SYSTEM　　java.lang --system

```
/*
System:类中的方法和属性都是静态的。
out:标准输出，默认是控制台。
in：标准输入，默认是键盘。

描述系统一些信息。

获取系统属性信息：Properties getProperties();
*/
import java.util.*;
class SystemDemo
{
	public static void main(String[] args)
	{
		Properties prop = System.getProperties();

		//因为 Properties 是 Hashtable 的子类，也就是 Map 集合的一个子类对象。
		//那么可以通过 map 的方法取出该集合中的元素。
		//该集合中存储都是字符串。没有泛型定义。

		//如何在系统中自定义一些特有信息呢？
		System.setProperty("mykey","myvalue");

		//获取指定属性信息。
		String value = System.getProperty("os.name");
		System.out.println("value="+value);
		//可不可以在 jvm 启动时，动态加载一些属性信息呢？
		String v = System.getProperty("haha");

		System.out.println("v="+v);
		/*
		//获取所有属性信息。
		for(Object obj : prop.keySet())
		{
			String value = (String)prop.get(obj);

			System.out.println(obj+"::"+value);
		}
		*/
	}
}
```

Runtime
单例设计模式

```
/*
Runtime 对象
该类并没有提供构造函数。
说明不可以 new 对象。那么会直接想到该类中的方法都是静态的。
发现该类中还有非静态方法。
说明该类肯定会提供了方法获取本类对象。而且该方法是静态的，并返回值类型是本类类型。

由这个特点可以看出该类使用了单例设计模式完成。

该方式是 static Runtime getRuntime();
*/

class    RuntimeDemo
{
    public static void main(String[] args) throws Exception
    {
        Runtime r = Runtime.getRuntime();
        Process p = r.exec("notepad.exe    SystemDemo.java");

        //Thread.sleep(4000);
        //p.destroy();

    }
}
```

```
import java.util.*;
import java.text.*;
class    CalendarDemo
{
      public static void main(String[] args)
      {
            Calendar c = Calendar.getInstance();
            String[] mons = {"一月","二月","三月","四月"
                              ,"五月","六月","七月","八月"
                              ,"九月","十月","十一月","十二月"};
            String[] weeks = {
                                    "","星期日","星期一","星期二","星期三","星期四","星期五","星
期六",
                                          };
            int index = c.get(Calendar.MONTH);
            int index1 = c.get(Calendar.DAY_OF_WEEK);
            sop(c.get(Calendar.YEAR)+"年");
            //sop((c.get(Calendar.MONTH)+1)+"月");
            sop(mons[index]);
            sop(c.get(Calendar.DAY_OF_MONTH)+"日");
            //sop("星期"+c.get(Calendar.DAY_OF_WEEK));
            sop(weeks[index1]);
      }

      public static void sop(Object obj)
      {
            System.out.println(obj);
      }
}
```

Random

1. Lang math.random
2. Util random

IO 流用来处理设备之间的数据传输
Java 对数据的操作是通过流的方式
Java 用于操作流的对象都在 IO 包中
流按操作数据分为两种：字节流与字符流。
流按流向分为：输入流，输出流。

GBK
UTF-8
字符流中融合了编码表

字节流的抽象基类：
InputStream ， OutputStream。
字符流的抽象基类：
Reader ， Writer。
注：由这四个类派生出来的子类名称都是以其父类名作为子类名的后缀。
如：InputStream 的子类 FileInputStream。
如：Reader 的子类 FileReader。

先学习一下字符流的特点。

既然 IO 流是用于操作数据的，
那么数据的最常见体现形式是：文件。

那么先以操作文件为主来演示。

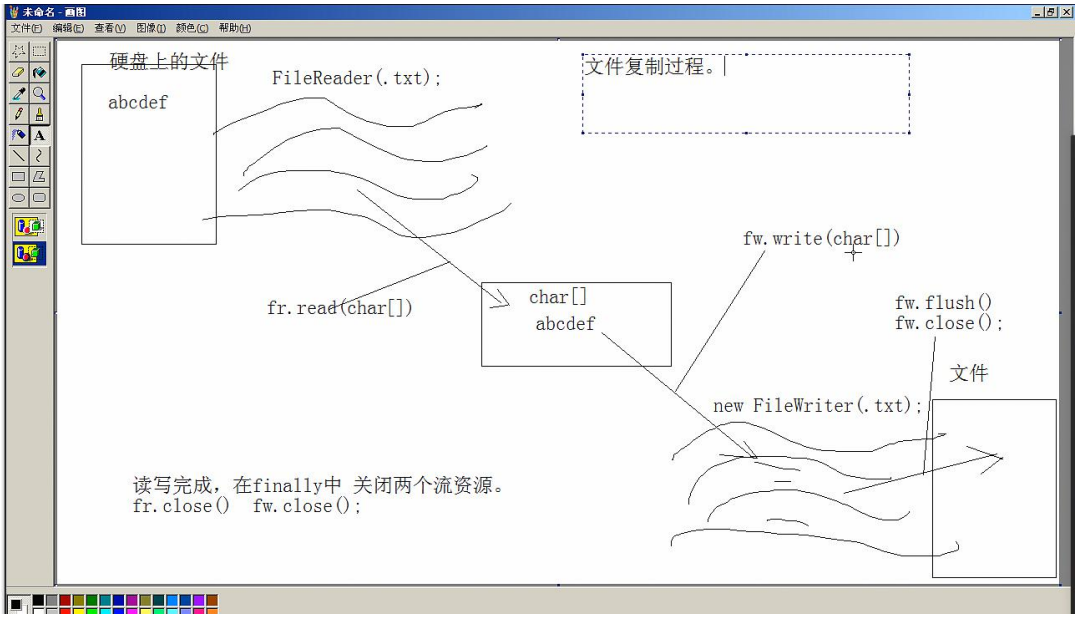
需求:在硬盘上，创建一个文件并写入一些文字数据。

找到一个专门用于操作文件的 Writer 子类对象。FileWriter。 后缀名是父类名。 前缀名是该流对象的功能。

```
*/
import java.io.*;
class  FileWriterDemo
{
    public static void main(String[] args) throws IOException
    {
        //创建一个 FileWriter 对象。该对象一被初始化就必须要明确被操作的文件。
        //而且该文件会被创建到指定目录下。如果该目录下已有同名文件，将被覆盖。
        //其实该步就是在明确数据要存放的目的地。
        FileWriter fw = new FileWriter("demo.txt");

        //调用 write 方法，将字符串写入到流中。
        fw.write("abcde");
```

```
        //刷新流对象中的缓冲中的数据。
        //将数据刷到目的地中。
        //fw.flush();

        //关闭流资源，但是关闭之前会刷新一次内部的缓冲中的数据。
        //将数据刷到目的地中。
        //和 flush 区别：flush 刷新后，流可以继续使用，close 刷新后，会将流关闭。
        fw.close();
    }
}

import java.io.*;class FileWriterDemo3
{
        public static void main (String[] args)
        {
                FileWriter fw = null;
                try
                {
                        fw = new FileWriter ("demo.txt", true);
                        fw.write ("haha");
                }
                catch (IOException e)
                {
                        System.out.println (e.toString());
                }
                finally
                {
                        try{
                        if (fw!=null)
                                fw.close ();
                        }
                        catch (IOException e)
                        {
                                System.out.println (e.toString());
                        }

                }
        }
}


import java.io.*;
class FileReaderDemo2
```

```
{
        public static void main (String[] args) throws IOException
        {
                //FileReader fr =  new FileReader ("demo.txt");                char []
                int num = 0;
                while ((num=fr.read(buf))!=-1)
                {
                        System.out.println(new String (buf, 0, num));
                }
        }
}
```



字符流的缓冲区

缓冲区的出现提高了对数据的读写效率。

对应类

BufferedWriter

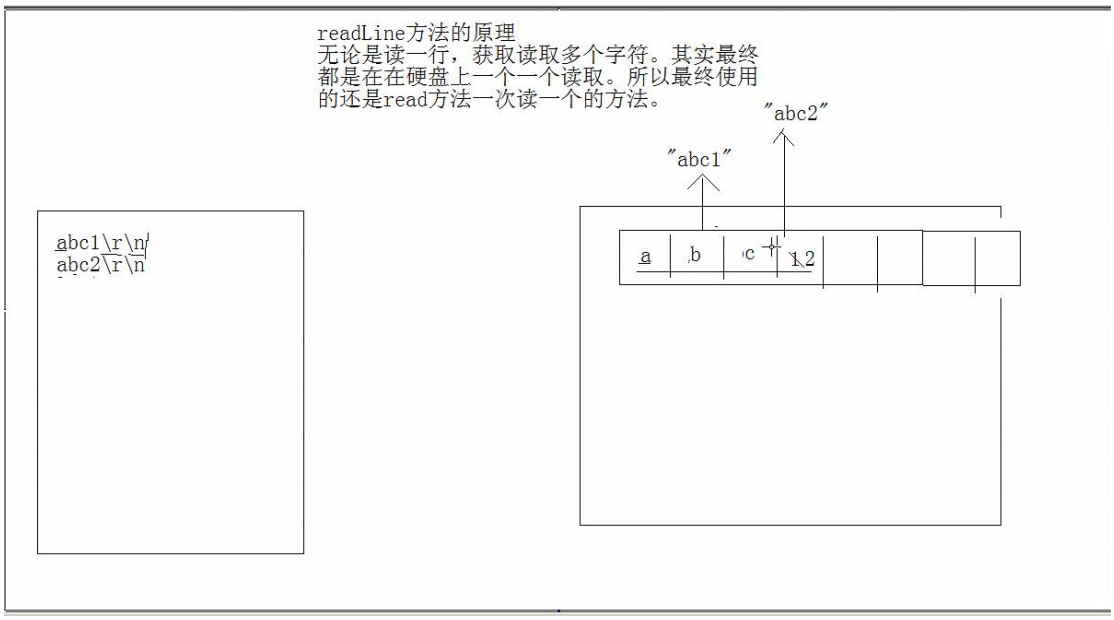
BufferedReader

缓冲区要结合流才可以使用。

在流的基础上对流的功能进行了增强。

```
/*
缓冲区的出现是为了提高流的操作效率而出现的。
所以在创建缓冲区之前，必须要先有流对象。
```

该缓冲区中提供了一个跨平台的换行符。
newLine();
*/



/*
装饰设计模式：
当想要对已有的对象进行功能增强时，
可以定义类，将已有对象传入，基于已有的功能，并提供加强功能。
那么自定义的该类称为装饰类。

装饰类通常会通过构造方法接收被装饰的对象。
并基于被装饰的对象的功能，提供更强的功能。
*/
装饰和继承
继承体系

```
MyReader//专门用于读取数据的类。
    |--MyTextReader
        |--MyBufferTextReader
    |--MyMediaReader
        |--MyBufferMediaReader
    |--MyDataReader
        |--MyBufferDataReader
```

装饰

```
13
14 class MyBufferReader
15 {
16     MyBufferReader(MyTextReader text)
17     {}
18     MyBufferReader(MyMediaReader media)
19     {}
20 }
21 上面这个类扩展性很差。
22 找到其参数的共同类型。通过多态的形式。可以提高扩展性。
23
24 class MyBufferReader
25 {
26     MyBufferReader(MyReader r)
27     {}
28
29 }
```

现有体系

```
29
30
31 MyReader//专门用于读取数据的类。
32     |--MyTextReader
33     |--MyMediaReader
34     |--MyDataReader
35     |--MyBufferReader
36
37
```

装饰模式比继承要灵活。避免了继承体系臃肿。
而且降低了类于类之间的关系。

装饰类因为增强已有对象，具备的功能和已有的是相同的，只不过提供了更强功能。
所以装饰类和被装饰类通常是都属于一个体系中的。

Mybufferedreader 继承装饰 reader
MyLineNumReader 继承装饰 myBufferedReader

/*
字符流：
FileReader
FileWriter。

BufferedReader
BufferedWriter

字节流：
FileInputStream
FileOutputStream

BufferedInputStream
BufferedOutputStream

TranStream
通过刚才的键盘录入一行数据并打印其大写，发现其实就是读一行数据的原理。
也就是 readLine 方法。

能不能直接使用 readLine 方法来完成键盘录入的一行数据的读取呢？

readLine 方法是字符流 BufferedReader 类中的方法。

而键盘录入的 read 方法是字节流 InputStream 的方法。

那么能不能将字节流转成字符流在使用字符流缓冲去的 readLine 方法呢？

*/

流操作的基本规律：
最痛苦的就是流对象有很多，不知道该用哪一个。

通过三个明确来完成。

1，明确源和目的。
源：输入流。InputStream　Reader
目的：输出流。OutputStream　Writer。
2，操作的数据是否是纯文本。
是：字符流。
不是：字节流。

3，当体系明确后，在明确要使用哪个具体的对象。

通过设备来进行区分：
源设备：内存，硬盘。键盘
目的设备：内存，硬盘，控制台。

2，需求：将键盘录入的数据保存到一个文件中。
这个需求中有源和目的都存在。
那么分别分析
源：InputStream Reader
是不是纯文本？是！Reader

设备：键盘。对应的对象是 System.in.
不是选择 Reader 吗？System.in 对应的不是字节流吗？
为了操作键盘的文本数据方便。转成字符流按照字符串操作是最方便的。
所以既然明确了 Reader，那么就将 System.in 转换成 Reader。
用了 Reader 体系中转换流,InputStreamReader

```
InputStreamReader isr = new InputStreamReader(System.in);
```

需要提高效率吗？需要！BufferedReader

```
BufferedReader bufr = new BufferedReader(isr);
```

目的：OutputStream or Writer
是否是存文本？是！Writer。
设备：硬盘。一个文件。使用 FileWriter。

```
FileWriter fw = new FileWriter("c.txt");
```

需要提高效率吗？需要。

```
BufferedWriter bufw = new BufferedWriter(fw);
```

****
扩展一下，想要把录入的数据按照指定的编码表（utf-8），将数据存到文件中。

目的：OutputStream Writer
是否是存文本？是！Writer。
设备：硬盘。一个文件。使用 FileWriter。
但是 FileWriter 是使用的默认编码表。GBK.

但是存储时，需要加入指定编码表 utf-8。而指定的编码表只有转换流可以指定。
所以要使用的对象是 OutputStreamWriter。
而该转换流对象要接收一个字节输出流。而且还可以操作的文件的字节输出流。
FileOutputStream

```
OutputStreamWriter osw =
        new OutputStreamWriter(new FileOutputStream("d.txt"),"UTF-8");
```

需要高效吗？需要。

BufferedWriter bufw = new BufferedWriter(osw);

所以，记住。转换流什么使用。字符和字节之间的桥梁，通常，涉及到字符编码转换时，

需要用到转换流。

练习：将一个文本数据打印在控制台上。要按照以上格式自己完成三个明确。

流操作的是数据

File 类
用来将文件或者文件夹封装成对象
方便对文件与文件夹的属性信息进行操作。
File 对象可以作为参数传递给流的构造函数。
了解 File 类中的常用方法。

File 类常见方法：
1，创建。
boolean createNewFile():在指定位置创建文件，如果该文件已经存在，则不创建，返回 false。
和输出流不一样，输出流对象一建立创建文件。而且文件已经存在，会覆盖。

boolean mkdir():创建文件夹。
boolean mkdirs():创建多级文件夹。
2，删除。
boolean delete()：删除失败返回 false。如果文件正在被使用，则删除不了返回 falsel。
void deleteOnExit();在程序退出时删除指定文件。

3，判断。
boolean exists() :文件是否存在.
isFile():
isDirectory();
isHidden();
isAbsolute();

4，获取信息。
getName():
getPath():
getParent():

getAbsolutePath()
long lastModified()
long length()　联想到字节流的 available

ListRoot
List

```
public static void method_3 ()
        {
                File dir = new File ("D://javafile5");

                String [] files = dir . list(new FilenameFilter ()
                {
                        public boolean accept(File dir,String name)
                        {
                                return name.endsWith (".java");
                        }
                });

            for (String file: files)
                {
                        sop (file);
                }
        }
```

递归 stack

```
/*
列出指定目录下文件或者文件夹，包含子目录中的内容。
也就是列出指定目录下所有内容。

因为目录中还有目录，只要使用同一个列出目录功能的函数完成即可。
在列出过程中出现的还是目录的话，还可以再次调用本功能。
也就是函数自身调用自身。
这种表现形式，或者编程手法，称为递归。

递归要注意：
1，限定条件。
2，要注意递归的次数。尽量避免内存溢出。

*/
```

/*
Properties 是 hashtable 的子类。
也就是说它具备 map 集合的特点。而且它里面存储的键值对都是字符串。

是集合中和 IO 技术相结合的集合容器。

该对象的特点：可以用于键值对形式的配置文件。

那么在加载数据时，需要数据有固定格式：键=值。

练习：限制程序运行次数。当运行次数到达 5 次时，给出，请您注册的提示。并不再让该程序执行。

*/

```
public static void loadDemo()throws IOException
	{
		Properties prop = new Properties();
		FileInputStream fis = new FileInputStream("info.txt");

		//将流中的数据加载进集合。
		prop.load(fis);

		prop.setProperty("wangwu","39");

		FileOutputStream fos = new FileOutputStream("info.txt");
		prop.store(fos,"haha");

	//	System.out.println(prop);
		prop.list(System.out);

		fos.close();
		fis.close();
	}
```

字符打印流（常用）：

PrintWriter

构造函数可以接收的参数类型：

1，file 对象。File

2，字符串路径。String

3，字节输出流。OutputStream

4，字符输出流，Writer。

```
import java.io.*;
import java.util.*;
class SequenceDemo
{
	public static void main(String[] args) throws IOException
	{

		Vector<FileInputStream> v = new Vector<FileInputStream>();

		v.add(new FileInputStream("c:\\1.txt"));
		v.add(new FileInputStream("c:\\2.txt"));
		v.add(new FileInputStream("c:\\3.txt"));

		Enumeration<FileInputStream> en = v.elements();
		SequenceInputStream sis = new SequenceInputStream(en);

		FileOutputStream fos = new FileOutputStream("c:\\4.txt");

		byte[] buf = new byte[1024];
		int len =0;
		while((len=sis.read(buf))!=-1)
		{
			fos.write(buf,0,len);
		}

		fos.close();
		sis.close();

	}
}
```

```
import java.io.*;
import java.util.*;

class SplitFile
{
	public static void main(String[] args) throws IOException
	{
		//splitFile();
		merge();
	}

	public static void merge()throws IOException
	{
		ArrayList<FileInputStream> al = new ArrayList<FileInputStream>();
		for(int x=1; x<=3; x++)
		{
			al.add(new FileInputStream("c:\\splitfiles\\"+x+".part"));
		}
		final Iterator<FileInputStream> it = al.iterator();
		Enumeration<FileInputStream> en = new Enumeration<FileInputStream>()
		{
			public boolean hasMoreElements()
			{
				return it.hasNext();
			}
			public FileInputStream nextElement()
			{
				return it.next();
			}
		};
		SequenceInputStream sis = new SequenceInputStream(en);
		FileOutputStream fos = new FileOutputStream("c:\\splitfiles\\0.bmp");
		byte[] buf = new byte[1024];
		int len = 0;
		while((len=sis.read(buf))!=-1)
		{
			fos.write(buf,0,len);
		}

		fos.close();
		sis.close();
	}
	public static void splitFile()throws IOException
	{
```

```
            FileInputStream fis =    new FileInputStream("c:\\1.bmp");
            FileOutputStream fos = null;
            byte[] buf = new byte[1024*1024];
            int len = 0;
            int count = 1;
            while((len=fis.read(buf))!=-1)
            {
                  fos = new FileOutputStream("c:\\splitfiles\\"+(count++)+".part");
                  fos.write(buf,0,len);
                  fos.close();
            }
            fis.close();

      }
}
```

IO 包中的其他类

打印流

PrintWriter 与 PrintStream

可以直接操作输入流和文件。

序列流

SequenceInputStream

对多个流进行合并。

操作对象

ObjectInputStream 与 ObjectOutputStream

被操作的对象需要实现 Serializable （标记接口);

练习：文件分割程序。

Serializable UID

编译器可以根据这个来判断这个对象产生之后他的类是否修改了。

可以将堆内存内容进行序列化，不能将方法区，静态的内容序列化。

```
import java.io.*;
class Person implements Serializable
{
	public static final long serialVersionUID = 42L;
	private String name;
	transient int age;
	static String country = "cn";
	Person(String name,int age,String country)
	{
		this.name = name;
		this.age = age;
		this.country = country;
	}
	public String toString()
	{
		return name+":"+age+":"+country;
	}
}
```

```
/*
RandomAccessFile
该类不是算是 IO 体系中子类。
而是直接继承自 Object。

但是它是 IO 包中成员。因为它具备读和写功能。
内部封装了一个数组，而且通过指针对数组的元素进行操作。
可以通过 getFilePointer 获取指针位置，
同时可以通过 seek 改变指针的位置。

其实完成读写的原理就是内部封装了字节输入流和输出流。

通过构造函数可以看出，该类只能操作文件。
而且操作文件还有模式：只读 r，，读写 rw 等。

如果模式为只读 r。不会创建文件。会去读取一个已存在文件，如果该文件不存在，则会出现异常。
如果模式 rw。操作的文件不存在，会自动创建。如果存则不会覆盖。
*/
```

```
public static void writeFile_2()throws IOException
	{
		RandomAccessFile raf = new RandomAccessFile("ran.txt","rw");
		raf.seek(8*0);
		raf.write("周期".getBytes());
		raf.writeInt(103);

		raf.close();
	}




/*
DataInputStream 与 DataOutputStream

可以用于操作基本数据类型的数据的流对象。
*/

DataOutputStream dos = new DataOutputStream(new FileOutputStream("utfdate.txt"));

dos.writeUTF("你好");


转换流
//		OutputStreamWriter osw =
			new OutputStreamWriter(new FileOutputStream("gbk.txt"),"gbk");
//


//		OutputStreamWriter osw =
				new OutputStreamWriter(new FileOutputStream("gbk.txt"),"UTF”);
//
```

/*
用于操作字节数组的流对象。

ByteArrayInputStream ：在构造的时候，需要接收数据源，。而且数据源是一个字节数组。

ByteArrayOutputStream： 在构造的时候，不用定义数据目的，因为该对象中已经内部封装了可变长度的字节数组。
这就是数据目的地。

因为这两个流对象都操作的数组，并没有使用系统资源。
所以，不用进行 close 关闭。

在流操作规律讲解时
源设备，
    键盘 System.in，硬盘 FileStream，内存 ArrayStream。
目的设备：
    控制台 System.out，硬盘 FileStream，内存 ArrayStream。

用流的读写思想来操作数据。
*/

字符编码
字符流的出现为了方便操作字符。
更重要是的加入了编码转换。
通过子类转换流来完成。
InputStreamReader
OutputStreamWriter
在两个对象进行构造的时候可以加入字符集

/*
有五个学生，每个学生有 3 门课的成绩，
从键盘输入以上数据（包括姓名，三门课成绩），
输入的格式：如：zhagnsan，30，40，60 计算出总成绩，
并把学生的信息和计算出的总分数高低顺序存放在磁盘文件"stud.txt"中。

1，描述学生对象。
2，定义一个可操作学生对象的工具类。

思想：
1，通过获取键盘录入一行数据，并将该行中的信息取出封装成学生对象。
2，因为学生有很多，那么就需要存储，使用到集合。因为要对学生的总分排序。
　　所以可以使用 TreeSet。
3，将集合的信息写入到一个文件中。

*/