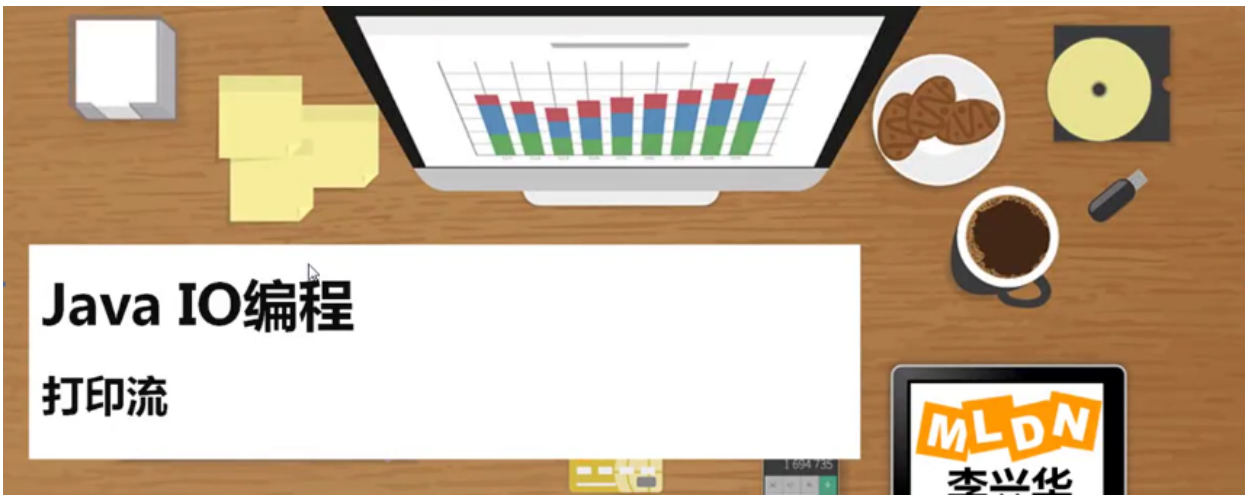
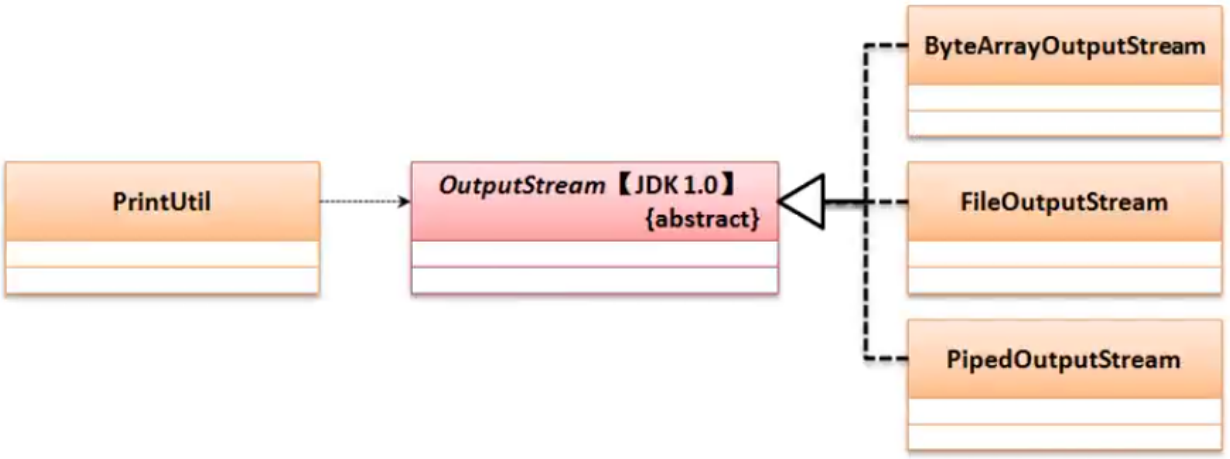


博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

如果现在要想通过程序实现内容的输出，核心的本质一定要依靠OutputStream类完成，但是OutputStream类有一个最大的缺点，这个类中的数据输出操作功能有限：public void write(byte[] b) throws IOException，所有的数据一定要转为字节数组后才可以输出，于是假设说现在项目里面可能输出的是long、double、Date，在这样的情况下就必须将这些数据变为字节的形式来处理，这样的处理一定是非常麻烦的，所以在开发之中最初的时候为了解决此类的重复操作，往往会由开发者自行定义一些功能类以简化输出过程。



范例：打印流设计思想

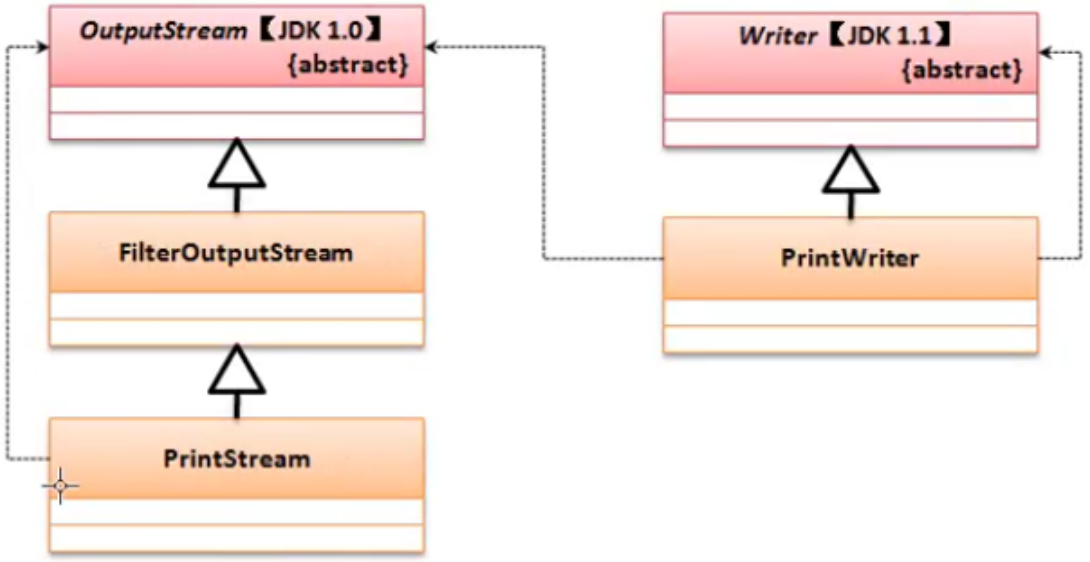
```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "mldn.txt"); // 定义操作文件
        PrintUtil pu = new PrintUtil(new FileOutputStream(file)) ;
        pu.println("姓名：小强子");
        pu.print("年龄：");
```

```
            pu.println(78);
            pu.close();
        }
}
class PrintUtil implements AutoCloseable {    // 实现一些常用数据的输出
        private OutputStream output ; // 不管你现在如何进行输出操作，核心使用的就是
OutputStream
        public PrintUtil(OutputStream output) {// 由外部来决定输出的位置
            this.output = output ;
        }
        @Override
        public void close() throws Exception {
            this.output.close();
        }
        public void println(long num) {
            this.println(String.valueOf(num));
        }
        public void print(long num) {
            this.print(String.valueOf(num));
        }
        public void print(String str) {  // 输出字符串
            try {
                this.output.write(str.getBytes());      // 输出
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public void println(String str) {
            this.print(str + "\r\n");
        }
}
```

在整个的操作过程之中打印流的设计思想的本质在于：提供已有类的功能，例如：OutputStream是唯一可以实现输出的操作标准类，所以应该以其为核心根本，但是这个类输出的操作功能有限，所以不方便进行输出各个数据类型，那么就为它做出了一层包装，所以此时采用的设计思想就是“装饰设计模式”。

但是既然所有的开发者都已经发现了原始中的OutputStream功能的不足，设计者也一定可以发现，所以为了解决输出问题，在java.io包里面提供有打印流：PrintStream、PrintWriter；

打印流



PrintStream	PrintWriter
public class PrintStream extends FilterOutputStream implements Appendable, Closeable	public class PrintWriter extends Writer
public FilterOutputStream(OutputStream out)	public PrintWriter(OutputStream out) public PrintWriter(Writer out)

下面使用PrintWriter来实现数据的输出操作。

范例：数据输出

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		File file = new File("d:" + File.separator + "mldn.txt"); // 定义操作文件
		PrintWriter pu = new PrintWriter(new FileOutputStream(file)) ;
		pu.println("姓名：小强子");
		pu.print("年龄：");
		pu.println(78);
		pu.close();
	}
}
```

从JDK1.5开始PrintWriter类里面追加有格式化输出的操作支持：public PrintWriter printf(String format, Object... args)；

范例：格式化输出

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
```

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		File file = new File("d:" + File.separator + "mldn.txt"); // 定义操作文件
		PrintWriter pu = new PrintWriter(new FileOutputStream(file)) ;
		String name = "小强子子" ;
		int age = 78 ;
		double salary = 72823.6323113 ;
		pu.printf("姓名：%s、年龄：%d、收入：%9.2f", name,age,salary) ;
		pu.close();
	}
}
```

比起直接使用OutputStream类，那么使用PrintWriter、PrintStream类的处理操作会更加的简单。以后只要是程序进行内容输出的时候全部使用打印流。

2、具体内容

System类是一个系统类，而且是一个从头到尾一直都在使用的系统类，而在这个系统类之中实际上提供有三个常量：

·标准输出（显示器）：public static final PrintStream out;

·错误输出：public static final PrintStream err;

·标准输入（键盘）：public static final InputStream in;

范例：观察输出

```
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		try {
			Integer.parseInt("a") ;
		} catch (Exception e) {
			System.out.println(e);
			System.err.println(e);
		}
	}
}
```

```
java.lang.NumberFormatException: For input string: "a"
```

```
java.lang.NumberFormatException: For input string: "a"
```

System.out和System.err都是同一种类型的，如果现在使用的是Eclipse则在使用System.err输出的时候会使用红色字体，而System.out会使用黑色字体。

最早设置连个输出的操作是有目的的：System.out是输出那些希望用户可以看见的信息，System.err是输出那些不希望用户看见的信息。

·修改out的输出位置：public static void setOut(PrintStream out)；

·修改err的输出位置：public static void setErr(PrintStream err)；

范例：修改System.err位置

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		System.setErr(new PrintStream(new FileOutputStream(new File("d:" +
File.separator + "mldn-err.txt"))));
		try {
			Integer.parseInt("a");
		} catch (Exception e) {
			System.out.println(e);
			System.err.println(e);		// 输出到文件里了
		}
	}
}
```

在System类里面还提供有一个in的常量，而这个常量对应的是标准输入设备键盘的输入处理，可以实现键盘数据输入。

范例：实现键盘输入

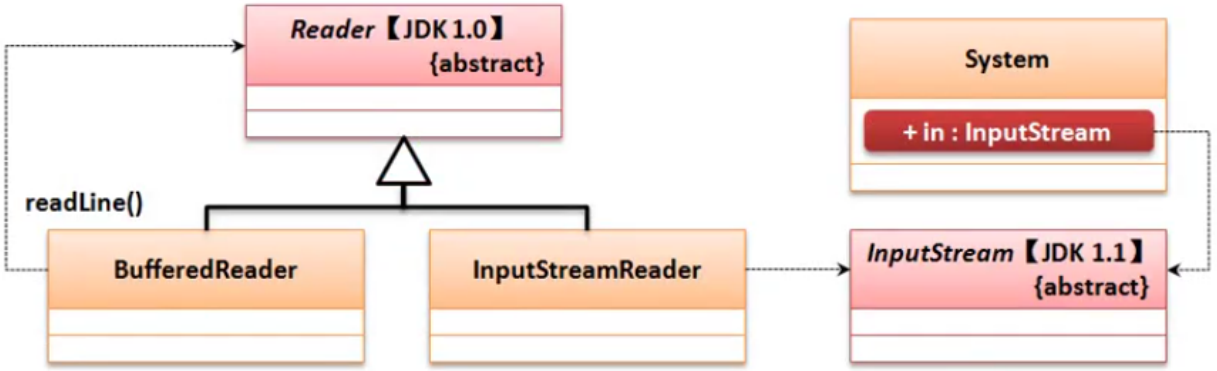
```
import java.io.InputStream;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		InputStream input = System.in ; // 此时的输入流为键盘输入
		System.out.print("请输入信息：");
		byte [] data = new byte [1024] ;
		int len = input.read(data) ;
		System.out.println("输入内容为：" + new String(data, 0, len));
	}
}
```

但是这样的键盘输入处理本身是有缺陷的：如果你先现在的长度不足，那么只能够接收部分数据，所以这个输入就有可能需要进行重复的输入流数据接收，而且在接收的时候还有可能会牵扯到输入中文的情况，如果对于中文的处理不当，则也有可能造成乱码问题。

2、具体内容

BuffereReader类提供的是一个缓冲字符输入流的概念，也就是说利用BufferedReader类可以很好的解决输入流数据的读取问题，这个类是在最初的时候提供的最完善的数据输入的处理（JDK1.5之前，JDK1.5之后出了一个功能更强大的类代替此类），之所以使用这个类来处理，是因为这个类中提供有一个重要的方法：

·**读取一行数据**：public String readLine() throws IOException；



将利用这个类实现键盘输入数据的标准化定义。

范例：实现键盘数据输入

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
		System.out.print("请输入信息：");
		String msg = input.readLine() ; // 接收输入信息
		System.out.println("输入内容为：" + msg);
	}
}
```

在以后实际的开发过程之中经常会遇见输入数据的情况，而所有输入数据的类型都是通过String描述的，那么这样就方便了接受者进行各种处理。

范例：接收整型输入并且验证

```
import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
		System.out.print("请输入您的年龄：");
		String msg = input.readLine() ; // 接收输入信息
		if (msg.matches("\\d{1,3}")) {  // 是否由数字所组成
			int age = Integer.parseInt(msg) ;
			System.out.println("年龄为：" + age);
		} else {
			System.out.println("请确保您看懂我的提示，不要随意输入，伤不起！");
		}
	}
}
```

对于现代的Java开发由键盘输入数据的情况并不多了，但是做为一些基础的逻辑训练还是可以使用键盘输入数据的，而键盘输入数据的标准做法（JDK1.5）就是上面的实现操作。实际开发中所有输入的数据全部都是字符串，这样可以方便用户验证与进行字符串的复杂处理。

2、具体内容

java.util.Scanner是从JDK1.5之后追加的一个程序类，其主要的目的是为了解决输入流的访问问题的，可以理解为BufferedReader的替代功能类，在Scanner类里面有如下几种操作方法：

·构造方法：public Scanner(InputStream source)；

·判断是否有数据：public boolean hasNext()；

·取出数据：public String next()；

·设置分隔符：public Scanner useDelimiter(String pattern)；

范例：使用Scanner实现键盘数据输入

```
import java.util.Scanner;
public class JavaAPIDemo {
```

```
    public static void main(String[] args) throws Exception {
        Scanner scan = new Scanner(System.in) ;
        System.out.print("请输入年龄：");
        if (scan.hasNextInt()) {    // 判断是否有整数输入
            int age = scan.nextInt() ; // 直接获取数字
            System.out.println("您的年龄：" + age);
        } else {
            System.out.println("咋看不懂人话呢？输入的是年龄。");
        }
        scan.close();
    }
}
```

此时可以明显的感受到Scanner的处理会更加的简单。

范例：输入一个字符串

```
import java.util.Scanner;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Scanner scan = new Scanner(System.in) ;
        System.out.print("请输入信息：");
        if (scan.hasNext()) {
            String msg = scan.next() ;
            System.out.println("输入信息为：" + msg);
        }
        scan.close();
    }
}
```

使用Scanner输入数据还有一个最大的特点是可以直接利用正则进行验证判断。

范例：输入一个人的生日(yyyy-MM-dd)

```
import java.text.SimpleDateFormat;
import java.util.Scanner;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Scanner scan = new Scanner(System.in);
        System.out.print("请输入您的生日：");
        if (scan.hasNext("\\d{4}-\\d{2}-\\d{2}")) {
            String str = scan.next("\\d{4}-\\d{2}-\\d{2}");
            System.out.println("输入信息为：" + new SimpleDateFormat("yyyy-MM-
dd").parse(str));
        }
        scan.close();
    }
}
```

现在可以发现Scanner的整体设计要好于BufferedReader，而且要比直接使用InputStream类读取要方便。例如：现在如果要读取一个文本文件中所有内容信息，如果采用的是InputStream类，那么就必须依靠内存输出流进行临时数据的保存，随后还需要判断读取的内容是否是换行。

范例：使用Scanner读取

```
//mldn-intfo.txt 若不存在，可以自己指向任一文件
import java.io.File;
import java.util.Scanner;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Scanner scan = new Scanner(new File("D:" + File.separator + "mldn-info.txt"));
		scan.useDelimiter("\n");// 设置读取分隔符
		while (scan.hasNext()) {
			System.out.println(scan.next());
		}
		scan.close();
	}
}
```

在以后的开发过程之中，如果程序需要输出数据一定使用打印流，输入数据使用Scanner（BufferedReader）。