读书笔记 --Java 核心技术 -- 高级特征

作者：KioLuo
原文链接：https://ld246.com/article/1494604532254
来源网站：链滴

第一章--流与文件----------
流
读写字节
java.io.InputStream 1.0

<code>abstract int read() //从数据中读入一个字节，并返回该字节，在碰到流的结尾时返回-</code>
<code>int read(byte[] b) //读入一个字节数组，并返回实际读入的字节数，或者在碰到流的结时返回-1</code>
<code>int read(byte[] b, int off, int len) //读入一个字节数组。这个read方法返回实际读入字节数，或者在碰到流的结尾时返回-1</code>
<code>long skip(long n) //在输入流中跳过n个字节，返回实际跳过的字节数（如果碰到流的尾，则可能小于n）</code>
<code>int available() //返回在不阻塞的情况下可用的字节数</code>
<code>void close() //关闭这个输入流</code>
<code>void mark(int readLimit) //在输入流的当前位置打一个标记，如果输入流中已经读入字节多于readLimit个，则这个流允许忽略这个标记</code>
<code>void reset() //返回到最后的标记，随后对read的调用将重新读入这些字节</code></l
>
<code>boolean markSupported() //如果这个流支持打标记，则返回true</code>

java.io.OutputStream 1.0

<code>abstract void write(int n) //写出一个字节的数据</code>
<code>void write(byte[] b)</code>
<code>void write(byte[] b, int off, int len) //写出所有字节或者某个范围的字节到数组b中</ode>
<code>void close() //清空并关闭输出流</code>
<code>void flush() //清空输出流，也就是将所有缓冲的数据发送到目的地</code>

流家族
InputStream 和 OutputStream 为基础
DataInputStream 和 DataOutputStream 可以以二进制格式读写所有的基本 Java 类型
ZipInputStream 和 ZipOutputStream 可以以常见的 ZIP 压缩格式读写文件
4 个附加接口：Closeable、Flushable、Readable 和 Appendable
java.io.Closeable 5.0

<code>void close() //关闭这个Closeable，可能会抛出IOException</code>

java.io.Flushable 5.0

<code>void flush() //清空这个Flushable</code>

java.lang.Readable 5.0

<code>int read(CharBuffer cb) //尝试读入cb可以持有的数量的char值。返回读入的char值数量</code>

java.lang.Appendable 5.0

<code>Appendable append(char c)</code>
<code>Appendable append(CharSequence cs) //向这个Appendable中追加给定的码元或给定的序列中的所有码元，返回this</code>

java.lang.CharSequence 1.4

<code>char charAt(int index) //返回给定索引处的码元</code>
<code>int length() //返回这个序列中的码元的数量</code>
<code>CharSequence subSequence(int startIndex, int endIndex) //返回由存储在startIndex到endIndex - 1处的所有码元构成的CharSequence</code>
<code>String toString() //返回这个序列中所有码元构成的字符串</code>

组合流过滤器

<code>FileInputStream fin = new FileInputStream("employee.dat");</code>
<code>DataInputStream din = new DataInputStream(fin);</code>
<code>double s = din.readDouble();</code>

使用缓冲机制

<code>DataInputStream din = new DataInputStream(</code>
<code> new BufferedInputStream(</code>
<code> new FileInputStream("employee.dat")));</code>

从一个 ZIP 压缩文件中读入数字

<code>ZipInputStream zin = new ZipInputStream(new FileInputStream("employee.zip");</code>
<code>DataInputStream din = new DataInputStream(zin);</code>

java.io.FileInputStream 1.0

<code>FileInputStream(String name)</code>
<code>FileInputStream(File file) //使用由name字符串或file对象指定路径名的文件创建一个的文件输入流</code>

java.io.FileOutputStream 1.0

<code>FileOutputStream(String name)</code>
<code>FileOutputStream(String name, boolean append)</code>
<code>FileOutputStream(File file)</code>
<code>FileOutputStream(File file, boolean append) //</code> 使用由 name 字符串或 file 对象指定路径名的文件创建一个新的文件输出流

java.io.BufferedInputStream 1.0

<code>BufferedInputStream(InputStream in) //创建一个带缓冲区的流</code>

java.io.BufferdOutputStream 1.0

<code>BufferedOutputStream(OutputStream out) //创建一个带缓冲区的流</code>

java.io.PushbackInputStream 1.0

<code>PushbackInputStream(InputStream in)</code>
<code>PushbackInputStream(InputStream in, int size) //构建一个可以预览一个字节或者有指定尺寸的回推缓冲区的流</code>
void unread(int b) //回推一个字节，它可以在下次调用 read 时被再次获取，b:要再次读入的

节

文本输入与输出

<code>InputStreamReader in = new InputStreamReader(new FileInputStream("input.tx
"), "ISO8859_5");</code>
等价于
<code>FileReader in = new FileReader("input.txt"):</code>

以文本格式打印字符串和数字

<code>PrintWriter out = new PrintWriter("employee.txt"):</code>
<code>等同于</code>
<code>PrintWriter out = new PrintWriter(new FileWriter("employee.txt"));</code>

System.out 是 PrintStream 类，在内部采用与 PrintWriter 相同的方式将 Unicode 字符转换成默认的主机编码方式。与 PrintWriter 不同的是，它们允许我们用 write(int)和 write(byte[])方法输原生字节
java.io.PrintWriter 1.1

<code>PrintWriter(Writer out)</code>
<code>PrintWriter(Writer out, boolean autoFlush) //创建一个新的PrintWriter</code></l
>
<code>PrintWriter(OutputStream out)</code>
<code>PrintWriter(OutputStream out, boolean autoflush) //通过创建必需的中介OutputSt
eamWriter，从已有的OutputStream中创建一个新的PrintWriter</code>
<code>PrintWriter(String filename)</code>
<code>PrintWriter(File file) //通过创建必需的中介FileWriter，创建一个想给定的文件写出的
的PrintWriter</code>
<code>void print(Object obj) //通过打印从toString产生的字符串来打印一个对象</code></
i>
<code>void print(String s) //打印一个Unicode字符串</code>
<code>void println(String s) //打印一个字符串，后面紧跟一个行终止符</code>
<code>void print(char[] s) //打印给定的字符串中的所有Unicode字符</code>
<code>void print(char c) //打印一个Unicode字符</code>
<code>void print(int i)</code>
<code>void print(long l)</code>
<code>void print(float f)</code>
<code>void print(double d)</code>
<code>void print(boolean b) //以文本格式打印给定的值</code>
<code>void printf(String format, Object... args) //按照格式化字符串指定的方式打印给定的
</code>
<code>boolean checkError() //如果产生格式化或输出错误，返回true</code>

使用 Scanner 读入文本输入
Java SE 5.0 之前，处理文本的唯一方式就是 BufferedReader 类，它拥有一个 readLine 方法，以读入一行文本

<code>BufferedReader in = new BufferedReader(new FileReader("employee.txt"));</co
e>

字符集

<code>//编码Unicode字符串</code>

<code>Charset cset = Charset.forName("ISO-8859-1"):</code>
<code>String str = ...;</code>
<code>ByteBuffer buffer = cset.encode(str);</code>
<code>byte[] bytes = buffer.array();</code>
<code>//解码字节序列</code>
<code>byte[] bytes = ...;</code>
<code>ByteBuffer bbuf = ByteBuffer.wrap(bytes, offset, length);</code>
<code>CharBuffer cbuf = cset.decode(bbuf);</code>
<code>String str = cbuf.toString();</code>

java.nio.charset.Charset 1.4

<code>static SortedMap availableCharsets() //获取这个虚拟机可用的所有字符集。返回一映射表，它的键是字符集的名字，值是字符集</code>
<code>static Charset forName(String name) //获取给定名字的字符集</code>
<code>Set aliases() //返回这个字符集的别名集</code>
<code>ByteBuffer encode(String str) //将给定的字符串编码为字节序列</code>
<code>CharBuffer decode(ByteBuffer buffer) //解码给定的字节序列，无法识别的输入将被换成Unicode的“替代字符” ('\uFFFD')</code>

java.nio.ByteBuffer 1.4

<code>byte[] array() //返回这个缓冲区所管理的字节数组</code>
<code>static ByteBuffer wrap(byte[] bytes)</code>
<code>static ByteBuffer wrap(byte[] bytes, int offset, int length) //返回管理给定的字节组或给定字节数组的某个范围的字节缓冲区</code>

java.nio.CharBuffer

<code>char[] array() //返回这个缓冲区所管理的码元数组</code>
<code>char charAt(int index) //返回给定索引处的码元</code>
<code>String toString() //返回由这个缓冲区所管理的码元构成的字符串</code>

读写二进制数据
DataInputStream 类实现了 DataInput 接口
java.io.DataInput 1.0

<code>boolean readBoolean()</code>
<code>byte readByte()</code>
<code>char readChar()</code>
<code>double readDouble()</code>
<code>float readFloat()</code>
<code>int readInt()</code>
<code>long readLong()</code>
<code>short readShort() //读入一个给定类型的值</code>
<code>void readFully(byte[] b) //将字节读入到数组b中，其间阻塞直至所有字节都读入</code>
<code>void readFully(byte[] b, int off, int len) //将字节读入到数组b中，其间阻塞直至所字节都读入</code>
<code>String readUTF() //读入由“修订过的UTF-8”格式的字符构成的字符串</code>
<code>int skipBytes(int n) //跳过n个字节，其间阻塞直至所有字节都被跳过</code>

java.io.DataOutput 1.0

<code>void writeBoolean(boolean b)</code>
<code>void writeByte(int b)</code>
<code>void writeChar(int c)</code>
<code>void writeDouble(double d)</code>
<code>void writeFloat(float f)</code>
<code>void writeInt(int i)</code>
<code>void writeLong(long l)</code>
<code>void writeShort(int s) //写出一个给定类型的值</code>
<code>void writeChars(String s) //写出字符串中的所有字符</code>
<code>void writeUTF(String s) //写出由“修订过的UTF-8”格式的字符构成的字符串</code

随机访问文件
RandomAccessFile 类可以在文件中的任何位置查找或写入数据，通过“r"或"rw"来指定。同时
现了 DataInput 和 DataOutput 接口
java.io.RandomAccessFile 1.0

<code>RandomAccessFile(String file, String mode)</code>
<code>RandomAccessFile(File file, String mode) //file:要打开的文件，mode:"r"表示只读
式，"rw"表示读/写模式</code>
<code>long getFilePointer() //返回文件指针的当前位置</code>
<code>void seek(long pos) //将文件指针从文件的开始设置到pos个字节处</code>
<code>long length() //返回文件按照字节来度量的长度</code>

ZIP 文档
通读 ZIP 文件代码：

<code>ZipInputStream zin = new ZipInputStream(new FileInputStream(zipname));</co
e>
<code>ZipEntry entry;</code>
<code>while ((entry = zin.getNextEntry()) != null)</code>
<code>{</code>
<code> analyze entry;</code>
<code> read the contents of zin;</code>
<code> zin.closeEntry();</code>
<code>}</code>
<code>zin.close();</code>

写出 ZIP 文件代码：

<code>FileOutputStream fout = new FileOutputStream("test.zip");</code>
<code>ZipOutputStream zout = new ZipOutputStream(fout);</code>
<code>for all files</code>
<code>{</code>
<code> ZipEntry ze = new ZipEntry(filename);</code>
<code> zout.putNextEntry(ze);</code>
<code> send data to zout;</code>
<code> zout.closeEntry();</code>
<code>}</code>
<code>zout.close();</code>

java.util.zip.ZipInputStream 1.1

<code>ZipInputStream(InputStream in) //创建一个ZipInputStream</code>

<code>ZipEntry getNextEntry() //为下一项返回ZipEntry对象，否则没有更多的项时返回null
/code>
<code>void closeEntry() //关闭这个ZIP文件中当前打开的项</code>

java.util.zip.ZipOutputStream 1.1

<code>ZipOutputStream(OutputStream out) //创建一个将压缩数据写出到指定的OutputStr
am的ZipOutputStream</code>
<code>void putNextEntry(ZipEntry ze) //将给定的ZipEntry中的信息写出到流中，并定为用
写出数据的流，然后这些数据可以通过write()写出到这个流中</code>
<code>void closeEntry() //关闭这个ZIP文件中当前打开的项</code>
<code>void setLevel(int level) //设置后续的各个DEFAULTED项的默认压缩级别。</code></
i>
<code>void setMethod(int method) //设置用于这个ZipOutputStream的默认压缩方法，压
方法：DEFLATED或STORED</code>

java.util.zip.ZipEntry 1.1

<code>ZipEntry(String name) //这一项的名字</code>
<code>long getCrc() //返回用于这个ZipEntry的CRC32校验和的值</code>
<code>String getName() //返回这一项的名字</code>
<code>long getSize() //返回这一项不被压缩的大小</code>
<code>boolean isDirectory() //当这一项是目录时返回true</code>
<code>void setSize(long size) //设置这一项的大小，当压缩方法是STORED时才必需</code

<code>void setCrc(long crc) //给这一项设置CRC32校验和，这个校验和是使用CRC32类计
的，STORED时必需</code>

java.util.zip.ZipFile 1.1

<code>ZipFile(String name)</code>
<code>ZipFile(File file) //创建一个ZipFile，用于从给定的字符串或File对象中读入数据</cod
>
<code>Enumeration entries() //返回一个Enumeration对象，它枚举了描述这个ZipFile中各
项的ZipEntry对象</code>
<code>ZipEntry getEntry(String name) //返回给定名字所对应的项，或者在没有对应项的时
返回null</code>
<code>InputStream getInputStream(ZipEntry ze) //返回用于给定项的InputStream</code

<code>String getName() //返回这个ZIP文件的路径</code>

对象流与序列化
对象序列化（object serialization)
ObjectInputStream 和 ObjectOutputStream 类，readObject 和 writeObject 方法
在对象流中存储或恢复的所有类都需要实现 Serializable 接口
java.io.ObjectOutputStream 1.1

<code>ObjectOutputStream(OutputStream out) //创建一个ObjectOutputStream使得你
以将对象写出到指定的OutputStream</code>
<code>void writeObject(Object obj) //写出指定的对象到ObjectOutputStream，这个方法
存储指定对象的类、类的签名以及这个类及其超类中所有非静态和非瞬时的域的值</code>

java.io.ObjectInputStream 1.1

<code>ObjectInputStream(InputStream in) //创建一个ObjectInputStream用于从指定的InutSream中读回对象信息</code>
<code>Object readObject() //从ObjectInputStream中读入一个对象</code>

对象流输出中包含所有对象的类型和数据域
每个对象都被赋予一个序列号
相同对象的重复出现将被存储为对这个对象的序列号的引用
修改默认的序列化机制
防止不应该被序列化的域被序列化，标记成 transient
可序列化的类可以定义下列方法：

<code>private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException;</code>
<code>private void writeObject(ObjectOutputStream out) throws IOException;</code>

之后，数据域就不再自动序列化，而是调用这些方法
类还可以定义它自己的机制，必须实现 Externalizable 接口，定义两个方法：

<code>public void readExternal(ObjectInputStream in) throws IOException, ClassNotFundException;</code>
<code>public void writeExternal(ObjectOutputStream out) throws IOException;</code>

序列化单例和类型安全的枚举
请记住向遗留代码中所有类型安全的枚举以及向所有支持单例设计模式的类中添加 readResolve 方法
版本管理
运行 JDK 中的单机程序 serialver
serialver Employee
这个类的所有版本都需要定义
public static final long serialVersionUID = -1845284363289478 93L;
文件管理
File 类
可以用 FileNameFilter 对象作为 list 方法的参数来减小列表长度，一个实现 FileNameFilter 接的类需要定义 accept 方法
使用 File 类中存储在名为 separator 的静态实例域中的有关当前目录分隔符的信息可以得到系恰当的分隔符
java.io.File 1.0

<code>boolean canRead()</code>
<code>boolean canWrite()</code>
<code>boolean canExecute() //表明文件是否可读、可写或可执行</code>
<code>boolean setReadable(boolean state, boolean ownerOnly)</code>
<code>boolean setWritable(boolean state, boolean ownerOnly)</code>
<code>boolean setExecutable(boolean state, boolean ownerOnly) //设置这个文件的可、可写或可执行状态。如果ownerOnly为true，状态设置只对文件拥有者有效，否则，对所有人有效这些方法在设置状态成功后返回true</code>
<code>static boolean createTempFile(String prefix, String suffix)</code>
<code>static boolean createTempFile(String prefix, String suffix, File directory) //在系的默认临时目录或给定目录中创建一个临时文件，并使用给定的前缀或后缀来生成文件名</code></l>
<code>boolean delete() //尝试删除这个文件</code>
<code>void deleteOnExit() //请求在虚拟机关闭时将文件删除</code>

<code>boolean exists() //如果目录存在则返回true</code>
<code>String getAbsolutePath() //返回包含绝对路径名的字符串，应该用getCanonicalPat
代替它</code>
<code>File getCanonicalFile() //返回包含这个文件的规范路径名的File对象</code>
<code>String getCanonicalPath() //返回包含这个文件的规范路径名的字符串</code>
<code>String getName() //返回包含这个File对象的文件名的字符串（不包含路径）</code>
/li>
<code>String getParent() //返回这个File对象的父亲名字的字符串</code>
<code>File getParentFile() //返回这个File目录的父目录的File对象</code>
<code>String getPath() //返回包含这个文件的路径名的字符串</code>
<code>boolean isDirectory() //如果这个File对象表示一个文件而不是一个目录或一个设备，
返回true</code>
<code>boolean isFile() //如果这个File对象表示一个文件而不是一个目录或一个设备，则返回t
ue</code>
<code>boolean isHidden() //如果这个File对象表示的是一个隐藏文件或目录，则返回true</c
de>
<code>long lastModified() //返回这个文件最后被修改的时间（毫秒数）</code>
<code>String[] list()</code>
<code>String[] list(FilenameFilter filter) //返回由这个File对象包含的满足过滤器条件的文件
和目录名构成的字符串数组</code>
<code>File[] listFiles() </code>
<code>File[] listFiles(FilenameFilter filter) //返回由这个File对象包含的文件和目录所对应的Fi
e对象构成的数组</code>
<code>static File[] listRoots() //返回由所有可获得的文件根对应的File对象构成的数组</cod
>
<code>boolean createNewFile() //自动创建一个由File对象给定名字的新文件</code>
<code>boolean mkdir() //创建一个由这个File对象给定名字的子目录</code>
<code>boolean mkdirs() //与mkdir不同，这个方法在必要时将创建父目录</code>
<code>boolean renameTo(File newName) //如果文件名被修改，则返回true</code>
<code>boolean setLastModified(long time) //设置这个文件的最后修改时间</code>
<code>boolean setReadOnly() //将这个文件设置成只读</code>
<code>URL toURL() //将这个File对象转换成一个文件的URL</code>
<code>long getTotalSpace()</code>
<code>long getFreeSpace()</code>
<code>long getUsableSpace() //获得由File对象所描述的分区的总大小、未分配字节的数量
可用字节的数量</code>

java.io.FilenameFilter 1.0

<code>boolean accept(File dir，String name) //应该定义为在文件能够匹配过滤器标准时返
true</code>

新 IO
内存映射文件
java.io.FileInputStream 1.0

<code>FileChannel getChannel() //返回用于访问这个流的通道</code>

java.io.FileOutputStream 1.0

<code>FileChannel getChannel() //返回用于访问这个流的通道</code>

java.io.RandomAccessFile 1.0

<code>FileChannel getChannel() //返回用于访问这个流的通道</code>

java.nio.channels.FileChannel 1.4

<code>MappedByteBuffer map(FileChannel.MapMode mode, long position, long size) /将文件的一个区域映射到内存中</code>

java.nio.Buffer 1.4

<code>boolean hasRemaining() //如果当前的缓冲区位置没有达到这个缓冲区的界限位置则回true</code>
<code>int limit() //返回这个缓冲区的界限位置</code>

java.nio.ByteBuffer 1.4

<code>byte get() //从当前位置获得一个字节，并将当前位置推到下一个字节</code>
<code>byte get(int index) //从指定索引处获得一个字节</code>
<code>ByteBuffer put(byte b) //向当前位置推入一个字节，并将当前位置推到下一个字节，回对这个缓冲区的引用</code>
<code>ByteBuffer put(int index, byte b) //向指定索引处推入一个字节，返回对这个缓冲区引用</code>
<code>ByteBuffer get(byte[] destination)</code>
<code>ByteBuffer get(byte[] destination, int offset, int length) //用缓冲区的字节来填充节数组，或者字节数组的某个区域，并将当前位置向前推读入的字节数个位置，如果缓冲区不够，不读入任何字节，并抛出BufferUnderflowException</code>
<code>ByteBuffer put(byte[] source)</code>
<code>ByteBuffer put(byte[] source, int offset, int length) //将字节数组中的所有字节或给定区域的字节都推入缓冲区，并将当前位置向前推写出的字节数个位置</code>
<code>Xxx getXxx()</code>
<code>Xxx getXxx(int index)</code>
<code>ByteBuffer putXxx(xxx value)</code>
<code>ByteBuffer putXxx(int index, xxx value) //获得或放置一个二进制数</code>
<code>ByteBuffer order(ByteOrder order)</code>
<code>ByteOrder order() //设置或获得字节顺序，order的值是ByteOrder类的常量BIG_ENDAN或LITTLE_ENDIAN中的一个</code>

缓冲区数据结构
java.nio.Buffer 1.4

<code>Buffer clear() //通过将位置复位到0，并将界限复位到容量，使这个缓冲区为写出做好备。返回this</code>
<code>Buffer flip() //通过将界限设置为位置，并将位置复位到0，使这个缓冲区为读入做好准，返回this</code>
<code>Buffer rewind() //通过将读写位置复位到0，并保持极限不变，使这个缓冲区为重新读相同的值做好准备，返回this</code>
<code>Buffer mark() //将这个缓冲区的标记设置到读写位置，返回this</code>
<code>Buffer reset() //将这个缓冲区的位置设置到标记，从而允许被标记的部分可以再次被入或写出，返回this</code>
<code>int remaining() //返回剩余可读入或可写出的值的数量，即界限与位置之间的差异</code>
<code>int position() //返回这个缓冲区的位置</code>
<code>int capacity() //返回这个缓冲区的容量</code>

java.nio.CharBuffer 1.4

<code>char get()</code>
<code>CharBuffer get(char[] destination)</code>
<code>CharBuffer get(char[] destination, int offset, int length) //从这个缓冲区的位置处始，获得一个char值，或者某个范围的char值，然后将位置向前推过所读入的字符，最后两个方法返this</code>
<code>CharBuffer put(char c)</code>
<code>CharBuffer put(char[] source)</code>
<code>CharBuffer put(char[] source, int offset, int length)</code>
<code>CharBuffer put(String source)</code>
<code>CharBuffer put(CharBuffer source) //从这个缓冲区的位置处开始，推入一个char值或者某个范围的char值，然后将位置向前推过所写出的字符</code>
<code>CharBuffer read(CharBuffer destination) //从这个缓冲区中获得char值，然后将它们入目标缓冲区，直至达到目标缓冲区的界限</code>

文件加锁机制
文件加锁机制是依赖于操作系统的
文件锁是由整个 Java 虚拟机持有的。如果有两个程序是由同一个虚拟机启动的，那么它们不可每一个都获得一个在同一个文件上的锁，当调用 lock 和 tryLock 方法时，如果虚拟机已经在同一个件上持有了另一个重叠的锁，那么这两个方法将抛出 OverlappingFileLockException
java.nio.channels.FileChannel 1.4

<code>FileLock lock() //在整个文件上获得一个独占的锁，这个方法将阻塞直至获得锁</code>
<code>FileLock tryLock() //在整个文件上获得一个独占的锁，或者在无法获得锁的情况下返回ull</code>
<code>FileLock lock(long position, long size, boolean shared)</code>
<code>FileLock tryLock(long position, long size, boolean shared) //在文件的一个区域上得锁</code>

java.nio.channels.FileLock 1.4

<code>void release() //释放这个锁</code>

正则表达式
语法

<code>String patternString = "]*)\\s*>"</code>

用法

<code>Pattern pattern = Pattern.compile(patternString);</code>
<code>Matcher matcher = pattern.matcher(input);</code>
<code>if (matcher.matches()) ...</code>

java.util.regex.Pattern 1.4

<code>static Pattern compile(String expression)</code>
<code>static Pattern compile(String expression, int flags) //把正则表达式字符串编译到个用于快速处理匹配的模式对象中</code>
<code>Matcher matcher(CharSequence input) //返回一个matcher对象，可以用它在输入位模式的匹配</code>
<code>String[] split(CharSequence input)</code>
<code>String[] split(CharSequence input, int limit) //将输入分割成标号，其中模式指定了

</code>

隔符的形式</code>

java.util.regex.Matcher 1.4

<code>boolean matches() //如果输入匹配模式，则返回true</code>
<code>boolean lookingAt() //如果输入的开头匹配模式，则返回true</code>
<code>boolean find()</code>
<code>boolean find(int start) //尝试查找下一个匹配，找到返回true</code>
<code>int start()</code>
<code>int end() //返回当前匹配的开始索引和结尾之后的索引</code>
<code>String group() //返回当前的匹配</code>
<code>int groupCount() //返回输入模式中的群组数量</code>
<code>int start(int groupIndex)</code>
<code>int end(int groupIndex) //返回当前匹配中给定群组的开始和结尾之后的位置</code

<code>String group(int groupIndex) //返回匹配给定群组的字符串</code>
<code>String replaceAll(String replacement)</code>
<code>String replaceFirst(String replacement) //返回从匹配器输入获得的通过将所有匹配
第一个匹配用替换字符串替换之后的字符串</code>
<code>Matcher reset()</code>
<code>Matcher reset(CharSequence input) //复位匹配器的状态。第二个方法将使匹配器作
于另一个不同的输入，两个方法都返回this</code>

第二章--XML----------
XML 文档
文档头
<pre><code class="highlight-chroma"><span class="highlight
cl"><?xml version="1.0" encoding="UTF-8"?>
</code></pre>
文档类型定义（DTD，Document Type Definition）
<pre><code class="highlight-chroma"><span class="highlight
cl"><!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN
"http:java.sun.com/j2ee/dtds/web-app_2_2.dtd">
</code></pre>
通常属性只应该在修改值的解释时使用，而不是在指定值时使用
字符引用：&#233 （十进制）; &#xD9（十六进制）
实体引用：
<pre><code class="highlight-chroma"><span class="highlight
cl"> < 小于
  > 大于
  & &
mp;
  " 引

  ' 省
号
</code></pre>
CDATA 部分: 用和来限定其界限，可用来包含含有 <、>、&之类字符的字符串
处理指令：用限定界限
注释：用 <!-和--> 限定界限
解析 XML 文档
DOM（Document Object Model，文档对象模型）解析器：树型解析器
用于 XML 的简单 API（Simple API for XML, SAX)解析器：流机制解析器
读入一个 XML 文档

<code>DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();</code>
<code>DocumentBuilder builder = factory.newDocumentBuilder();</code>
<code>File f = ...</code>
<code>Document doc = builder.parse(f);</code>

javax.xml.parsers.DocumentBuilderFactory 1.4

<code>static DocumentBuilderFactory newInstance() //返回DocumentBuilderFactory类一个实例</code>
<code>DocumentBuilder newDocumentBuilder() //返回DocumentBuilder类的一个实例</ode>

javax.xml.parsers.DocumentBuilder 1.4

<code>Document parse(File f)</code>
<code>Document parse(String url)</code>
<code>Document parse(InputStream in) //解析来自给定文件、URL或输入流的XML文档，回解析后的文档</code>

org.w3c.dom.Document 1.4

<code>Element getDocumentElement() //返回文档的根元素</code>

org.w3c.dom.Element 1.4

<code>String getTagName() //返回元素的名字</code>
<code>String getAttribute(String name) //返回给定名字的属性值，没有该属性时返回空字串</code>

org.w3c.dom.Node 1.4

<code>NodeList getChildNodes() //返回包含所有子元素节点的节点列表</code>
<code>Node getFirstChild()</code>
<code>Node getLastChild() //获取该节点的第一个或最后一个子节点，在该节点没有子节点返回null</code>
<code>Node getNextSibling()</code>
<code>Node getPreviousSibling() //获取该节点的下一个或上一个兄弟节点，在该节点没有弟节点时返回null</code>
<code>Node getParentNode() //获取该节点的父结点，在该节点是文档节点时返回null</coe>
<code>NamedNodeMap getAttributes() //返回含有描述该节点所有属性的Attr节点的映射</code>
<code>String getNodeName() //返回该节点的名字，当该结点是Attr节点时，该名字就是属名</code>
<code>String getNodeValue() //返回该节点的值，当该节点是Attr节点时，该值就是属性值<code>

org.w3c.dom.CharacterData 1.4

<code>String getData() //返回存储在节点中的文本</code>

org.w3c.dom.NodeList 1.4

<code>int getLength() //返回列表中的节点数</code>
<code>Node item(int index) //返回给定索引号的节点</code>

org.w3c.dom.NamedNodeMap 1.4

<code>int getLength() //返回该节点映射表中的节点数</code>
<code>Node item(int index) //返回给定索引号的节点</code>

验证 XML 文档
DTD
<pre><code class="highlight-chroma"><span class="highlight
cl"><!ELEMENT font (name,size)>
</code></pre>
XML Schema （xsd 文件）
<pre><code class="highlight-chroma"><span class="highlight
cl"><xsd:element name="font">
  <xsd:sequen
e>
  <xsd:elem
nt name="name" type="xsd:string" />
  <xsd:elem
nt name="size" type="xsd:int" />
  </xsd:sequen
e>
</xsd:element
gt;
</code></pre>
文档类型定义：
将 DTD 纳入 XML 文档中
<pre><code class="highlight-chroma"><span class="highlight
cl"><?xml version="1.0"?>
<!DOCTYPE con
iguration [
  <!ELEMENT c
nfiguration ...>
  more rules
  ...
]>
<configuration
gt;
...
</configuratio
>
</code></pre>
将 DTD 存储在外面
<pre><code class="highlight-chroma"><span class="highlight
cl"><!DOCTYPE configuration SYSTEM "config.dtd">
</code></pre>
或者
<pre><code class="highlight-chroma"><span class="highlight
cl"><!DOCTYPE configuration SYSTEM "http://myserver.com/config.dtd">
</code></pre>
如果使用 DTD 的相对 URL（比如"config.dtd"），需要给解析器一个文件或 URL 对象，而不是

nputStream，如果必须从一个输入流来解析，请提供一个实体渲染器
来源于 SGML 的用于识别“众所周知的”DTD 的机制，如
<pre><code class="highlight-chroma"><!DOCTYPE web-app
 PUBLIC "-//Sun icrosystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2 dtd" >
</code></pre>
如果使用的是 DOM 解析器，想要支持公共标识符，需要调用 DocumentBuilder 类的 setEntity esolver 方法来安装 EntityResolver 接口的某个实现类的一个对象，该接口只有一个方法：resolveEnt ty
ELEMENT 规则
E*　0 或多个 E
E+　1 或多个 E
E?　0 或 1 个 E
E1|E2|...|En　E1, E2, ..., En 中的一个
E1, E2, ..., En　E1 随后是 E2，...，En
#PCDATA　文本
(#PCDATA|E1|E2|...|En)*　0 或多个任意顺序的文本和 E1，E2，...，En（混合式内容）
ANY　允许任意子元素
EMPTY　不允许有子元素
元素的规范可以包含嵌套的和复杂的正则表达式，如
<pre><code class="highlight-chroma"><!ELEMENT chapter (intro, (heading, (para | image | table | note)+)+)
</code></pre>
描述合法元素属性的语法规则：
<pre><code class="highlight-chroma"><!ATTLIST element attribute type default>
</code></pre>
两个属性规范范例：
<pre><code class="highlight-chroma"><!ATTLIST font style (plain|bold|italic|bold-italic) "plain">
<!ATTLIST size nit CDATA #IMPLIED>
</code></pre>
使用 DTD 验证输入

<code>factory.setValidating(true);</code>
<code>factory.setIgnoringElementContentWhitespace(true);</code>

在验证时，应该安装一个错误处理器，即一个实现了 ErrorHandler 接口的对象
javax.xml.parsers.DocumentBuilder 1.4

<code>void setEntityResolver(EntityResolver resolver) //设置解析器，来定位要分析的XM 文档中引用的实体</code>
<code>void setErrorHandler(ErrorHandler handler) //设置报告解析过程中出现的错误和警 的处理器</code>

org.xml.sax.EntityResolver 1.4

<code>public InputSource resolveEntity(String publicID, String systemID) //返回包含被定ID引用数据的一个输入源，或者，当解析器不知道如何解析某个特定名字时，返回null。如果没有 供公共ID，那么参数publicID可以为null</code>


```html
<p><strong>org.xml.sax.InputSource 1.4</strong></p>
<ol>
<li><code>InputSource(InputStream in)</code></li>
<li><code>InputSource(Reader in)</code></li>
<li><code>InputSource(String systemID) //根据流、读入器、或系统ID（通常是相对或绝对URL
构建输入源</code></li>
</ol>
<p><strong>org.xml.sax.ErrorHandler 1.4</strong></p>
<ol>
<li><code>void fatalError(SAXParseException exception)</code></li>
<li><code>void error(SAXParseException exception)</code></li>
<li><code>void warning(SAXParseException exception) //覆盖这些方法以提供处理器，对致命
误、非致命错误和警告进行处理</code></li>
</ol>
<p><strong>org.xml.sax.SAXParseException 1.4</strong></p>
<ol>
<li><code>int getLineNumber()</code></li>
<li><code>int getColumnNumber() //返回引起异常的已处理的输入信息末尾的行号和列号</cod
></li>
</ol>
<p><strong>javax.xml.parsers.DocumentBuilderFactory 1.4</strong></p>
<ol>
<li><code>boolean isValidating()</code></li>
<li><code>void setValidating(boolean value) //获取和设置工厂的validating属性，当设为true
，工厂生成的解析器会验证它们的输入信息</code></li>
<li><code>boolean isIgnoringElementContentWhiteSpace(boolean value) //获取和设置工厂的
gnoringElementContentWhiteSpace属性，当设为true时，工厂生成的解析器会忽略没有混合内容
即元素与#PCDATA混合）的元素节点之间的空白</code></li>
</ol>
<p><strong>XML Schema</strong></p>
<p>如果要在文档中引用 Schema 文件，需要在根元素中加上属性</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">&lt;?xml version="1.0"?&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;configuration
mlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
</span></span><span class="highlight-line"><span class="highlight-cl">    xsi:noNamespa
eSchemaLocation="config.xsd"&gt;
</span></span><span class="highlight-line"><span class="highlight-cl">...
</span></span><span class="highlight-line"><span class="highlight-cl">&lt;/configuratio
&gt;
</span></span></code></pre>
<p>解析带有 Schema 的 XML 文件和解析带有 DTD 的文件相似，但有 3 点差别：<br>
//必须打开对命名空间的支持</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">factory.setNamespaceAware(true);
</span></span></code></pre>
<p>//必须执行如下步骤</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">final String JAXP_SCHEMA_LANGUAGE = "http://java.sun.com/xml/jaxp/properties/schem
Language";
</span></span><span class="highlight-line"><span class="highlight-cl">final String W3C_
ML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
</span></span><span class="highlight-line"><span class="highlight-cl">factory.setAttribut
(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

</code></pre>
使用 XPath 来定位信息
XPath 表达式/configuration/database/username
使用 @ 操作符可以得到属性值，如
/gridbag/row[1]/cell[1]/@anchor
javax.xml.xpath.XPathFactory 5.0

<code>static XPathFactory newInstance() //返回XPathFactory实例来创建XPath对象</code>
<code>XPath newXpath() //构建XPath对象来计算XPath表达式</code>

javax.xml.xpath.XPath 5.0

<code>String evaluate(String expression，Object startingPoint) /从给定的起点计算表达，起点可以是一个节点或节点列表，如果结果是一个节点或节点集，则返回的字符串包含所有文本节子元素的数据</code>
<code>Object evaluate(String expression，Object startingPoint，QName resultType) //给定的起点计算表达式，起点可以是一个节点或节点列表。ResultType是XPathConstants类的常量SRING, NODE, NODESET, NUMBER或BOOLEAN之一</code>

使用命名空间（URI）
HTTP URL 格式最常用
xmlns:alias="namespaceURI" 用于定义命名空间和别名
打开命名空间处理特性

<code>factory.setNamespaceAware(true);</code>

由 getNodeName 和 getTagName 等方法返回带有别名前缀的限定名
org.w3c.dom.Node 1.4

<code>String getLocalName() //返回本地名（不带别名前缀），或者在解析器不支持命名空时返回null</code>
<code>String getNamespaceURI() //返回命名空间URI，或者在解析器不支持命名空间时返回ull</code>

javax.xml.parsers.DocumentBuilderFactory 1.4

<code>boolean isNamespaceAware()</code>
<code>void setNamespaceAware(boolean value) //获取或设置工厂的namespaceAware性</code>

流机制解析器
使用 SAX 解析器（事件回调）
在使用 SAX 解析器时，需要一个处理器来定义不同的解析器事件的事件动作，ContentHandler接口定义了若干个回调方法，如 startElement, endElement, characters, startDocument 和 endDoument

<code>SAXParserFactory factory = SAXParserFactory.newInstance();</code>
<code>SAXParser parser = factory.newSAXParser();</code>
<code>parser.parse(source, handler);</code>

处理器属于 DefaultHandler 的一个子类，DefaultHandler 类为 4 个接口定义了空的方法：ContentHandler, DTDHandler, EntityResolver, ErrorHandler

javax.xml.parsers.SAXParserFactory 1.4

<code>static SAXParserFactory newInstance() //返回SAXParserFactory类的一个实例</co
e>
<code>SAXParser newSAXParser() //返回SAXParser类的一个实例</code>
<code>boolean isNamespaceAware()</code>
<code>void setNamespaceAware(boolean value) //获取和设置工厂的namespaceAware
性</code>
<code>boolean isValidating()</code>
<code>void setValidating(boolean value) //获取和设置工厂的validating属性</code>

javax.xml.parsers.SAXParser 1.4

<code>void parse(File f, DefaultHandler handler)</code>
<code>void parse(String url, DefaultHandler handler)</code>
<code>void parse(InputStream in, DefaultHandler handler) //解析来自给定文件、URL或
入流的XML文档，并把解析事件报告给指定的处理器</code>

org.xml.sax.ContentHandler 1.4

<code>void startDocument()</code>
<code>void endDocument()</code>
<code>void startElement(String uri, String lname, String qname, Attributes attr)</code

<code>void endElement(String uri, String lname, String qname) //在元素的起始或结束
被调用</code>
<code>void characters(char[] data, int start, int length) //解析器报告字符数据时被调用</
ode>

org.xml.sax.Attributes 1.4

<code>int getLength() //返回存储在该属性集合中属性数量</code>
<code>String getLocalName(int index) //返回给定索引的属性的本地名（无别名前缀），或
不支持命名空间特性时返回空字符串</code>
<code>String getURI(int index) //返回给定索引的属性的命名空间URI</code>
<code>String getQName(int index) //返回给定索引的属性的限定名</code>
<code>String getValue(int index)</code>
<code>String getValue(String qname)</code>
<code>String getValue(String uri, String lname) //根据给定索引、限定名或命名空间URI+
地名，返回属性值，该值不存在时返回null</code>

使用 StAX 解析器（提供解析事件的迭代器）

<code>InputStream in = url.openStream();</code>
<code>XMLInputFactory factory = XMLInputFactory.newInstance();</code>
<code>XMLStreamReader parser = factory.createXMLStreamReader(in);</code>
<code>while (parser.hasNext()) {</code>
<code> int event = parser.next();</code>
<code> Call parser methods to obtain event details</code>
<code>}</code>

javax.xml.stream.XMLInputFactory 6

<code>static XMLInputFactory newInstance() //返回XMLInputFactory类的一个实例</cod

>
<code>void setProperty(String name，Object value）//设置这个工厂的属性，或者在要设的属性不支持设置成给定值时，抛出IllegalArgumentException</code>
<code>XMLStreamReader createXMLStreamReader(InputStream in)</code>
<code>XMLStreamReader createXMLStreamReader(InputStream in，String characterEncding)</code>
<code>XMLStreamReader createXMLStreamReader(Reader in)</code>
<code>XMLStreamReader createXMLStreamReader(Source in）//创建一个从给定的流、读器或JAXP源读入的解析器</code>

javax.xml.stream.XMLStreamReader 6

<code>boolean hasNext()//如果有另一个解析事件则返回true</code>
<code>int next()//将解析器的状态设置为下一个解析事件，并返回下列常量之一：START_ELMENT、CHARACTERS、START_DOCUMENT、END_DOCUMENT、CDATA、COMMENT、SPAC（可忽略的空白字符）、PROCESSING_INSTRUCTION、ENTITY_REFERENCE、DTD</code>
<code>boolean isStartElement()</code>
<code>boolean isEndElement()</code>
<code>boolean isCharacters()</code>
<code>boolean isWhiteSpace()//如果当前事件是一个开始元素、结束元素、字符数据或空字符，返回true</code>
<code>QName getName()</code>
<code>String getLocalName()//获取在START_ELEMENT或END_ELEMENT事件中的元素的字</code>
<code>String getText()//返回一个CHARACTERS、COMMENT或CDATA事件，或一个ENTIY_REFERENCE的替换值，或者一个DTD的内部子集所对应的字符</code>
<code>int getAttributeCount()</code>
<code>QName getAttributeName(int index)</code>
<code>String getAttributeLocalName(int index)</code>
<code>String getAttributeValue(int index)</code>
<code>String getAttributeValue(String namespaceURI，String name）//只要当前事件是SART_ELEMENT，则获取给定属性的值，如果namespaceURI为null，则不检查名字空间</code></l>

生成 XML 文档
用文档的内容构建一棵 DOM 树

<code>Document doc = builder.newDocument();</code>
<code>Element rootElement = doc.createElement(rootName);</code>
<code>Element childElement = doc.createElement(childName);</code>
<code>Text textNode = doc.createTextNode(textContents);</code>
<code>doc.appendChild(rootElement);</code>
<code>rootElement.appendChild(childElement);</code>
<code>childElement.appendChild(textNode);</code>
<code>rootElement.setAttribute(name, value);</code>

但是 DOM API 目前还不支持 DOM 树写到输出流，需要使用可扩展的格式页转换（XSLT) API

<code>//construct the "do nothing" transformation</code>
<code>Transformer t = TransformerFactory.newInstance().newTransformer();</code></l>
<code>//set output properties to get a DOCTYPE node</code>
<code>t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, systemIdentifier);</code></l

>
<code>t.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, publicIdentifier);</code>

<code>//set indentation</code>
<code>t.setOutputProperty(OutputKeys.INDENT, "yes");</code>
<code>t.setOutputProperty(OutputKeys.METHOD, "xml");</code>
<code>//apply the "do nothing" transformation and send the output to a file</code></l
>
<code>t.transform(new DOMSource(doc), new StreamResult(new FileOutputStream(fil
)));</code>

javax.xml.parsers.DocumentBuilder 1.4

<code>Document newDocument() //返回一个空文档</code>

org.w3c.dom.Document 1.4

<code>Element createElement(String name) //返回具有给定名字的元素</code>
<code>Text createTextNode(String data) //返回具有给定数据的文本节点</code>

org.w3c.dom.Node 1.4

<code>Node appendChild(Node child) //将一个节点附加到该节点的子节点列表，返回该节
</code>

org.w3c.dom.Element 1.4

<code>void setAttribute(String name, String value) //将有给定名字的属性设置为指定的值
/code>
<code>void setAttributeNS(String uri, String qname, String value) //将带有给定命名空间
RI和限定名的属性设置为指定的值</code>

javax.xml.transform.TransformerFactory 1.4

<code>static TransformerFactory newInstance() //返回TransformerFactory类的一个实例<
code>
<code>transformer newTransformer() //返回Transformer类的一个实例，带有标识符或“
操作”的转换</code>

javax.xml.transform.Transformer 1.4

<code>void setOutputProperty(String name, String value) //设置输出属性</code>
<code>void transform(Source from, Result to) //转换一个XML文档</code>

javax.xml.transform.dom.DOMSource 1.4

<code>DOMSource(Node n) //根据指定的节点构建一个源。通常，n是文档节点</code></l
>

javax.xml.transform.stream.StreamResult 1.4

<code>StreamResult(File f)</code>
<code>StreamResult(OutputStream out)</code>
<code>StreamResult(Writer out)</code>

<code>StreamResult(String systemID)根据文件、流、写入程序或系统ID（通常是相对或绝对RL）来构建数据流结果</code>

使用 StAX 写 XML 文档

<code>XMLOutputFactory factory = XMLOutputFactory.newInstance();</code>
<code>XMLStreamWriter writer = factory.createXMLStreamWriter(out);</code>
<code>writer.writeStartDocument();</code>
<code>writer.writeStartElement(name);</code>
<code>writer.writeAttribute(name, value);</code>
<code>writer.writeCharacters(text);</code>
<code>writer.writeEndElement();</code>
<code>writer.writeEmptyElement(name);</code>
<code>writer.writeEndDocument();</code>

javax.xml.stream.XMLOutputFactory 6.

<code>static XMLOutputFactory newInstance() //返回这个XMLOutputFactory类的一个例</code>
<code>XMLStreamWriter createXMLStreamWriter(OutputStream in)</code>
<code>XMLStreamWriter createXMLStreamWriter(OutputStream in, String characterEncding)</code>
<code>XMLStreamWriter createXMLStreamWriter(Writer in)</code>
<code>XMLStreamWriter createXMLStreamWriter(Result in) //创建写入给定流、写出器和AXP结果的写出器</code>

javax.xml.stream.XMLStreamWriter 6

<code>void writeStartDocument()</code>
<code>void writeStartDocument(String xmlVersion)</code>
<code>void writeStartDocument(String encoding, String xmlVersion) //在文档的顶部写XML处理指令</code>
<code>void setDefaultNamespace(String namespaceURI)</code>
<code>void setPrefix(String prefix, String namespaceURI)</code>
<code>设置默认的命名空间，或具有前缀的命名空间，作用域仅为当前元素</code>
<code>void writeStartElement(String localName)</code>
<code>void writeStartElement(String namespaceURI, String localName) //写出一个开始签，其中namespaceURI将用相关联的前缀来代替</code>
<code>void writeEndElement() //关闭当前元素</code>
<code>void writeEndDocument() //关闭所有打开的元素</code>
<code>void writeEmptyElement(String localName)</code>
<code>void writeEmptyElement(String namespaceURI, String localName) //写出一个自合的标签</code>
<code>void writeAttribute(String localName, String value)</code>
<code>void writeAttribute(String namespaceURI, String localName, String value) //写一个用于当前元素的属性</code>
<code>void writeCharacters(String text) //写出字符数据</code>
<code>void writeCData(String text) //写出CDATA块</code>
<code>void writeDTD(String dtd) //写出dtd字符串，该字串需要包含一个DOCTYPE声明</code>
<code>void writeComment(String comment) //写出一个注释</code>
<code>void close() //关闭这个写出器</code>

XSL 转换（XSLT）

典型模板：
<pre><code class="highlight-chroma"><span class="highlight
cl"><xsl:template match="/staff/employee">
  <tr><xsl:
pply-templates/></tr>
</xsl:template
gt;
</code></pre>
处理属性值：
<pre><code class="highlight-chroma"><span class="highlight
cl"><xsl:template match="/staff/employee/hiredate">
  <td><xsl:
alue-of select="@year"/>-<xsl:value-of select="@month"/>-<xsl:value-of select=
@day"/></td>
</xsl:template
gt;
</code></pre>
实现 XML 转换

<code>File styleSheet = new File(filename);</code>
<code>StreamSource styleSource = new StreamSource(styleSheet);</code>
<code>Transformer t = TransformerFactory.newInstance().newTransformer(styleSource);
/code>
<code>t.transform(source, result);</code>

Source 接口有 3 个实现类：DOMSource, SAXSource, StreamSource
Result 接口有 3 个实现类：DOMResult，SAXResult，StreamResult
javax.xml.transform.TransformerFactory 1.4

<code>transformer newTransformer(Source stylesheet) //返回一个transformer类的实例
用来从指定的源中读取样式表</code>

javax.xml.transform.stream.StreamSource 1.4

<code>StreamSource(File f)</code>
<code>StreamSource(InputStream in)</code>
<code>StreamSource(Reader in)</code>
<code>StreamSource(String stystemID) //根据一个文件、流、阅读器或系统ID（通常是相
或绝对URL）来构建一个数据流源</code>

javax.xml.transform.sax.SAXSource 1.4

<code>SAXSource(XMLReader reader, InputSource source) //构建一个SAX数据源，以便
给定输入源获取数据，并使用给定的阅读器来解析输入数据</code>

org.xml.sax.XMLReader 1.4

<code>void setContentHandler(ContentHandler handler) //设置在输入被解析时会被告知
析事件的处理器</code>
<code>void parse(InputSource source) //根据给定输入源解析输入数据，并将解析事件发送
内容处理器</code>

javax.xml.transform.dom.DOMResult 1.4

<code>DOMResult(Node n) //根据给定结点构建一个数据源，通常n是一个新文档节点</cod
>

org.xml.sax.helpers.AttributesImpl 1.4

<code>void addAtrribute(String uri, String lname, String qname, String type, String val
e) //将一个属性添加到该属性集合</code>
<code>void clear() //删除属性集合中的所有属性</code>

第三章--网络----------
套接字 Socket
套接字超时设置
java.net.Socket 1.0

<code>Socket(String host, int port) //构建一个套接字，用来连接给定的主机和端口</code

<code>InputStream getInputStream()</code>
<code>OutputStream getOutputStream() //获取可以从套接字中读取数据的流，以及可以
套接字写出数据的流</code>
<code>Socket() //创建一个还未被连接的套接字</code>
<code>void connect(SocketAddress address) //将该套接字连接到给定的地址</code>
<code>void connect(SocketAddress address, int timeoutInMilliSeconds) //将套接字连接
给定的地址，如果在给定的时间内没有响应，则返回</code>
<code>void setSoTimeout(int timeoutInMilliseconds) //设置该套接字上读请求的阻塞时间
如果超出时间，则抛出一个InterruptedIOException异常</code>
<code>boolean isConnected() //如果该套接字已被连接，则返回true</code>
<code>boolean isClosed() //如果该套接字已被关闭，返回true</code>

因特网地址

<code>InetAddress address = InetAddress.getByName("time-A.timefreq.bldrdoc.gov");<
code>
<code>byte[] addressBytes = address.getAddress();</code>

java.net.InetAddress 1.0

<code>static InetAddress getByName(String host)</code>
<code>static InetAddress[] getAllByName(String host) //为给定的主机名，创建一个InetA
dress对象，或者一个包含了该主机名所对应的所有因特网地址的数组</code>
<code>static InetAddress getLocalHost() //为本地主机创建一个InetAddress对象</code>
/li>
<code>byte[] getAddress() //返回一个包含数字型地址的字节数组</code>
<code>String getHostAddress() //返回一个由十进制数组成的字符串，各数字间用圆点符号
开</code>
<code>String getHostName() //返回主机名</code>

实现服务器

<code>ServerSocket s = new ServerSocket(8189);</code>
<code>Socket incoming = s.accept();</code>
<code>InputStream inStream = incoming.getInputStream();</code>
<code>OutputStream outStream = incoming.getOutputStream();</code>
<code>Scanner in = new Scanner(inStream);</code>
<code>PrintWriter out = new PrintWriter(outStream, true /* autoFlush */);</code></l

>
<code>out.println("Hello!Enter BYE to exit. ");</code>
<code>String line = in.nextLine();</code>
<code>out.println("Echo:" + line);</code>
<code>if (line.trim().equals("BYE")) done = true;</code>
<code>incoming.close();</code>

java.net.ServerSocket 1.0

<code>ServerSocket(int port) //创建一个监控端口的服务器套接字</code>
<code>Socket accept() //等待连接，阻塞当前线程直到建立连接为止</code>
<code>void close() //关闭服务器套接字</code>

为多个客户端服务

<code>while (true) {</code>
<code> Socket incoming = s.accept();</code>
<code> Runnable r = new ThreadedEchoHandler(incoming);</code>
<code> Thread t = new Thread(r);</code>
<code> t.start();</code>
<code>}</code>

半关闭
java.net.Socket 1.0

<code>void shutdownOutput() //将输出流设为“流结束” </code>
<code>void shutdownInput() //将输入流设为“流结束” </code>
<code>boolean isOutputShutdown() //如果输出已被关闭返回true</code>
<code>boolean isInputShutdown() //如果输入已被关闭，返回true</code>

可中断套接字
java.nio 包的特性：SocketChannel 类，通道（channel)没有相关联的流，需要调用 Buffer 对来实现 ReadableByteChannel 和 WritableByteChannel 接口声明的 read 和 write 方法

<code>SocketChannel channel = SocketChannel.open(new InetSocketAddress(host, por));</code>
<code>Scanner in = new Scanner(channel);</code>
<code>OutputStream outStream = Channels.newOutputStream(channel);</code>

如果线程发生中断，不会阻塞，而是会抛出异常
java.net.InetSocketAddress 1.4

<code>InetSocketAddress(String hostname, int port) //通过主机和端口参数创建一个地址象，并在创建过程中解析主机名，如果主机名不能被解析，那么该地址对象的unresolved属性被设为true</code>
<code>boolean isUnresolved() //如果不能解析该地址对象，返回true</code>

java.nio.channels.SocketChannel 1.4

<code>static SocketChannel open(SocketAddress address) //打开一个套接字通道，并将连接到远程地址</code>

java.nio.channels.Channels 1.4

<code>static InputStream newInputStream(ReadableByteChannel channel) //创建一个入流，用以从指定的通道读取数据</code>
<code>static OutputStream newOutputStream(WritableByteChannel channel) //创建一输出流，用以向指定的通道写入数据</code>

发送 E-mail

<code>Socket s = new Socket("mail.yourserver.com", 25);</code>
<code>PrintWriter out = new PrintWriter(s.getOutputStream));</code>

信息规范：
<pre><code class="highlight-chroma">HELO sending host
MAIL FROM: <s nder e-mail address>
RCPT TO: <recip ent e-mail address>
DATA
mail message
(any number of li es)
.
QUIT
</code></pre>
SMTP 规范规定，每一行都要以\r 再紧跟一个\n 来结尾
建议 URL 连接
URL 和 URI

<code>URL url = new URL(urlString);</code>
<code>InputStream inStream = url.openStream();</code>
<code>Scanner in = new Scanner(inStream);</code>

URI 句法：
[scheme:]schemeSpecificPart[#fragment]
一个分层 URI 的 schemeSpecificPart 具有以下结构：
[//authority][path][?query]
对基于服务器的 URI，authority 部分采用以下形式：
[user-info@]host[:port]

<code>relative = base.relativize(combined);</code>
<code>combined = base.resolve(relative);</code>

使用 URLConnection 获取信息
1. 调用 URL 类中的 openConnection 方法获得 URLConnection 对象

<code>URLConnection connection = url.openConnection()</code>

2. 使用以下方法来设置任意的请求属性

<code>setDoInput</code>
<code>setDoOutput</code>
<code>setIfModifiedSince</code>
<code>setUseCaches</code>
<code>setAllowUserInteraction</code>

<code>setRequestProperty</code>
<code>setConnectTimeout</code>
<code>setReadTimeout</code>

3. 调用 connect 方法连接远程资源

<code>connection.connect();</code>

4. 建立连接后，可以查询头信息

<code>getContentType</code>
<code>getContentLength</code>
<code>getContentEncoding</code>
<code>getDate</code>
<code>getExpiration</code>
<code>getLastModified</code>

5. 访问资源数据

<code>getInputStream</code>

setRequestProperty 方法，设置“名-值”对

<code>String input = username + ":" + password;</code>
<code>String encoding = base64Encode(input);</code>
<code>connection.setRequestProperty("Authorization", "Basic " + encoding);</code>

java.net.URL 1.0

<code>InputStream openStream() //打开一个用于读取资源数据的输入流</code>
<code>URLConnection openConnection() //返回一个URLConnection对象，该对象负责管与资源之间的连接</code>

java.net.URLConnection 1.0

<code>void setDoInput() //如果doInput为true，那么用户可以接收来自该URLConnection输入</code>
<code>void setDoOutput(boolean doOutput)</code>
<code>boolean getDoOutput(boolean doOutput) //如果doOutput为true，那么用户可以输出发送到该URLConnection</code>
<code>void setIfModifiedSince(long time)</code>
<code>long getIfModifiedSince() //属性ifModifiedSince用于配置URLConnection对象，使只获取那些自从某个给定时间以来被修改过的数据</code>
<code>void setUseCaches(boolean useCaches)</code>
<code>boolean getUseCaches() //如果useCaches为true，那么数据可以从本地缓存中得到URLConnection本身并不维护这个缓存，缓存必须由浏览器之类的外部程序提供</code>
<code>void setAllowUserInteraction(boolean allowUserInteraction)</code>
<code>boolean getAllowUserInteraction() //如果为true，那么可以查询用户的口令</code>
<code>void setConnectTimeout(int timeout)</code>
<code>int getConnectionTimeout() //设置或得到连接超时时限</code>
<code>void setRequestProperty(String key, String value) //设置请求头的一个字段</code>

<code>Map<String，List<String>> getRequestProperties() //返回请求头属性一个映射表，相同的键对应的所有值被放置在同一个映射表中</code>
<code>void connect() //连接远程资源并获取响应头信息</code>
<code>Map<String，List<String>> Map getHeaderFields() //返回响应的一个射表，相同的键对应的所有值被放置在同一个映射表中</code>
<code>String getHeaderFieldKey(int n) //得到响应头第n个字段的键，如果n等于0或大于响头字段的总数返回null值</code>
<code>String getHeaderField(int n) //得到响应头第n个字段的值</code>
<code>int getContentLength() //如果知道内容长度，则返回该长度值，否则返回-1</code>
<code>String getContentType() //获取内容的类型，比如text/plain或image/gif</code></l>
<code>String getContentEncoding() //获取内容的编码，比如gzip</code>
<code>long getDate()</code>
<code>long getExpiration()</code>
<code>long getLastModified() //获取创建日期、过期日以及最后一次被修改的日期</code>
<code>InputStream getInputStream()</code>
<code>OutputStream getOutputStream() //返回从资源读取信息或向资源写入信息的流</code>
<code>Object getContent() //选择适当的内容处理器，以便读取资源数据并将它转换成对象该方法不能用于读取诸如text/plain或image/gif之类的标准内容类型，除非安装了自己的内容处理器</code>

提交表单数据
GET 和 POST 命令，向 Web 服务器发送信息
URL 编码模式编码，参数规则：
保留字符 A-Z、a-z、0-9 以及 . - * _
用 + 字符替换所有的空格
将其他所有字符编码为 UTF-8，并将每个字节都编码为 % 后面紧跟一个两位的十六进制数字

使用 POST 命令的方法：

<code>URL url = new URL("http://host/script");</code>
<code>URLConnection connection = url.openConnection();</code>
<code>connection.setDoOutput(true);</code>
<code>PrintWriter out = new PrintWriter(connection.getOutputStream());</code>
<code>out.print(name1 + "=" + URLEncoder.encode(value1，"UTF-8") + "&");</ode>
<code>out.print(name2 + "=" + URLEncoder.encode(value2，"UTF-8"));</code>
<code>out.close();</code>

java.net.HttpURLConnection 1.0

<code>InputStream getErrorStream() //返回一个流，通过这个流可以读取Web服务器的错信息</code>

java.net.URLEncoder 1.0

<code>static String encode(String s，String encoding) //采用指定的字符编码模式对字符串进行编码，并返回它的URL编码形式</code>

java.net.URLDecoder 1.2

<code>static String decode(String s，String encoding）//采用指定编码模式对已编码字符串进行解码，并返回结果</code>

第三章--数据库编程----------
JDBC 的典型用法
客户端（可视化表示） ---HTTP、RMI---> 中间层（业务逻辑） ---JDBC-数据库协议-> 据库服务器
JDBC URL 一般语法：
jdbc:subprotocol:other stuff
注册驱动器类
Class.forName("org.postgresql.Driver");
或者
java -Djdbc.drivers=org.postgresql.Driver ProgramName
或者
System.setProperty("jdbc.drivers", "org.postgresql.Driver");
连接到数据库

<code>String url = "jdbc:postgresql:COREJAVA";</code>
<code>String username = "dbuser";</code>
<code>String password = "scret";</code>
<code>Connection conn = DriverManager.getConnection(url, username, password);</code>

java.sql.DriverManager 1.1

<code>static Connection getConnection(String url，String user，String password）//建一个到指定数据库的连接，并返回一个Connection对象</code>

执行 SQL 语句
java.sql.Connection 1.1

<code>Statement createStatement() //创建一个Statement对象，用以执行不带参数的SQL询和更新</code>
<code>void close() //用于立即关闭当前的连接以及释放由它所创建的JDBC资源</code>

java.sql.Statement 1.1

<code>ResultSet executeQuery(String sqlQuery）//执行给定字符串中的SQL语句，并返回个用于查看查询结果的ResultSet对象</code>
<code>int executeUpdate(String sqlStatement）//执行字符串中指定的INSERT、UPDATE或ELETE等SQL语句，也可以执行CREATE TABLE等语句，返回受影响的记录总数</code>
<code>boolean execute(String sqlStatement）//执行字符串中指定的SQL语句，可能会产生个结果集和更新数。如果第一个执行结果是结果集，则返回true，反之返回false。调用getResultSet getUpdateCount方法可以得到第一个执行结果</code>
<code>ResultSet getResultSet() //返回前一条查询语句的结果集，如果前一条语句未产生结集则返回null值，对于每一条执行过的语句，该方法只能被调用一次</code>
<code>void close() //关闭Statement对象以及它所对应的结果集</code>
<code>boolean isClosed() //如果语句被关闭返回true</code>

java.sql.ResultSet 1.1

<code>boolean next() //将结果集中的当前行向前移动一行。如果已经到达最后一行的后面，回false，注意初始情况必须调用该方法才能转到第一行</code>
<code>Xxx getXxx(int columnNumber)</code>

<code>Xxx getXxx(String columnName) //用给定的列序号或列标签返回该列的值，并将之换成指定类型</code>
<code>int findColumn(String columnName) //根据给定的列名，返回该列的序号</code>
<code>void close() //立即关闭当前的结果集</code>
<code>boolean isClosed() //如果语句被关闭，则返回true</code>

管理连接、语句和结果集
分析 SQL 异常
每个 SQLException 都有一个由多个 SQLException 对象构成的链，这些对象可以通过 getNextxception 方法获取
java.sql.SQLException 1.1

<code>SQLException getNextException() //返回链接到该SQL异常的下一个SQL异常，或者到达链尾时返回null</code>
<code>Iterator<Throwable> iterator() //获取迭代器，可以迭代链接的SQL异常和它们成因</code>
<code>String getSQLState() //获取“SQL状态”，即标准化的错误代码</code>
<code>int getErrorCode() //获取提供商相关的错误代码</code>

java.sql.Warning 1.1

<code>SQLWarning getNextWarning() //返回链接到该警告的下一个警告，或者在到达链尾返回null</code>

java.sql.Connection 1.1
java.sql.Statement 1.1
java.sql.ResultSet 1.1

<code>QLWarning getWarnings()</code>
<code>SQLWarning getWarnings() //返回未处理警告中的第一个，或者在没有未处理警告时回null</code>

java.sql.DataTruncation 1.1

<code>boolean getParameter() //如果在参数上进行了数据截断，则返回true，如果在列上行了数据截断返回false</code>
<code>int getIndex() //返回被截断的参数和列的索引</code>
<code>int getDataSize() //返回应该被传输的字节数量，或者在该值未知的情况下返回-1</coe>
<code>int getTransferSize() //返回实际被传输的字节数量，或者在该值未知的情况下返回-1<code>

元数据

<code>ResultSet result = stat.getResultSet();</code>
<code>ResultSetMetaData metaData = result.getMetaData();</code>
<code>int columnCount = metaData.getColumnCount();</code>
<code>metaData.getColumnLabel(i);</code>

预备语句
占位符?

<code>PreparedStatement PublisherQueryStat = conn.prepareStatement(publisherQue

y);</code>
<code>publisherQueryStat.setString(1, publisher);</code>
<code>ResultSet rs = publisherQueryStat.executeQuery();</code>

java.sql.Connection 1.1

<code>PreparedStatement prepareStatement(String sql) //返回一个含预编译语句的PreparedStatement对象</code>

java.sql.PreparedStatement 1.1

<code>void setXxx(int n, Xxx x) //设置第n个参数值为x</code>
<code>void clearParameters() //清除预备语句中的所有当前参数</code>
<code>ResultSet executeQuery() //执行预备SQL查询，并返回一个ResultSet对象</code>
<code>int executeUpdate() //执行预备SQL语句INSERT UPDATE或DELETE，返回受影响的录数，如果执行CREATE TABLE,返回0</code>

读写 LOB
二进制大对象称为 BLOB，字符型大对象称为 CLOB
获取一张图像：

PreparedStatement stat = conn.preapareStatement("SELECT Cover FROM BookCovers WERE ISBN=?");

<code>stat.set(1, isbn);</code>

<code>ResultSet result = stat.executeQuery();</code>

<code>if (result.next()) {</code>

<code> Blob coverBlob = result.getBlob(1);</code>

<code> Image coverImage = ImageIO.read(coverBlob.getInputstream());</code>

<code>}</code>

存储一张图像：

<code>Blob coverBlob = connection.createBlob();</code>
<code>int offset = 0;</code>
<code>OutputStream out = coverBlob.setBinaryStream(offset);</code>
<code>ImageIO.write(coverImage, "PNG", out);</code>
<code>PreparedStatement stat = conn.prepareStatement("INSET INTO Cover VALUES (?, ?)");</code>
<code>stat.set(1, isbn);</code>

<code>stat.set(2, coverBlob);</code>
<code>stat.executeUpdate();</code>

java.sql.ResultSet 1.1

<code>Blob getBlob(int columnIndex)</code>
<code>Blob getBlob(String columnLabel)</code>
<code>Clob getClob(int columnIndex)</code>
<code>Clob getClob(String columnLabel) //获取给定列的BLOB或CLOB</code>

java.sql.Blob 1.2

<code>long length() //获取该BLOB的长度</code>
<code>byte[] getBytes(long startPosition, long length) //获取该BLOB中给定范围的数据</ode>
<code>InputStream getBinaryStream()</code>
<code>InputStream getBinaryStream(long startPosition, long length) //返回一个输入流用于读取该BLOB中全部或给定范围的数据</code>
<code>OutputStream setBinaryStream(long startPosition) //返回一个输出流，用于从给定置开始写入该BLOB</code>

java.sql.Clob 1.4

<code>long length() //获取该CLOB中的字符总数</code>
<code>String getSubString(long startPosition, long length) //获取该CLOB中给定范围的符</code>
<code>Reader getCharacterStream()</code>
<code>Reader getCharacterStream(long startPosition, long length) //返回一个读入器（不是流），用于读取CLOB中全部或给定范围的数据</code>
<code>Writer setCharacterStream(long startPosition) //返回一个写出器（而不是流），用从给定位置开始写入该CLOB</code>

java.sql.Connection 1.1

<code>Blob createBlob()</code>
<code>Clob createClob() //创建一个空的BLOB或CLOB</code>

SQL 转义
转义主要用于下列特性：
日期和时间字面常量
调用标量函数
调用存储过程
外连接
在 LIKE 子句中的转义字符
多结果集
遍历 execute 所有结果：

<code>boolean done = false;</code>
<code>boolean isResult = stmt.execute(command);</code>
<code>while (!done) {</code>
<code> if (isResult) {</code>
<code> ResultSet result = stmt.getResultSet();</code>
<code> do something with result</code>
<code> } else {</code>

<code> int updateCount = stmt.getUpdateCount();</code>
<code> if (updateCount >= 0)</code>
<code> do something with updateCount</code>
<code> else</code>
<code> done = true;</code>
<code> }</code>
<code> isResult = stmt.getMoreResults();</code>
<code>}</code>

java.sql.Statement 1.1

<code>boolean getMoreResults() //获取该语句的下一个结果集，如果存在返回true</code

获取自动生成键

<code>stmt.executeUpdate(insertStatement, Statement.RETURN_GENERATED_KEYS);</
ode>
<code>ResultSet rs = stmt.getGeneratedKeys();</code>
<code>if (rs.next()) {</code>
<code> int key = rs.getInt(1);</code>
<code> ...</code>
<code>}</code>

java.sql.Statement 1.1

<code>boolean execute(String statement, int autogenerated)</code>
<code>int executeUpdate(String statement, int autogenerated) //如果autogenerated被
置为Statement.RETURN_GENERATED_KEYS，并且该语句是一条INSERT语句，那么第一列中就是
动生成的键</code>

可滚动和可更新的结果集

<code>Statement stat = conn.createStatement(type, concurrency);</code>
<code>PreparedStatement stat = conn.prepareStatement(command, type, concurrency);
/code>

ResultSet 类的 type 值
| TYPE_FORWARD_ONLY | 结果集不能滚动 |
| TYPE_SCROLL_INSENSITIVE | 结果集可以滚动，但对数据库变化不敏感 |
| TYPE_SCROLL_SENSITIVE | 结果集可以滚动，且对数据库变化敏感 |
ResultSet 类的 Concurrency 值
| CONCUR_READ_ONLY | 结果集不能用于更新数据库 |
| CONCUR_UPDATE | 结果集可以用于更新数据库 |
需要用 getType 和 getConcurrency 方法先检查结果集的功能再使用，否则可能抛出 SQLExcep
ion 异常
在结果集上的滚动

<code>if (rs.previous()) ...</code>

将游标向后或向前移动多行

<code>rs.relative(n);</code>

将游标设置到指定的行号上

<code>rs.absolute(n);</code>
<code>int currentRow = rs.getRow();</code>

获得可更新的结果集

<code>Statement stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);</code>

迭代遍历所有结果并更新相关内容

<code>String query = "SELECT * FROM Books";</code>
<code>ResultSet rs = stat.executeQuery(query);</code>
<code>while (rs.next()) {</code>
<code> if (...) {</code>
<code> double increase = ...</code>
<code> double price = rs.getDouble("Price");</code>
<code> rs.updateDouble("Price", price + increase);</code>
<code> rs.updateRow();</code>
<code> }</code>
<code>}</code>

添加新行

<code>rs.moveToInsertRow();</code>
<code>rs.updateString("Title", title);</code>
<code>rs.updateString("ISBN", isbn);</code>
<code>rs.updateString("Publisher_Id", pubid);</code>
<code>rs.updateDouble("Price", price);</code>
<code>rs.insertRow();</code>
<code>rs.moveToCurrentRow();</code>

删除行

<code>rs.deleteRow();</code>

java.sql.Connection 1.1

<code>Statement createStatement(int type, int concurrency)</code>
<code>PreparedStatement prepareStatement(String command, int type, int concurrenc) //创建一个语句或预备语句，且该语句可以产生指定类型和并发模式的结果集</code>

java.sql.ResultSet 1.1

<code>int getType() //返回结果集的类型</code>
<code>int getConcurrency() //返回结果集的并发设置</code>
<code>boolean previous() //把光标移动到前一行</code>
<code>int getRow() //得到当前行的序号</code>
<code>boolean absolute(int r) //移动光标到第r行</code>
<code>boolean relative(int d) //将光标移动d行，如果d为负数，则光标向后移动，如果光标于某一行上，返回true</code>
<code>boolean first()</code>
<code>boolean last() //移动光标到第一行或最后一行</code>

<code>void beforeFirst()</code>
<code>void afterLast() //移动光标到第一行之前或最后一行之后的位置</code>
<code>boolean isFirst()</code>
<code>boolean isLast() //测试光标是否在第一行或最后一行</code>
<code>boolean isBeforeFirst()</code>
<code>boolean isAfterLast() //测试光标是否在第一行之前或最后一行之后的位置</code></l
>
<code>void moveToInsertRow() //移动光标到插入行</code>
<code>void moveToCurrentRow() //将光标从插入行移回到调用moveToInsertRow之前所
那一行</code>
<code>void insertRow() //将插入行上的内容插入到数据库和结果集中</code>
<code>void deleteRow() //从数据库和结果集中删除当前行</code>
<code>void updateXxx(int column，Xxx data)</code>
<code>void updateXxx(String columnName，Xxx data) //更新结果中当前行上的某个字段
</code>
<code>void updateRow() //将当前行的更新信息发送到数据库</code>
<code>void cancelRowUpdates() //撤销对当前行的更新</code>

java.sql.DatabaseMetaData 1.1

<code>boolean supportsResultSetType(int type) //如果数据库支持给定类型的结果集，则
回true</code>
<code>boolean supportsResultSetConcurrency(int type，int concurrency) //如果数据库
持给定类型和并发模式的结果集，则返回true</code>

行集
CachedRowSet
WebRowSet
FilteredRowSet
JoinRowSet
JdbcRowSet
被缓存的行集
使用一个结果集来填充 CachedRowSet 对象

<code>ResultSet result = ...;</code>
<code>CachedRowSet crs = new com.sun.rowset.CachedRowSetImpl();</code>
<code>crs.populate(result);</code>
<code>conn.close();</code>

或者可以让 CachedRowSet 对象自动创建一个数据库连接并将查询结果填充到行集，最后断开
接

<code>crs.setURL("jdbc:derby://localhost:1527/COREJAVA");</code>
<code>crs.setUsername("dbuser");</code>
<code>crs.setPassword("secret");</code>
<code>crs.setCommand("SELECT * FROM Books WHERE PUBLISHER = ?");</code>
<code>crs.setString(1, publisherName);</code>
<code>crs.execute();</code>

指定每一页尺寸

<code>CachedRowSet crs = ...;</code>
<code>crs.setCommand(command);</code>
<code>crs.setPageSize(20);</code>

<code>...</code>
<code>crs.execute();</code>
<code>crs.nextPage();</code>

将修改写回到数据库中

<code>crs.acceptChanges(conn);</code>
<code>或</code>
<code>crs.acceptChanges(); //这个方法只有设置连接数据库所需信息才有效</code>

javax.sql.RowSet 1.4

<code>String getURL()</code>
<code>void setURL(String url) //获取或设置数据库的URL</code>
<code>String getUsername()</code>
<code>void setUsername(String username) //获取或设置连接数据库所需的用户名</code

<code>String getPassword()</code>
<code>void setPassword(String password) //获取或设置连接数据库所需的密码</code></l
>
<code>String getCommand()</code>
<code>void setCommand(String command) //获取或设置向行集中填充数据时需要执行的
令</code>
<code>void execute() //通过执行使用setCommand方法设置的命令集来填充行集。为了使
动管理器可以获得连接，必须事先设定URL、用户名和密码</code>

javax.sql.rowset.CachedRowSet 5.0

<code>void execute(Connection conn) //通过执行使用setCommand方法设置的命令集来
充行集，该方法使用给定的连接，并负责关闭它</code>
<code>void populate(ResultSet result) //将指定的结果集中的数据填充到被缓存的行集中</c
de>
<code>String getTableName()</code>
<code>void setTableName(String tableName) //获取或设置数据库表名称，填充被缓存的
集时所需的数据来自于该表</code>
<code>int getPageSize()</code>
<code>void setPageSize(int size) //获取和设置页的尺寸</code>
<code>boolean nextPage()</code>
<code>boolean previousPage() //加载下一页或上一页，如果要加载的页存在，返回true</c
de>
<code>void acceptChanges()</code>
<code>void acceptChanges(Connection conn) //重新连接数据库，并写回行集中修改过的
据，如果因为数据库中的数据已经被修改而导致无法写回行集中的数据，该方法可能会抛出SyncProvi
erException异常</code>

元数据
描述数据库或其组成部分的数据称为元数据
获取数据库所有表名（第三列是表名）

<code>DatabaseMetaData meta = conn.getMetaData();</code>
<code>ResultSet mrs = meta.getTables(null, null, null, new String[] {"TABLE"});</code

<code>while (mrs.next())</code>
<code> tableNames.addItem(mrs.getString(3));</code>

通过 ResultSetMetaData 提供结果集的相关信息，如每一列的名称、类型和字段宽度：

<code>ResultSet mrs = stat.executeQuery("SELECT * FROM " + tableName);</code></l
>
<code>ResultSetMetaData meta = mrs.getMetaData();</code>
<code>for (int i = 1; i <= meta.getColumnCount(); i++) {</code>
<code> String columnName = meta.getColumnLabel(i);</code>
<code> int columnWidth = meta.getColumnDisplaySize(i);</code>
<code> ...</code>
<code>}</code>

java.sql.Connection 1.1

<code>DatabaseMetaData getMetaData() //返回一个DatabaseMetaData对象，该对象封
了有关数据库连接的元数据</code>

java.sql.DatabaseMetaData 1.1

<code>ResultSet getTables(String catalog, String schemaPattern, String tableNamePatt
rn, String types[]) //返回某个目录中的所有表的描述，该目录必须符合给定的模式（schema）、
名字模式以及类型标准</code>
<code>int getJDBCMajorVersion()</code>
<code>int getJDBCMinorVersion() //返回建立数据库连接的JDBC驱动程序的主版本号和次
本号</code>
<code>int getMaxConnections() //返回可同时连接到数据库的最大连接数</code>
<code>int getMaxStatements() //返回单个数据库连接允许同时打开的最大语句数</code></
i>

java.sql.ResultSet 1.1

<code>ResultSetMetaData getMetaData() //返回与当前ResultSet对象中的列相关的元数据<
code>

java.sql.ResultSetMetaData 1.1

<code>int getColumnCount() //返回当前ResultSet对象中的列数</code>
<code>int getColumnDisplaySize(int column) //返回给定列序号的列的最大宽度</code></l
>
<code>String getColumnLabel(int column) //返回该列所建议的名称</code>
<code>String getColumnName(int column) //返回指定的列序号所对应的列名</code></li

事务
回滚（rollback）
默认为自动提交模式（autocommit mode）

<code>conn.setAutoCommit(false);</code>
<code>Statement stat = conn.createStatement();</code>
<code>stat.executeUpdate(command1);</code>
<code>stat.executeUpdate(command2);</code>
<code>...</code>
<code>conn.commit();</code>
<code>如果出现错误，请调用：</code>

<code>conn.rollback();</code>

保存点

<code>Statement stat = conn.createStatement(); //start transaction; rollback() goes her
</code>
<code>stat.executeUpdate(command1);</code>
<code>Savepoint svpt = conn.setSavepoint(); //set savepoint; rollback(svpt) goes here</
ode>
<code>stat.executeUpdate(command2);</code>
<code>if (...) conn.rollback(svpt); //undo effect of command2</code>
<code>...</code>
<code>conn.commit();</code>
<code>conn.releaseSavepoint(svpt);</code>

批量更新
使用 DatabaseMetaData 类中的 supportsBatchUpdates 方法可以获知数据库是否支持这种特

可以是 INSERT、UPDATE、DELETE 等操作，也可以是 CREATE TABLE 和 DROP TABLE，但
可以是 SELECT 命令，会抛异常

<code>Statement stat = conn.createStatement();</code>
<code>String command = "CREATE TABLE ...";</code>
<code>stat.addBatch(command);</code>
<code>while (...) {</code>
<code> command = "INSERT INTO ... VALUES (" + ... + ")";</code>
<code> stat.addBatch(command);</code>
<code>}</code>
<code>int[] counts = stat.executeBatch();</code>

java.sql.Connection 1.1

<code>boolean getAutoCommit()</code>
<code>void setAutoCommit(boolean b) //获取该连接中的自动提交模式，或将其设置为b</
ode>
<code>void commit() //提交自上次提交以来所有执行过的语句</code>
<code>void rollback() //撤销自上次提交以来所有执行过的语句所产生的影响</code>
<code>Savepoint setSavepoint()</code>
<code>Savepoint setSavepoint(String name) //设置一个匿名或具名的保存点</code>
<code>void rollback(Savepoint svpt) //回滚到给定保存点</code>
<code>void releaseSavepoint(Savepoint svpt) //释放给定的保存点</code>

java.sql.Savepoint 1.4

<code>int getSavepointId() //获取该匿名保存点的ID号，如果有名字则抛出SQLException异
</code>
<code>String getSavepointName() //获取该保存点的名称，如果为匿名保存点则抛出SQLExc
ption异常</code>

java.sql.Statement 1.1

<code>void addBatch(String command) //添加命令到当前批量命令中</code>
<code>int[] executeBatch() //执行当前批量更新中的所有命令，返回一个记录数的数组</cod
>

java.sql.DatabaseMetaData 1.1

<code>boolean supportsBatchUpdates() //如果驱动程序支持批量更新，返回true</code><
li>

Web 与企业应用中的连接管理
通过目录接口（JNDI）查找

<code>Context jndiContext = new InitialContext();</code>
<code>DataSource source = (DataSource) jndiContext.lookup("java:comp/env/jdbc/core
ava");</code>
<code>Connection conn = source.getConnection();</code>

LDAP 介绍
轻量级目录访问协议（Lightweight Directory Access Protocol, LDAP）
通用的 LDAP 属性
| 属性 ID | 意义 |
| dc | 域构件 |
| cn | 通用名 |
| sn | 姓 |
| dn | 专有名称 |
| o | 组织 |
| ou | 组织单元 |
| uid | 唯一标识符 |
访问 LDAP 目录信息

<code>Hashtable env = new Hashtable();</code>
<code>env.put(Context.SECURITY_PRINCIPAL, username);</code>
<code>env.put(Context.SECURITY_CREDENTIALS, password);</code>
<code>DirContext initial = new InitialDirContext(env);</code>
<code>DirContext context = (DirContext) initial.lookup("ldap://localhost:389");</code>
/li>

javax.naming.directory.InitialDirContext 1.3

<code>InitialDirContext(Hashtable env) //使用给定的环境设置创建一个目录上下文，散列
包含了Context.SECURITY_PRINCIPAL、Context.SECURITY_CREDENTIALS以及其他键的相关信息<
code>

javax.naming.Context 1.3

<code>Object lookup(String name) //使用给定的名称查找对象，返回通常为一棵子树或一
叶对象</code>
<code>Context createSubcontext(String name) //使用给定的名字创建一个子上下文</cod
>
<code>void destroySubcontext(String name) //根据给定的名称删除其对应的子上下文</co
e>
<code>void close() //关闭该上下文</code>

javax.naming.directory.DirContext 1.3

<code>Attributes getAttributes(String name) //根据给定的名称，得到其对应条目的属性</
ode>

<code>void modifyAttributes(String name，int flag，Attributes modes) //根据给定的名，修改其对应条目的属性，flag为以下常量之一：DirContext.ADD_ATTRIBUTE、DirContext.REMOE_ATTRIBUTE或DirContext.REPLACE_ATTRIBUTE</code>

javax.naming.directory.Attributes 1.3

<code>Attribute get(String id) //根据给定的ID，得到其对应的属性</code>
<code>NamingEnumeration extends Attribute> getAll() //返回一个枚举对象，用于迭遍历该属性的所有值</code>
<code>Attribute put(Attribute attr)</code>
<code>Attribute put(String id，Object value) //将一个属性添加到属性集合中</code>

javax.naming.directory.BasicAttributes 1.3

<code>BasicAttributes(String id，Object value) //使用给定的ID和值，构造一个属性集合，集合只包含了单个属性</code>

javax.naming.directory.Attribute 1.3

<code>String getId() //获取该属性的ID</code>
<code>Object get() //如果值已排好序，则获取该属性的第一个值，未排序则返回其中任意一值</code>
<code>NamingEnumeration getAll() //返回一个枚举对象，用于迭代遍历该属性的所有值</code>

javax.naming.NamingEnumeration 1.3

<code>boolean hasMore() //如果该对象还包含其他元素，返回true</code>
<code>T next() //返回下一个元素</code>
