方法启动上述两个线程同时执行,主线程等待两个线程终止。

18.6 线程同步机制（重点）

18.6.1 基本概念

- 当多个线程同时访问同一种共享资源时，可能会造成数据的覆盖等不一致性问题，此时就需要对线程之间进行通信和协调，该机制就叫做线程的同步机制。
- 多个线程并发读写同一个临界资源时会发生线程并发安全问题。
- 异步操作:多线程并发的操作，各自独立运行。
- 同步操作:多线程串行的操作，先后执行的顺序。

18.6.2 解决方案

- 由程序结果可知：当两个线程同时对同一个账户进行取款时，导致最终的账户余额不合理。
- 引发原因：线程一执行取款时还没来得及将取款后的余额写入后台，线程二就已经开始取款。
- 解决方案：让线程一执行完毕取款操作后，再让线程二执行即可，将线程的并发操作改为串行操作。
- 经验分享：在以后的开发尽量减少串行操作的范围，从而提高效率。

18.6.3 实现方式

- 在Java语言中使用synchronized关键字来实现同步/对象锁机制从而保证线程执行的原子性，具体方式如下：

- 使用同步代码块的方式实现部分代码的锁定，格式如下：
  synchronized(类类型的引用) {
  编写所有需要锁定的代码；
  }
- 使用同步方法的方式实现所有代码的锁定。
  直接使用synchronized关键字来修饰整个方法即可
  该方式等价于:
  synchronized(this) { 整个方法体的代码 }

18.6.4 静态方法的锁定

- 当我们对一个静态方法加锁，如:
  public synchronized static void xxx(){....}
- 那么该方法锁的对象是类对象。每个类都有唯一的一个类对象。获取类对象的方式:类名.class。
- 静态方法与非静态方法同时使用了synchronized后它们之间是非互斥关系的。
- 原因在于：静态方法锁的是类对象而非静态方法锁的是当前方法所属对象。

18.6.5 注意事项

- 使用synchronized保证线程同步应当注意:
  - 多个需要同步的线程在访问同步块时，看到的应该是同一个锁对象引用。
  - 在使用同步块时应当尽量减少同步范围以提高并发的执行效率。

18.6.6 线程安全类和不安全类

- StringBuffer类是线程安全的类，但StringBuilder类不是线程安全的类。
- Vector类和 Hashtable类是线程安全的类，但ArrayList类和HashMap类不是线程安全的类。
- Collections.synchronizedList() 和 Collections.synchronizedMap()等方法实现安全。

18.6.7 死锁的概念

- 线程一执行的代码：

```
public void run(){
  synchronized(a){   //持有对象锁a，等待对象锁b
    synchronized(b){
      编写锁定的代码;
    }
  }
}
```

- 线程二执行的代码：

```
public void run(){
  synchronized(b){   //持有对象锁b，等待对象锁a
    synchronized(a){
      编写锁定的代码;
    }
  }
}
```

- 注意：
  在以后的开发中尽量减少同步的资源，减少同步代码块的嵌套结构的使用！

18.6.8 使用Lock（锁）实现线程同步

（1）基本概念

- 从Java5开始提供了更强大的线程同步机制—使用显式定义的同步锁对象来实现。
- java.util.concurrent.locks.Lock接口是控制多个线程对共享资源进行访问的工具。
- 该接口的主要实现类是ReentrantLock类，该类拥有与synchronized相同的并发性，在以后的线程安全控制中，经常使用ReentrantLock类显式加锁和释放锁。

（2）常用的方法

方法声明	功能介绍
ReentrantLock()	使用无参方式构造对象
void lock()	获取锁
void unlock()	释放锁

（3）与synchronized方式的比较

- Lock是显式锁，需要手动实现开启和关闭操作，而synchronized是隐式锁，执行锁定代码后自动释放。
- Lock只有同步代码块方式的锁，而synchronized有同步代码块方式和同步方法两种锁。
- 使用Lock锁方式时，Java虚拟机将花费较少的时间来调度线程，因此性能更好。

18.6.9 Object类常用的方法

方法声明	功能介绍
void wait()	用于使得线程进入等待状态，直到其它线程调用notify()或notifyAll()方法
void wait(long timeout)	用于进入等待状态，直到其它线程调用方法或参数指定的毫秒数已经过去为止
void notify()	用于唤醒等待的单个线程
void notifyAll()	用于唤醒等待的所有线程

18.6.10 线程池（熟悉）

（1）实现Callable接口

- 从Java5开始新增加创建线程的第三种方式为实现java.util.concurrent.Callable接口。
- 常用的方法如下：

方法声明	功能介绍
V call()	计算结果并返回

（2）FutureTask类

- java.util.concurrent.FutureTask类用于描述可取消的异步计算，该类提供了Future接口的基本实现，包括启动和取消计算、查询计算是否完成以及检索计算结果的方法，也可以用于获取方法调用后的返回结果。
- 常用的方法如下：

方法声明	功能介绍
FutureTask(Callable callable)	根据参数指定的引用来创建一个未来任务
V get()	获取call方法计算的结果

（3）线程池的由来

- 在服务器编程模型的原理，每一个客户端连接用一个单独的线程为之服务，当与客户端的会话结束时，线程也就结束了，即每来一个客户端连接，服务器端就要创建一个新线程。
- 如果访问服务器的客户端很多，那么服务器要不断地创建和销毁线程，这将严重影响服务器的性能。

（4）概念和原理

- 线程池的概念：首先创建一些线程，它们的集合称为线程池，当服务器接受到一个客户请求后，就从线程池中取出一个空闲的线程为之服务，服务完后不关闭该线程，而是将该线程还回到线程池中。
- 在线程池的编程模式下，任务是提交给整个线程池，而不是直接交给某个线程，线程池在拿到任务后，它就在内部找有无空闲的线程，再把任务交给内部某个空闲的线程，任务是提交给整个线程池，一个线程同时只能执行一个任务，但可以同时向一个线程池提交多个任务。

（5）相关类和方法

- 从Java5开始提供了线程池的相关类和接口：java.util.concurrent.Executors类和java.util.concurrent.ExecutorService接口。
- 其中Executors是个工具类和线程池的工厂类，可以创建并返回不同类型的线程池，常用方法如下：

方法声明	功能介绍
static ExecutorService newCachedThreadPool()	创建一个可根据需要创建新线程的线程池
static ExecutorService newFixedThreadPool(int nThreads)	创建一个可重用固定线程数的线程池
static ExecutorService newSingleThreadExecutor()	创建一个只有一个线程的线程池

- 其中ExecutorService接口是真正的线程池接口，主要实现类是ThreadPoolExecutor，常用方法如下：

方法声明	功能介绍
void execute(Runnable command)	执行任务和命令，通常用于执行Runnable
Future submit(Callable task)	执行任务和命令，通常用于执行Callable
void shutdown()	启动有序关闭

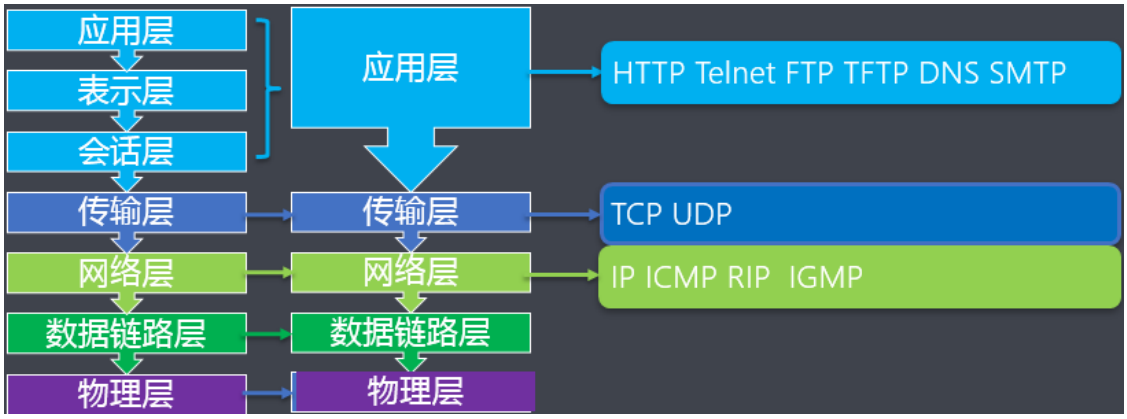
第十九章 网络编程

19.1 网络编程的常识

- 目前主流的网络通讯软件有：微信、QQ、飞信、阿里旺旺、陌陌、探探、...

19.1.1 七层网络模型

- OSI（Open System Interconnect），即开放式系统互联，是ISO（国际标准化组织）组织在1985年研究的网络互连模型。
- OSI七层模型和TCP/IP五层模型的划分如下：



- 当发送数据时，需要对发送的内容按照上述七层模型进行层层加包后发送出去。
- 当接收数据时，需要对接收的内容按照上述七层模型相反的次序层层拆包并显示出来。

19.1.2 相关的协议（笔试题）

（1）协议的概念

- 计算机在网络中实现通信就必须有一些约定或者规则，这种约定和规则就叫做通信协议，通信协议可以对速率、传输代码、代码结构、传输控制步骤、出错控制等制定统一的标准。

（2）TCP协议

- 传输控制协议(Transmission Control Protocol)，是一种面向连接的协议，类似于打电话。
    - 建立连接 => 进行通信 => 断开连接
    - 在传输前采用"三次握手"方式。
    - 在通信的整个过程中全程保持连接，形成数据传输通道。
    - 保证了数据传输的可靠性和有序性。
    - 是一种全双工的字节流通信方式，可以进行大数据量的传输。
    - 传输完毕后需要释放已建立的连接，发送数据的效率比较低。

（3）UDP协议

- 用户数据报协议(User Datagram Protocol)，是一种非面向连接的协议，类似于写信。
    - 在通信的整个过程中不需要保持连接，其实是不需要建立连接。
    - 不保证数据传输的可靠性和有序性。
    - 是一种全双工的数据报通信方式，每个数据报的大小限制在64K内。
    - 发送数据完毕后无需释放资源，开销小，发送数据的效率比较高，速度快。

19.1.3 IP地址（重点）

- 192.168.1.1 - 是绝大多数路由器的登录地址，主要配置用户名和密码以及Mac过滤。
- IP地址是互联网中的唯一地址标识，本质上是由32位二进制组成的整数，叫做IPv4，当然也有128位二进制组成的整数，叫做IPv6，目前主流的还是IPv4。
- 日常生活中采用点分十进制表示法来进行IP地址的描述，将每个字节的二进制转化为一个十进制整数，不同的整数之间采用小数点隔开。
- 如：
  0x01020304 => 1.2.3.4
- 查看IP地址的方式：
  Windows系统：在dos窗口中使用ipconfig或ipconfig/all命令即可
  Unix/linux系统：在终端窗口中使用ifconfig或/sbin/ifconfig命令即可
- 特殊的地址

  本地回环地址(hostAddress)：127.0.0.1 主机名(hostName)：localhost

19.1.4 端口号（重点）

- IP地址 - 可以定位到具体某一台设备。
- 端口号 - 可以定位到该设备中具体某一个进程。
- 端口号本质上是16位二进制组成的整数，表示范围是：0 ~ 65535，其中0 ~ 1024之间的端口号通常被系统占用，建议编程从1025开始使用。
- 特殊的端口：
  HTTP:80 FTP:21 Oracle:1521 MySQL:3306 Tomcat:8080
- 网络编程需要提供：IP地址 + 端口号，组合在一起叫做网络套接字：Socket。

19.2 基于tcp协议的编程模型（重点）

19.2.1 C/S架构的简介

- 在C/S模式下客户向服务器发出服务请求，服务器接收请求后提供服务。
- 例如：在一个酒店中，顾客找服务员点菜,服务员把点菜单通知厨师，厨师按点菜单做好菜后让服务员端给客户，这就是一种C/S工作方式。如果把酒店看作一个系统，服务员就是客户端，厨师就是服务器。这种系统分工和协同工作的方式就是C/S的工作方式。
- 客户端部分：为每个用户所专有的，负责执行前台功能。
- 服务器部分：由多个用户共享的信息与功能，招待后台服务。

19.2.2 编程模型

- 服务器：
  （1）创建ServerSocket类型的对象并提供端口号；
  （2）等待客户端的连接请求，调用accept()方法；
  （3）使用输入输出流进行通信；
  （4）关闭Socket；
- 客户端：
  （1）创建Socket类型的对象并提供服务器的IP地址和端口号；
  （2）使用输入输出流进行通信；
  （3）关闭Socket；

19.2.3 相关类和方法的解析

（1）ServerSocket类

- java.net.ServerSocket类主要用于描述服务器套接字信息（大插排）。
- 常用的方法如下：

方法声明	功能介绍
ServerSocket(int port)	根据参数指定的端口号来构造对象
Socket accept()	侦听并接收到此套接字的连接请求
void close()	用于关闭套接字

（2）Socket类

- java.net.Socket类主要用于描述客户端套接字，是两台机器间通信的端点（小插排）。
- 常用的方法如下：

方法声明	功能介绍
Socket(String host, int port)	根据指定主机名和端口来构造对象
InputStream getInputStream()	用于获取当前套接字的输入流
OutputStream getOutputStream()	用于获取当前套接字的输出流
void close()	用于关闭套接字

（3）注意事项

- 客户端 Socket 与服务器端 Socket 对应, 都包含输入和输出流。
- 客户端的socket.getInputStream() 连接于服务器socket.getOutputStream()。
- 客户端的socket.getOutputStream()连接于服务器socket.getInputStream()

19.3 基于udp协议的编程模型（熟悉）

19.3.1 编程模型

- 接收方：
  （1）创建DatagramSocket类型的对象并提供端口号；
  （2）创建DatagramPacket类型的对象并提供缓冲区；
  （3）通过Socket接收数据内容存放到Packet中，调用receive方法；
  （4）关闭Socket；
- 发送方：
  （1）创建DatagramSocket类型的对象；
  （2）创建DatagramPacket类型的对象并提供接收方的通信地址；
  （3）通过Socket将Packet中的数据内容发送出去，调用send方法；
  （4）关闭Socket；

19.3.2 相关类和方法的解析

（1）DatagramSocket类

- java.net.DatagramSocket类主要用于描述发送和接收数据报的套接字(邮局)。
  换句话说，该类就是包裹投递服务的发送或接收点。
- 常用的方法如下：

方法声明	功能介绍
DatagramSocket()	使用无参的方式构造对象
DatagramSocket(int port)	根据参数指定的端口号来构造对象
void receive(DatagramPacket p)	用于接收数据报存放到参数指定的位置
void send(DatagramPacket p)	用于将参数指定的数据报发送出去
void close()	关闭Socket并释放相关资源

（2）DatagramPacket类

- java.net.DatagramPacket类主要用于描述数据报，数据报用来实现无连接包裹投递服务。
- 常用的方法如下：

方法声明	功能介绍
DatagramPacket(byte[] buf, int length)	根据参数指定的数组来构造对象，用于接收长度为length的数据报
DatagramPacket(byte[] buf, int length, InetAddress address, int port)	根据参数指定数组来构造对象，将数据报发送到指定地址和端口
InetAddress getAddress()	用于获取发送方或接收方的通信地址
int getPort()	用于获取发送方或接收方的端口号
int getLength()	用于获取发送数据或接收数据的长度

（3）InetAddress类

- java.net.InetAddress类主要用于描述互联网通信地址信息。
- 常用的方法如下：

方法声明	功能介绍
static InetAddress getLocalHost()	用于获取当前主机的通信地址
static InetAddress getByName(String host)	根据参数指定的主机名获取通信地址

19.4 URL类（熟悉）

19.4.1 基本概念

- java.net.URL（Uniform Resource Identifier）类主要用于表示统一的资源定位器，也就是指向万维网上“资源”的指针。这个资源可以是简单的文件或目录，也可以是对复杂对象的引用，例如对数据库或搜索引擎的查询等。
- 通过URL可以访问万维网上的网络资源，最常见的就是www和ftp站点，浏览器通过解析给定的URL可以在网络上查找相应的资源。
- URL的基本结构如下：
  <传输协议>://<主机名>:<端口号>/<资源地址>

19.4.2 常用的方法

方法声明	功能介绍
URL(String spec)	根据参数指定的字符串信息构造对象
String getProtocol()	获取协议名称
String getHost()	获取主机名称
int getPort()	获取端口号
String getPath()	获取路径信息
String getFile()	获取文件名
URLConnection openConnection()	获取URLConnection类的实例

19.4.3 URLConnection类

（1）基本概念

- java.net.URLConnection类是个抽象类，该类表示应用程序和URL之间的通信链接的所有类的超类，主要实现类有支持HTTP特有功能的HttpURLConnection类。

（2）HttpURLConnection类的常用方法

方法声明	功能介绍
InputStream getInputStream()	获取输入流
void disconnect()	断开连接

第二十章 反射机制

20.1 基本概念

- 通常情况下编写代码都是固定的，无论运行多少次执行的结果也是固定的，在某些特殊场合中编写代码时不确定要创建什么类型的对象，也不确定要调用什么样的方法，这些都希望通过运行时传递的参数来决定，该机制叫做动态编程技术，也就是反射机制。
- 通俗来说，反射机制就是用于动态创建对象并且动态调用方法的机制。
- 目前主流的框架底层都是采用反射机制实现的。
- 如：
  Person p = new Person(); - 表示声明Person类型的引用指向Person类型的对象
  p.show(); - 表示调用Person类中的成员方法show

20.2 Class类

20.2.1 基本概念

- java.lang.Class类的实例可以用于描述Java应用程序中的类和接口，也就是一种数据类型。
- 该类没有公共构造方法，该类的实例由Java虚拟机和类加载器自动构造完成，本质上就是加载到内存中的运行时类。

20.2.2 获取Class对象的方式

- 使用数据类型.class的方式可以获取对应类型的Class对象（掌握）。
- 使用引用/对象.getClass()的方式可以获取对应类型的Class对象。
- 使用包装类.TYPE的方式可以获取对应基本数据类型的Class对象。
- 使用Class.forName()的方式来获取参数指定类型的Class对象（掌握）。
- 使用类加载器ClassLoader的方式获取指定类型的Class对象。

20.2.3 常用的方法（掌握）

方法声明	功能介绍
static Class<?> forName(String className)	用于获取参数指定类型对应的Class对象并返回
T newInstance()	用于创建该Class对象所表示类的新实例

20.3 Constructor类

20.3.1 基本概念

- java.lang.reflect.Constructor类主要用于描述获取到的构造方法信息

20.3.2 Class类的常用方法

方法声明	功能介绍
Constructor getConstructor(Class<?>... parameterTypes)	用于获取此Class对象所表示类型中参数指定的公共构造方法
Constructor<?>[] getConstructors()	用于获取此Class对象所表示类型中所有的公共构造方法

20.3.3 Constructor类的常用方法

方法声明	功能介绍
T newInstance(Object... initargs)	使用此Constructor对象描述的构造方法来构造Class对象代表类型的新实例
int getModifiers()	获取方法的访问修饰符
String getName()	获取方法的名称
Class<?>[] getParameterTypes()	获取方法所有参数的类型

20.4 Field类

20.4.1 基本概念

- java.lang.reflect.Field类主要用于描述获取到的单个成员变量信息。

20.4.2 Class类的常用方法

方法声明	功能介绍
Field getDeclaredField(String name)	用于获取此Class对象所表示类中参数指定的单个成员变量信息
Field[] getDeclaredFields()	用于获取此Class对象所表示类中所有成员变量信息

20.4.3 Field类的常用方法

方法声明	功能介绍
Object get(Object obj)	获取参数对象obj中此Field对象所表示成员变量的数值
void set(Object obj, Object value)	将参数对象obj中此Field对象表示成员变量的数值修改为参数value的数值
void setAccessible(boolean flag)	当实参传递true时，则反射对象在使用时应该取消 Java 语言访问检查
int getModifiers()	获取成员变量的访问修饰符
Class<?> getType()	获取成员变量的数据类型
String getName()	获取成员变量的名称

20.5 Method类

20.5.1 基本概念

- java.lang.reflect.Method类主要用于描述获取到的单个成员方法信息。

20.5.2 Class类的常用方法

方法声明	功能介绍
Method getMethod(String name, Class<?>... parameterTypes)	用于获取该Class对象表示类中名字为name参数为parameterTypes的指定公共成员方法
Method[] getMethods()	用于获取该Class对象表示类中所有公共成员方法

20.5.3 Method类的常用方法

方法声明	功能介绍
Object invoke(Object obj, Object... args)	使用对象obj来调用此Method对象所表示的成员方法，实参传递args
int getModifiers()	获取方法的访问修饰符
Class<?> getReturnType()	获取方法的返回值类型
String getName()	获取方法的名称
Class<?>[] getParameterTypes()	获取方法所有参数的类型
Class<?>[] getExceptionTypes()	获取方法的异常信息

20.6 获取其它结构信息

方法声明	功能介绍
Package getPackage()	获取所在的包信息
Class<? super T> getSuperclass()	获取继承的父类信息
Class<?>[] getInterfaces()	获取实现的所有接口
Annotation[] getAnnotations()	获取注解信息
Type[] getGenericInterfaces()	获取泛型信息