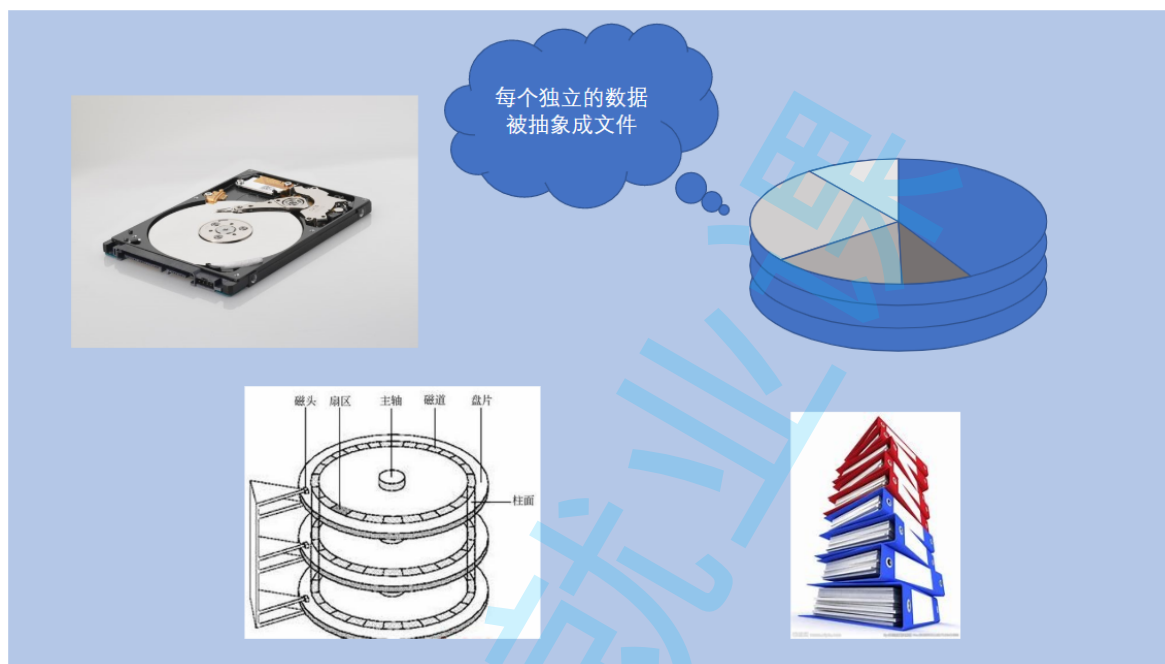


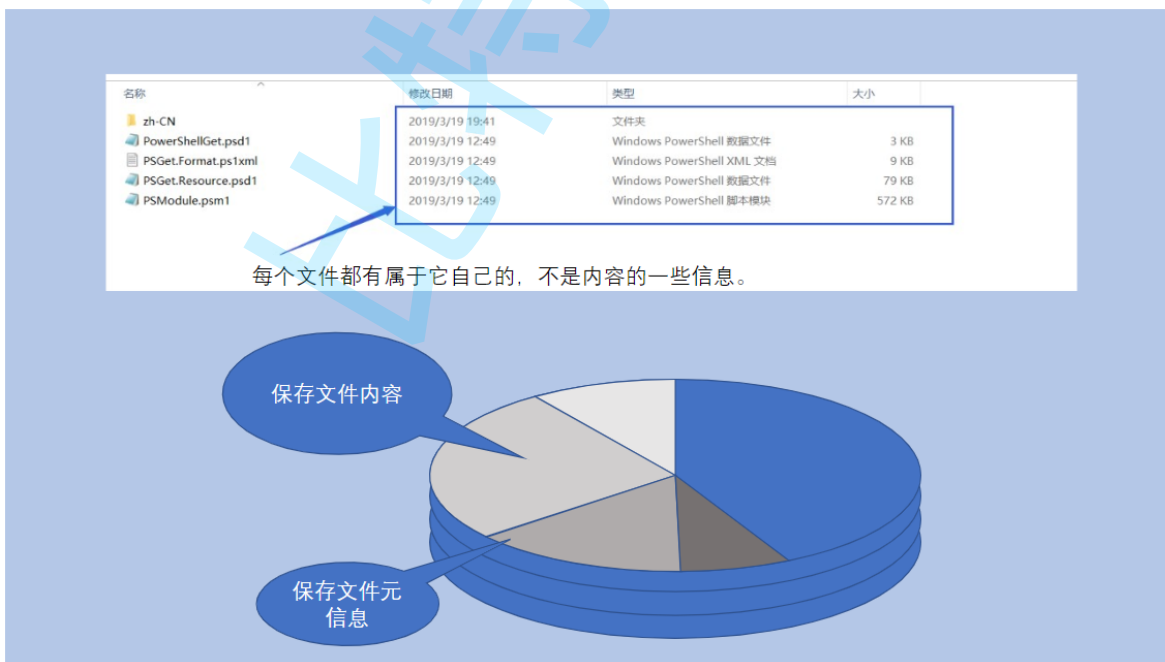
文件操作 —— IO

认识文件

我们先来认识狭义上的文件(file)。针对硬盘这种持久化存储的I/O设备，当我们想要进行数据保存时，往往不是保存成一个整体，而是独立成一个个的单位进行保存，这个独立的单位就被抽象成文件的概念，就类似办公桌上的一份份真实的文件一般。

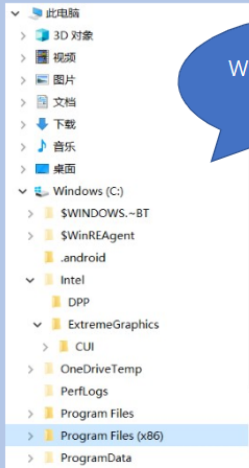


文件除了有数据内容之外，还有一部分信息，例如文件名、文件类型、文件大小等并不作为文件的数据而存在，我们把这部分信息可以视为文件的元信息。



树型结构组织 和 目录

同时，随着文件越来越多，对文件的系统管理也被提上了日程，如何进行文件的组织呢，一种合乎自然的想法出现了，就是按照层级结构进行组织——也就是我们数据结构中学习过的树形结构。这样，一种专门用来存放管理信息的特殊文件诞生了，也就是我们平时所谓文件夹(folder)或者目录(directory)的概念。



Windows 上的树型结构组织

```
-- bin -> usr/bin
boot
-- config-3.10.0-1160.11.1.el7.x86_64
-- efi
-- EFI
-- centos
-- grub
-- splash.xpm.gz
-- grub2
-- device.map
-- fonts
-- unicode.pf2
-- grub.cfg
-- grub.cfg.1572966586.rpmsave
-- grubenv
-- i386-pc
-- acpi.mod
-- adier32.mod
-- affs.mod
-- affs.mod
-- ahci.mod
-- all_video.mod
-- aout.mod
-- archeip.mod
-- ata.mod
-- at_keyboard.mod
-- backtrace.mod
```

Linux 上的树型结构组织

文件夹名称	创建时间	类型
Apowersoft	2021/3/5 15:36	文件夹
Bonjour	2021/3/5 15:37	文件夹
Common Files	2021/4/14 23:42	文件夹
Google	2021/5/8 10:29	文件夹
InstallShield Installation Information	2020/9/9 10:19	文件夹
Intel	2020/7/30 13:28	文件夹
Internet Explorer	2020/9/3 18:57	文件夹
Microsoft	2021/5/30 9:35	文件夹
Microsoft.NET	2019/12/20 17:04	文件夹
Mozilla Maintenance Service	2021/6/21 18:16	文件夹
MySQL	2020/11/25 18:22	文件夹
NetSarang	2020/9/9 10:19	文件夹
NVIDIA Corporation	2020/10/11 18:49	文件夹
QQMailPlugin	2020/9/1 11:26	文件夹
Tencent	2021/3/2 18:23	文件夹
Windows Defender	2021/5/17 0:14	文件夹
Windows Mail	2021/2/28 16:23	文件夹
Windows Media Player	2021/1/17 23:52	文件夹
Windows Multimedia Platform	2019/3/19 19:43	文件夹
Windows NT	2019/3/19 13:02	文件夹
Windows Photo Viewer	2021/1/17 23:52	文件夹
Windows Portable Devices	2019/3/19 19:43	文件夹
WindowsPowerShell	2019/3/19 12:52	文件夹

文件夹(folder)
目录(directory)
中保存的其实就是我们之前提到的关于文件的元信息。

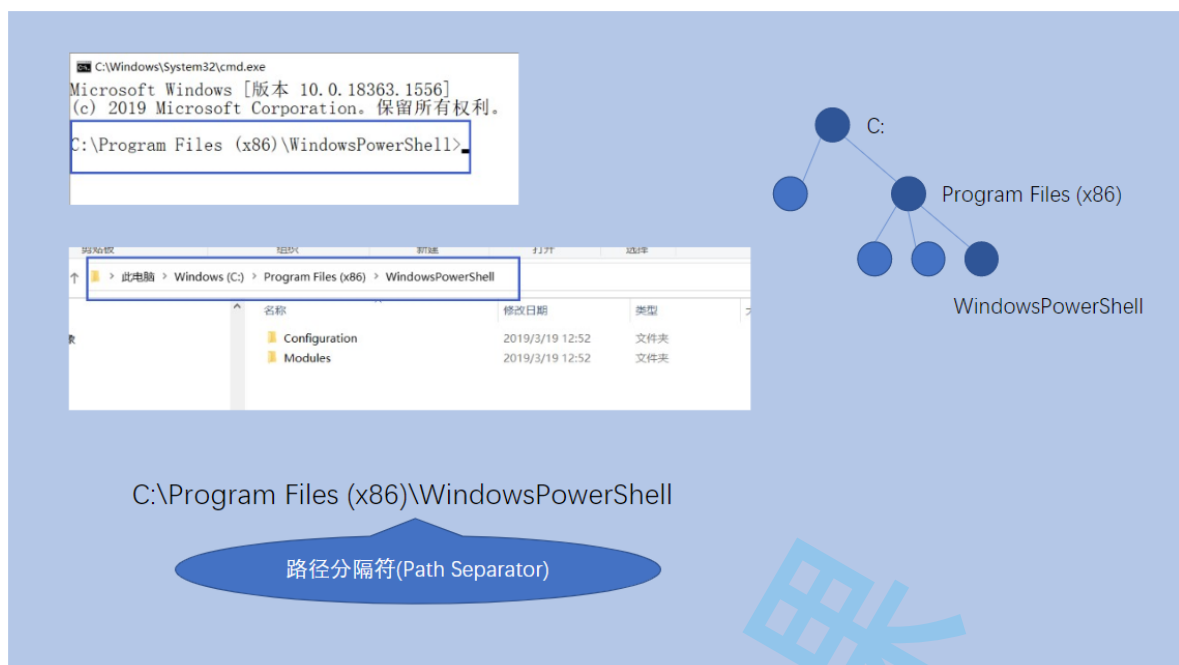
通过一个个的文件夹，我们可以将文件组织起来，更方便用户使用它。

逻辑上也更容易理解。



文件路径 (Path)

如何在文件系统中如何定位我们的一个唯一的文件就成为当前要解决的问题，但这难不倒计算机科学家，因为从树型结构的角度来看，树中的每个结点都可以被一条从根开始，一直到达的结点的路径所描述，而这种描述方式就被称为文件的绝对路径 (absolute path)。



除了可以从根开始进行路径的描述，我们可以从任意结点出发，进行路径的描述，而这种描述方式就被称为相对路径（relative path），相对于当前所在结点的一条路径。



其他知识

即使是普通文件，根据其保存数据的不同，也经常被分为不同的类型，我们一般简单的划分为文本文件和二进制文件，分别指代保存被字符集编码的文本和按照标准格式保存的非被字符集编码过的文件。

```
[root@VM-52-199-centos ~]# hexdump 我是中国人.txt
00000000 bde4 e5a0 bda5 bcef e68c 9188 98e6 e4af
00000010 adb8 9be5 e4bd baba 000a
00000019
```

```
[root@VM-52-199-centos ~]# cat 我是中国人.txt
你好，我是中国人
```

utf-8 编码	文本
0xe4bda0	你
0xe5a5bd	好
0xefbc8c	,
0xe68891	我
0xe698af	是
0xe4b8ad	中
0xe59bbd	国
0xe4baba	人
0x0a	\n

Name	Size	Offset	Description
Header			
Signature	2 bytes	0000h	Windows Structure: BITMAPFILEHEADER
FileSize	4 bytes	0000h	File size in bytes
reserved	4 bytes	0000h	unused (=0)
DataOffset	4 bytes	0000h	Offset from beginning of file to the beginning of the bitmap data
InfoHeader			
Size	4 bytes	0000h	Windows Structure: BITMAPINFOHEADER
Width	4 bytes	0000h	Size of InfoHeader = 40
Height	4 bytes	0000h	Horizontal width of bitmap in pixels
Planes	2 bytes	0000h	Vertical height of bitmap in pixels
Bits Per Pixel	2 bytes	0000h	Number of Planes (=1)
Compression	4 bytes	0000h	Bits per Pixel used to store palette entry information. This also identifies in an indirect way the number of possible colors. Possible values are: 1 = monochrome palette, NumColors = 1 4 = 4-bit palette, NumColors = 16 8 = 8-bit palette, NumColors = 256 16 = 16-bit RGB, NumColors = 65536 24 = 24-bit RGB, NumColors = 16777216
ImageSize	4 bytes	0000h	Type of Compression: 0 = BI_RGB, no compression 1 = BI_RLE8, 8-bit RLE 2 = BI_RLE4, 4-bit RLE
XpixelsPerM	4 bytes	0000h	(compressed) Size of image in bytes
YpixelsPerM	4 bytes	0000h	horizontal resolution: pixels per meter
Colors Used	4 bytes	0000h	vertical resolution: pixels per meter
Important Colors	4 bytes	0000h	Number of actual colors used

24 位位图的格式及数据

```
[root@VM-52-199-centos ~]# hexdump 我是个位图.bmp | less
```

```
00000000 4d42 1ee6 0004 0000 0000 0036 0000 0028
00000010 0000 012c 0000 012c 0000 0001 0018 0000
00000020 0000 1eb0 0004 0000 0000 0000 0000 0000
00000030 0000 0000 0000 0000 1c24 24ed ed1c 1c24 24ed
00000040 ed1c 1c24 24ed ed1c 1c24 24ed ed1c 1c24
00000050 24ed ed1c 1c24 24ed ed1c 1c24 24ed ed1c
00000060 1c24 24ed ed1c 1c24 24ed ed1c 1c24 24ed
00000070 ed1c 1c24 24ed ed1c 1c24 24ed ed1c 1c24
00000080 24ed ed1c 1c24 24ed ed1c 1c24 24ed ed1c
00000090 1c24 24ed ed1c 1c24 24ed ed1c 1c24 24ed
000000a0 ed1c 1c24 24ed ed1c 1c24 24ed ed1c 1c24
000000b0 24ed ed1c 1c24 24ed ed1c 1c24 24ed ed1c
000000c0 1c24 24ed ed1c 1c24 24ed ed1c 1c24 24ed
```

Windows 操作系统上，会按照文件名中的后缀来确定文件类型以及该类型文件的默认打开程序。但这个习俗并不是通用的，在 OSX、Unix、Linux 等操作系统上，就没有这样的习惯，一般不对文件类型做如此精确地分类。

21-CN	2019/3/19 19:43	文件夹	
mpvis.DLL	2021/1/16 16:08	应用程序扩展	160 KB
setup_wm.exe	2021/1/16 16:08	应用程序	1,760 KB
wmlaunch.exe	2021/1/16 16:08	应用程序	71 KB
wmpconfig.exe	2019/11/22 9:42	应用程序	100 KB
wmplayer.exe	2019/11/22 9:42	应用程序	163 KB
WMPMediaSharing.dll	2021/1/16 16:08	应用程序扩展	99 KB
wmpnsscdll	2021/1/16 16:08	应用程序扩展	422 KB
WMPNSSUI.dll	2019/3/19 19:42	应用程序扩展	18 KB
wmprph.exe	2021/1/16 16:08	应用程序	66 KB
wmpshare.exe	2019/11/22 9:42	应用程序	102 KB

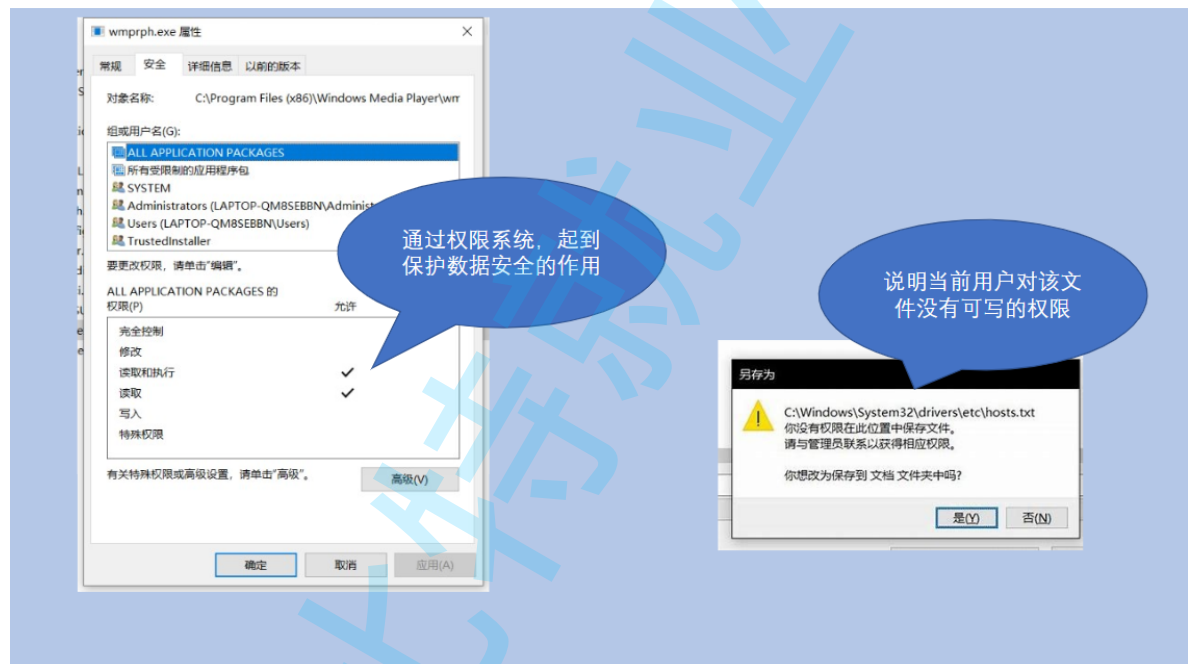
类似 exe、dll 这种，一般被称为文件的扩展名（extension）或者后缀（suffix）

不同的后缀往往代表不同的文件格式定义，起到不同的作用，例如：

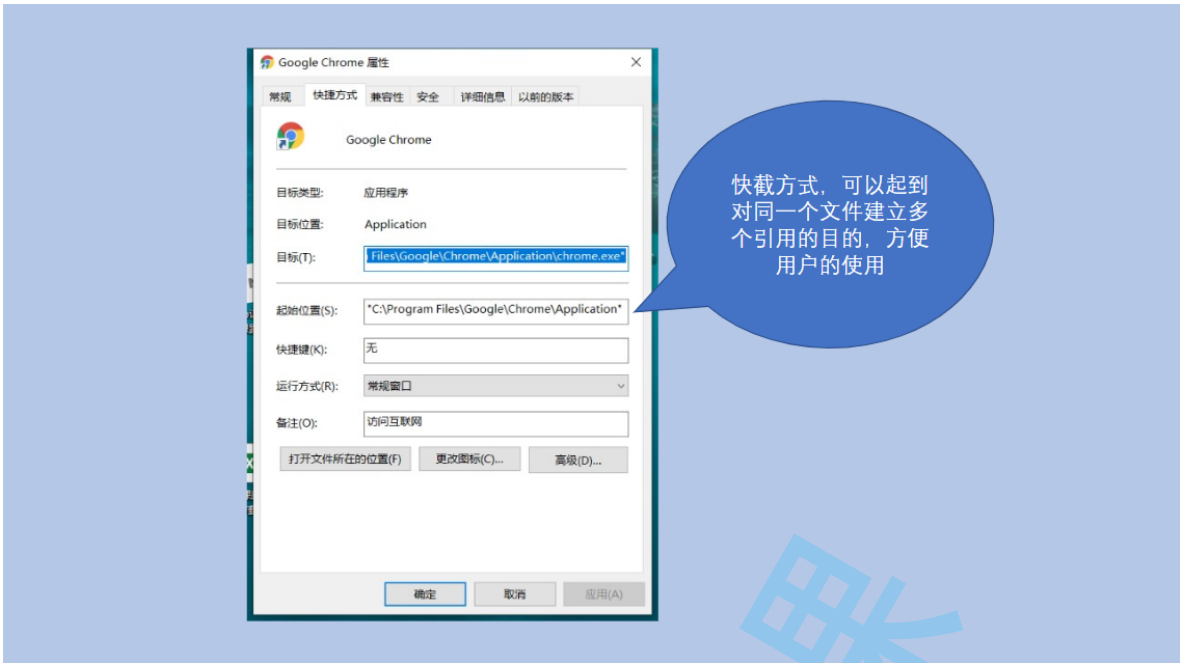
exe 在 Windows 中一般作为可执行程序来理解；

dll 在 Windows 中一般作为动态库（不包含入口的一组程序）来理解。

文件由于被操作系统进行了管理，所以根据不同的用户，会赋予用户不同的对待该文件的权限，一般地可以认为有可读、可写、可执行权限。



Windows 操作系统上，还有一类文件比较特殊，就是平时我们看到的快捷方式（shortcut），这种文件只是对真实文件的一种引用而已。其他操作系统上也有类似的概念，例如，软链接（soft link）等。



最后，很多操作系统为了实现接口的统一性，将所有的 I/O 设备都抽象成了文件的概念，使用这一理念最为知名的就是 Unix、Linux 操作系统——万物皆文件。

Java 中操作文件

本节内容中，我们主要涉及文件的元信息、路径的操作，暂时不涉及关于文件中内容的读写操作。

Java 中通过 `java.io.File` 类来对一个文件（包括目录）进行抽象的描述。注意，有 `File` 对象，并不代表真实存在该文件。

File 概述

我们先来看看 `File` 类中的常见属性、构造方法和方法

属性

修饰符及类型	属性	说明
static String	<code>pathSeparator</code>	依赖于系统的路径分隔符，String 类型的表示
static char	<code>pathSeparator</code>	依赖于系统的路径分隔符，char 类型的表示

构造方法

签名	说明
<code>File(File parent, String child)</code>	根据父目录 + 孩子文件路径，创建一个新的 File 实例
<code>File(String pathname)</code>	根据文件路径创建一个新的 File 实例，路径可以是绝对路径或者相对路径
<code>File(String parent, String child)</code>	根据父目录 + 孩子文件路径，创建一个新的 File 实例，父目录用路径表示

方法

修饰符及返回值类型	方法签名	说明
String	getParent()	返回 File 对象的父目录文件路径
String	getName()	返回 File 对象的纯文件名称
String	getPath()	返回 File 对象的文件路径
String	getAbsolutePath()	返回 File 对象的绝对路径
String	getCanonicalPath()	返回 File 对象的修饰过的绝对路径
boolean	exists()	判断 File 对象描述的文件是否真实存在
boolean	isDirectory()	判断 File 对象代表的文件是否是一个目录
boolean	isFile()	判断 File 对象代表的文件是否是一个普通文件
boolean	createNewFile()	根据 File 对象，自动创建一个空文件。成功创建后返回 true
boolean	delete()	根据 File 对象，删除该文件。成功删除后返回 true
void	deleteOnExit()	根据 File 对象，标注文件将被删除，删除动作会到 JVM 运行结束时才会进行
String[]	list()	返回 File 对象代表的目录下的所有文件名
File[]	listFiles()	返回 File 对象代表的目录下的所有文件，以 File 对象表示
boolean	mkdir()	创建 File 对象代表的目录
boolean	mkdirs()	创建 File 对象代表的目录，如果必要，会创建中间目录
boolean	renameTo(File dest)	进行文件改名，也可以视为我们平时的剪切、粘贴操作
boolean	canRead()	判断用户是否对文件有可读权限
boolean	canWrite()	判断用户是否对文件有可写权限

代码示例

示例1

观察 get 系列的特点和差异

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("../hello-world.txt");    // 并不要求该文件真实存在
        System.out.println(file.getParent());
        System.out.println(file.getName());
        System.out.println(file.getPath());
        System.out.println(file.getAbsolutePath());
        System.out.println(file.getCanonicalPath());
    }
}
```

运行结果

```
..
hello-world.txt
..\hello-world.txt
D:\代码练习\文件示例1\..\hello-world.txt
D:\代码练习\hello-world.txt
```

示例2

普通文件的创建、删除

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("hello-world.txt");    // 要求该文件不存在，才能看到相同
        的现象
        System.out.println(file.exists());
        System.out.println(file.isDirectory());
        System.out.println(file.isFile());
        System.out.println(file.createNewFile());
        System.out.println(file.exists());
        System.out.println(file.isDirectory());
        System.out.println(file.isFile());
        System.out.println(file.createNewFile());
    }
}
```

运行结果

```
false
false
false
true
true
false
true
false
```


示例3

普通文件的删除

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("some-file.txt"); // 要求该文件不存在，才能看到相同的现象
        System.out.println(file.exists());
        System.out.println(file.createNewFile());
        System.out.println(file.exists());
        System.out.println(file.delete());
        System.out.println(file.exists());
    }
}
```

运行结果

```
false
true
true
true
false
```

示例4

观察 deleteOnExit 的现象

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("some-file.txt"); // 要求该文件不存在，才能看到相同的现象
        System.out.println(file.exists());
        System.out.println(file.createNewFile());
        System.out.println(file.exists());
        file.deleteOnExit();
        System.out.println(file.exists());
    }
}
```

运行结果

```
false
true
true
true
```

程序运行结束后，文件还是被删除了

示例5

观察目录的创建

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File dir = new File("some-dir");    // 要求该目录不存在，才能看到相同的现象
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
        System.out.println(dir.mkdir());
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
    }
}
```

运行结果

```
false
false
true
true
false
```

示例6

观察目录创建2

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File dir = new File("some-parent\\some-dir");    // some-parent 和 some-dir 都不存在
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
        System.out.println(dir.mkdir());
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
    }
}
```

运行结果

```
false
false
false
false
false
```

mkdir() 的时候，如果中间目录不存在，则无法创建成功; mkdirs() 可以解决这个问题。

```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File dir = new File("some-parent\\some-dir"); // some-parent 和 some-dir 都不存在
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
        System.out.println(dir.mkdirs());
        System.out.println(dir.isDirectory());
        System.out.println(dir.isFile());
    }
}
```

运行结果

```
false
false
true
true
false
```

示例7

观察文件重命名

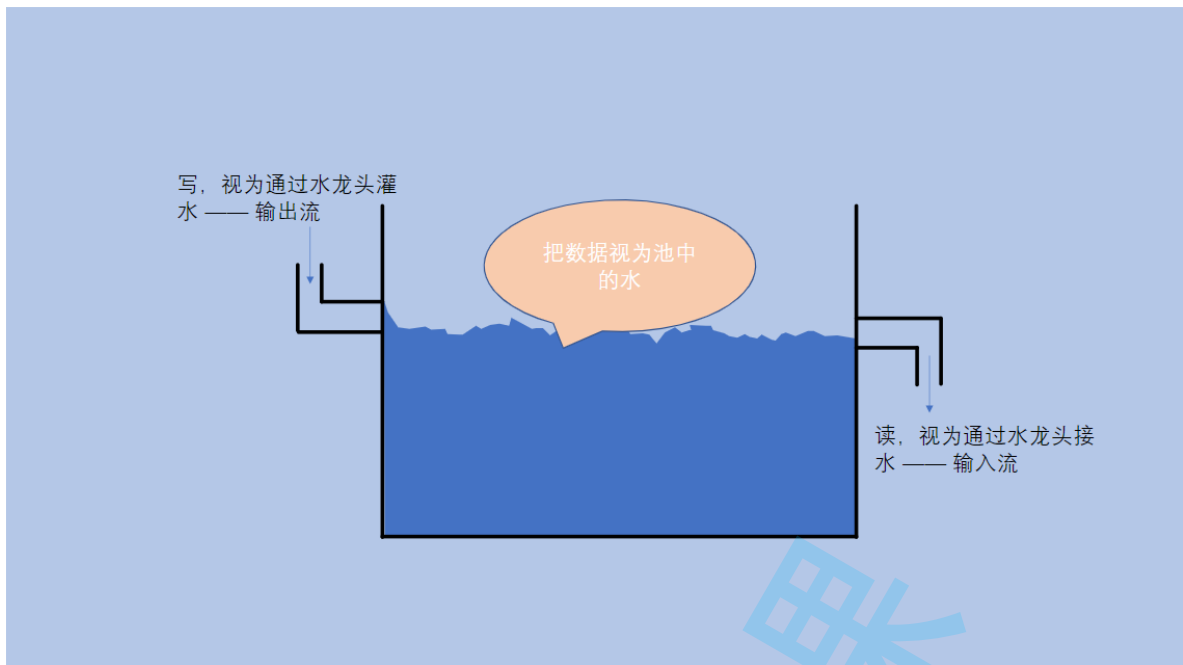
```
import java.io.File;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("some-file.txt"); // 要求 some-file.txt 得存在，可以是普通文件，可以是目录
        File dest = new File("dest.txt"); // 要求 dest.txt 不存在
        System.out.println(file.exists());
        System.out.println(dest.exists());
        System.out.println(file.renameTo(dest));
        System.out.println(file.exists());
        System.out.println(dest.exists());
    }
}
```

运行结果

```
true
false
true
false
true
```

文件内容的读写 —— 数据流



InputStream 概述

方法

修饰符及返回值类型	方法签名	说明
int	read()	读取一个字节的数据, 返回 -1 代表已经完全读完了
int	read(byte[] b)	最多读取 b.length 字节的数据到 b 中, 返回实际读到的数量; -1 代表以及读完了
int	read(byte[] b, int off, int len)	最多读取 len - off 字节的数据到 b 中, 放在从 off 开始, 返回实际读到的数量; -1 代表以及读完了
void	close()	关闭字节流

说明

InputStream 只是一个抽象类, 要使用还需要具体的实现类。关于 InputStream 的实现类有很多, 基本可以认为不同的输入设备都可以对应一个 InputStream 类, 我们现在只关心从文件中读取, 所以使用 **FileInputStream**

FileInputStream 概述

构造方法

签名	说明
FileInputStream(File file)	利用 File 构造文件输入流
FileInputStream(String name)	利用文件路径构造文件输入流

代码示例

示例1

将文件完全读完的两种方式。相比较而言，后一种的 IO 次数更少，性能更好。

```
import java.io.*;

// 需要先在项目目录下准备好一个 hello.txt 的文件，里面填充 "Hello" 的内容
public class Main {
    public static void main(String[] args) throws IOException {
        try (InputStream is = new FileInputStream("hello.txt")) {
            while (true) {
                int b = is.read();
                if (b == -1) {
                    // 代表文件已经全部读完
                    break;
                }

                System.out.printf("%c", b);
            }
        }
    }
}
```

```
import java.io.*;

// 需要先在项目目录下准备好一个 hello.txt 的文件，里面填充 "Hello" 的内容
public class Main {
    public static void main(String[] args) throws IOException {
        try (InputStream is = new FileInputStream("hello.txt")) {
            byte[] buf = new byte[1024];
            int len;

            while (true) {
                len = is.read(buf);
                if (len == -1) {
                    // 代表文件已经全部读完
                    break;
                }

                for (int i = 0; i < len; i++) {
                    System.out.printf("%c", buf[i]);
                }
            }
        }
    }
}
```

示例2

这里我们把文件内容中填充中文看看，注意，写中文的时候使用 UTF-8 编码。hello.txt 中填写 "你好中国"

注意：这里我利用了这几个中文的 UTF-8 编码后长度刚好是 3 个字节和长度不超过 1024 字节的现状，但这种方式并不是通用的

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

```
// 需要先在项目目录下准备好一个 hello.txt 的文件，里面填充 "你好中国" 的内容
public class Main {
    public static void main(String[] args) throws IOException {
        try (InputStream is = new FileInputStream("hello.txt")) {
            byte[] buf = new byte[1024];
            int len;

            while (true) {
                len = is.read(buf);
                if (len == -1) {
                    // 代表文件已经全部读完
                    break;
                }

                // 每次使用 3 字节进行 utf-8 解码，得到中文字符
                // 利用 String 中的构造方法完成
                // 这个方法了解下即可，不是通用的解决办法
                for (int i = 0; i < len; i += 3) {
                    String s = new String(buf, i, 3, "UTF-8");
                    System.out.printf("%s", s);
                }
            }
        }
    }
}
```

利用 Scanner 进行字符读取

上述例子中，我们看到了对字符类型直接使用 InputStream 进行读取是非常麻烦且困难的，所以，我们使用一种我们之前比较熟悉的类来完成该工作，就是 Scanner 类。

构造方法	说明
Scanner(InputStream is, String charset)	使用 charset 字符集进行 is 的扫描读取

示例1

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

// 需要先在项目目录下准备好一个 hello.txt 的文件，里面填充 "你好中国" 的内容
public class Main {
    public static void main(String[] args) throws IOException {
        try (InputStream is = new FileInputStream("hello.txt")) {
            try (Scanner scanner = new Scanner(is, "UTF-8")) {
                while (scanner.hasNext()) {
                    String s = scanner.next();
                    System.out.print(s);
                }
            }
        }
    }
}
```

OutputStream 概述

方法

修饰符及返回值类型	方法签名	说明
void	write(int b)	写入要给字节的数据
void	write(byte[] b)	将 b 这个字符数组中的数据全部写入 os 中
int	write(byte[] b, int off, int len)	将 b 这个字符数组中从 off 开始的数据写入 os 中，一共写 len 个
void	close()	关闭字节流
void	flush()	重要：我们知道 I/O 的速度是很慢的，所以，大多的 OutputStream 为了减少设备操作的次数，在写数据的时候都会将数据先暂时写入内存的一个指定区域里，直到该区域满了或者其他指定条件时才真正将数据写入设备中，这个区域一般称为缓冲区。但造成一个结果，就是我们写的的数据，很可能会遗留一部分在缓冲区中。需要在最后或者合适的位置，调用 flush（刷新）操作，将数据刷到设备中。

说明

OutputStream 同样只是一个抽象类，要使用还需要具体的实现类。我们现在还是只关心写入文件中，所以使用 **FileOutputStream**

利用 OutputStreamWriter 进行字符写入

示例1

```
import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            os.write('H');
            os.write('e');
            os.write('l');
            os.write('l');
            os.write('o');
            // 不要忘记 flush
            os.flush();
        }
    }
}
```

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



```

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            byte[] b = new byte[] {
                (byte)'G', (byte)'o', (byte)'o', (byte)'d'
            };
            os.write(b);

            // 不要忘记 flush
            os.flush();
        }
    }
}

```

```

import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            byte[] b = new byte[] {
                (byte)'G', (byte)'o', (byte)'o', (byte)'d', (byte)'B',
                (byte)'a', (byte)'d'
            };
            os.write(b, 0, 4);

            // 不要忘记 flush
            os.flush();
        }
    }
}

```

```

import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            String s = "Nothing";
            byte[] b = s.getBytes();
            os.write(b);

            // 不要忘记 flush
            os.flush();
        }
    }
}

```

```

import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            String s = "你好中国";
            byte[] b = s.getBytes("utf-8");
            os.write(b);
        }
    }
}

```

```

        // 不要忘记 flush
        os.flush();
    }
}

```

利用 PrintWriter 找到我们熟悉的方法

上述，我们其实已经完成输出工作，但总是有所不方便，我们接下来将 OutputStream 处理下，使用 PrintWriter 类来完成输出，因为

PrintWriter 类中提供了我们熟悉的 print/println/printf 方法

```

OutputStream os = ...;
OutputStreamWriter oswriter = new OutputStreamWriter(os, "utf-8"); // 告诉它，我
们的字符集编码是 utf-8 的
PrintWriter writer = new PrintWriter(oswriter);

// 接下来我们就可以方便的使用 writer 提供的各种方法了
writer.print("Hello");
writer.println("你好");
writer.printf("%d: %s\n", 1, "没什么");

// 不要忘记 flush
writer.flush();

```

示例1

```

import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        try (OutputStream os = new FileOutputStream("output.txt")) {
            try (OutputStreamWriter oswriter = new OutputStreamWriter(os, "UTF-
8")) {
                try (PrintWriter writer = new PrintWriter(oswriter)) {
                    writer.println("我是第一行");
                    writer.print("我的第二行\r\n");
                    writer.printf("%d: 我的第三行\r\n", 1 + 1);
                    writer.flush();
                }
            }
        }
    }
}

```

小程序练习

我们学会了文件的基本操作 + 文件内容读写操作，接下来，我们实现一些小工具程序，来锻炼我们的能力。

示例1

扫描指定目录，并找到名称中包含指定字符的所有普通文件（不包含目录），并且后续询问用户是否要删除该文件

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

public class Main {
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入要扫描的根目录（绝对路径 OR 相对路径）：");
        String rootDirPath = scanner.next();

        File rootDir = new File(rootDirPath);
        if (!rootDir.isDirectory()) {
            System.out.println("您输入的根目录不存在或者不是目录，退出");
            return;
        }

        System.out.print("请输入要找出的文件名中的字符：");
        String token = scanner.next();

        List<File> result = new ArrayList<>();
        // 因为文件系统是树形结构，所以我们使用深度优先遍历（递归）完成遍历
        scanDir(rootDir, token, result);
        System.out.println("共找到了符合条件的文件 " + result.size() + " 个，它们分别是");

        for (File file : result) {
            System.out.println(file.getCanonicalPath() + "    请问您是否要删除该文件? y/n");
            String in = scanner.next();
            if (in.toLowerCase().equals("y")) {
                file.delete();
            }
        }
    }

    private static void scanDir(File rootDir, String token, List<File> result) {
        File[] files = rootDir.listFiles();
        if (files == null || files.length == 0) {
            return;
        }

        for (File file : files) {
            if (file.isDirectory()) {
                scanDir(file, token, result);
            } else {
                if (file.getName().contains(token)) {
                    result.add(file.getAbsolutePath());
                }
            }
        }
    }
}
```

示例2

进行普通文件的复制

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

public class Main {
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);

        System.out.print("请输入要复制的文件（绝对路径 OR 相对路径）：");
        String sourcePath = scanner.next();
        File sourceFile = new File(sourcePath);
        if (!sourceFile.exists()) {
            System.out.println("文件不存在，请确认路径是否正确");
            return;
        }

        if (!sourceFile.isFile()) {
            System.out.println("文件不是普通文件，请确认路径是否正确");
            return;
        }

        System.out.print("请输入要复制到的目标路径（绝对路径 OR 相对路径）：");
        String destPath = scanner.next();
        File destFile = new File(destPath);
        if (destFile.exists()) {
            if (destFile.isDirectory()) {
                System.out.println("目标路径已经存在，并且是一个目录，请确认路径是否正确");
                return;
            }

            if (destFile.isFile()) {
                System.out.println("目录路径已经存在，是否要进行覆盖？y/n");
                String ans = scanner.next();
                if (!ans.toLowerCase().equals("y")) {
                    System.out.println("停止复制");
                    return;
                }
            }
        }

        try (InputStream is = new FileInputStream(sourceFile)) {
            try (OutputStream os = new FileOutputStream(destFile)) {
                byte[] buf = new byte[1024];
                int len;

                while (true) {
                    len = is.read(buf);
                    if (len == -1) {
                        break;
                    }

                    os.write(buf, 0, len);
                }

                os.flush();
            }
        }
    }
}
```

```

        System.out.println("复制已完成");
    }
}

```

示例3

扫描指定目录，并找到名称或者内容中包含指定字符的所有普通文件（不包含目录）

注意：我们现在的方案性能较差，所以尽量不要在太复杂的目录下或者大文件下实验

```

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

public class Main {
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入要扫描的根目录（绝对路径 OR 相对路径）：");
        String rootDirPath = scanner.next();

        File rootDir = new File(rootDirPath);
        if (!rootDir.isDirectory()) {
            System.out.println("您输入的根目录不存在或者不是目录，退出");
            return;
        }

        System.out.print("请输入要找出的文件名中的字符：");
        String token = scanner.next();

        List<File> result = new ArrayList<>();
        // 因为文件系统是树形结构，所以我们使用深度优先遍历（递归）完成遍历
        scanDirWithContent(rootDir, token, result);
        System.out.println("共找到了符合条件的文件 " + result.size() + " 个，它们分别是");

        for (File file : result) {
            System.out.println(file.getCanonicalPath());
        }
    }

    private static void scanDirWithContent(File rootDir, String token,
        List<File> result) throws IOException {
        File[] files = rootDir.listFiles();
        if (files == null || files.length == 0) {
            return;
        }

        for (File file : files) {
            if (file.isDirectory()) {
                scanDirWithContent(file, token, result);
            } else {
                if (isContentContains(file, token)) {
                    result.add(file.getAbsolutePath());
                }
            }
        }
    }
}

```

```
// 我们全部按照utf-8的字符文件来处理
private static boolean isContentContains(File file, String token) throws
IOException {
    StringBuilder sb = new StringBuilder();
    try (InputStream is = new FileInputStream(file)) {
        try (Scanner scanner = new Scanner(is, "UTF-8")) {
            while (scanner.hasNextLine()) {
                sb.append(scanner.nextLine());
                sb.append("\r\n");
            }
        }
    }

    return sb.indexOf(token) != -1;
}
}
```

代码参考

如何按字节进行数据读

```
try (InputStream is = ...) {
    byte[] buf = new byte[1024];
    while (true) {
        int n = is.read(buf);
        if (n == -1) {
            break;
        }

        // buf 的 [0, n) 表示读到的数据，按业务进行处理
    }
}
```

如何按字节进行数据写

```
try (OutputStream os = ...) {
    byte[] buf = new byte[1024];
    while (/* 还有未完成的业务数据 */) {
        // 将业务数据填入 buf 中，长度为 n
        int n = ...;
        os.write(buf, 0, n);
    }
    os.flush(); // 进行数据刷新操作
}
```

如何按字符进行数据读

```
try (InputStream is = ...) {
    try (Scanner scanner = new Scanner(is, "UTF-8")) {
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();

            // 根据 line 做业务处理
        }
    }
}
```

如何按字符进行数据写

```
try (OutputStream os = ...) {
    try (OutputStreamWriter osWriter = new OutputStreamWriter(os, "UTF-8")) {
        try (PrintWriter writer = new PrintWriter(osWriter)) {
            while (/* 还有未完成的业务数据 */) {
                writer.println(...);
            }
            writer.flush(); // 进行数据刷新操作
        }
    }
}
```