Day01

java.io.File

概念

File的每一个实例用于表示一个文件或目录的

- 使用File可以:
  - 1:访问文件或目录的属性
  - 2:创建/删除文件或目录
  - 3:访问一个目录中的所有子项
- 但是File不能:访问文件数据

涉及的新单词:

- **file-文件**
- **write-写**
- **read-读**
- **hidden-隐藏**
- **create-创建**
- **make-作**
- **directory-目录**
- **exists-存在**
- **accept-接受**

File的创建

构造器

- **File(String pathName)**

  根据给定的路径创建File对象来表示。

- **File(File parent,String pathName)**

  表示parent表示的目录中指定的路径下对应的文件或目录

  其中parent表达的应当是一个目录

绝对路径与相对路径

- **绝对路径**：从系统定义的根目录开始，优点:清晰明了 缺点:不利于跨平台
  - "./":当前目录，不同的运行环境定位不同。在IDEA中执行java程序时"./"表示当前项目目录
    - 在相对路径中"./"是可以忽略不写的，默认就是从"./"开始
  - 类加载路径，后期使用很多的相对路径。在后面学习项目阶段的课程中会介绍到
- **相对路径**：从运行环境定义的路径开始，优点:跨平台 缺点:路径定义模糊，需要视环境而定

例

```
package file;

import java.io.File;

public class FileDemo {
    public static void main(String[] args) {
        /*
            创建File对象用来表示当前项目目录下的demo.txt文件
            参数:文件的路径
            路径分为两种:
            绝对路径:从系统定义的根目录开始，优点:清晰明了 缺点:不利于跨平台
            相对路径:从运行环境定义的路径开始，优点:跨平台 缺点:路径定义模糊，需要视环境而定
         */
//        File file = new
File("C:/Users/TEACHER/IdeaProjects/JSD2303_SE/demo.txt");
        /*
            当前程序是在IDEA中运行的，因此IDEA就是当前程序的运行环境
            相对路径就是根据不同的运行环境而定。
            "./":称为"当前目录"，在IDEA中当前目录指的是当前项目目录
         */
        File file = new File("./demo.txt");
    }
}
```

####

访问属性的相关方法

- **String getName()**

  返回当前File对象表示的文件或目录的名字

- **long length()**

  返回当前File对象表示的文件的长度，单位是字节

- **boolean canRead()**

  返回当前File对象表示的文件或目录是否可读

- **boolean canWrite()**

  返回当前File对象表示的文件或目录是否可写

- **boolean isHidden()**

  返回当前File对象表示的文件或目录是否被隐藏

- **boolean exists()**

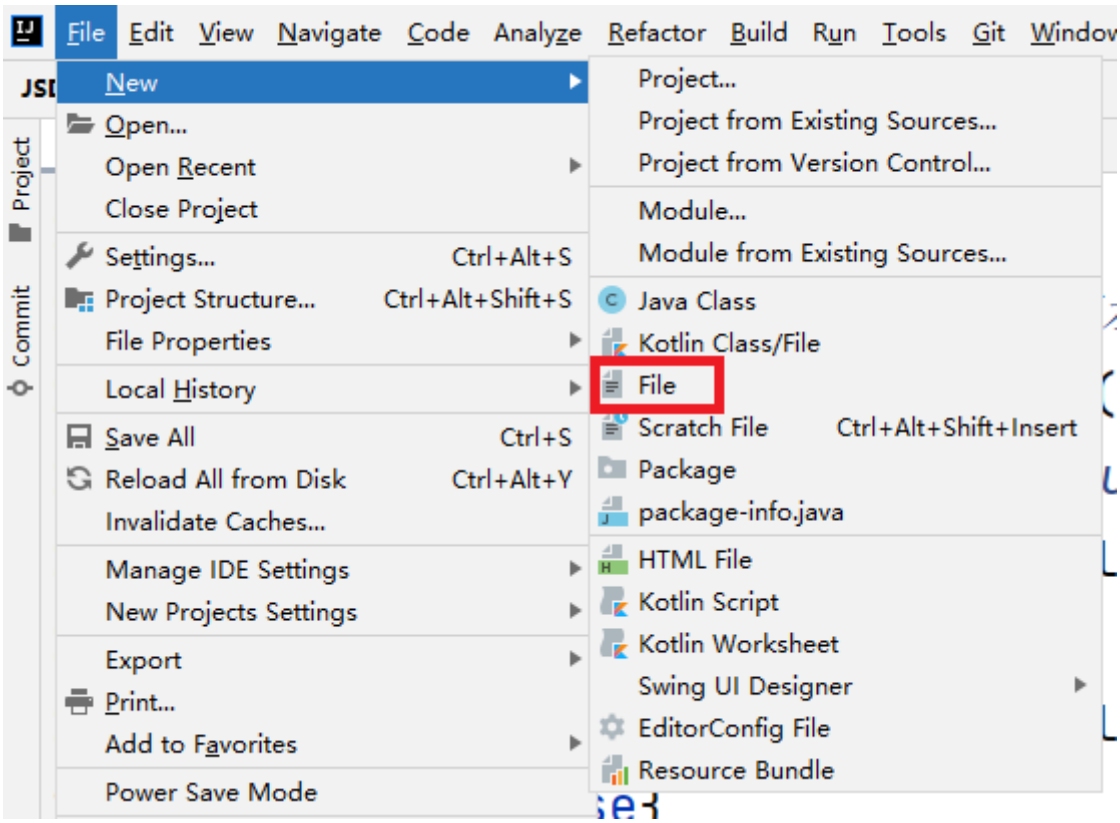
  判断当前File对象表示的文件或目录是否已经存在

例

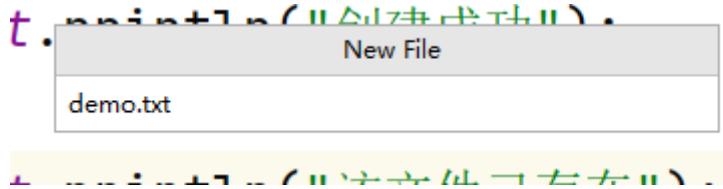
先手动在IDEA的当前项目目录下新建一个名为demo.txt的文件，并在文件中随意的输入些文字

具体操作:

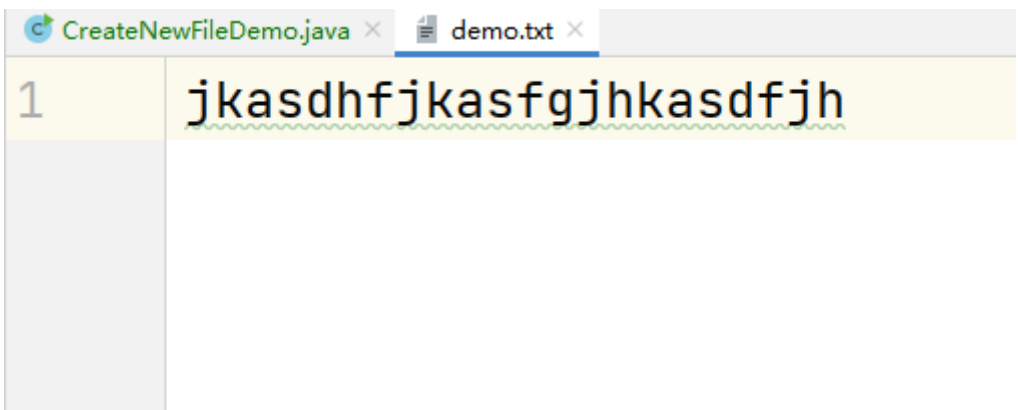
- IDEA菜单中File->New->File



- 在弹出框中输入demo.txt



- 双击打开这个文件随意输入些文字



```java
package file;

import java.io.File;

public class FileDemo {
    public static void main(String[] args) {
        File file = new File("./demo.txt");
        //获取名字
        String name = file.getName();
        System.out.println(name);
        //获取File表示的文件的长度(单位是字节)
        long len = file.length();
        System.out.println("占用的字节数:"+len);
        //是否可读
        boolean cr = file.canRead();
        //是否可写
        boolean cw = file.canWrite();
        //是否是被隐藏
        boolean ih = file.isHidden();
        System.out.println("可读:"+cr);
        System.out.println("可写:"+cw);
        System.out.println("是否隐藏:"+ih);

    }
}
```

创建与删除文件

文件的创建

方法

boolean createNewFile()

在当前File表示的路径下将文件创建出来。

如果指定的文件不存在并成功地创建，则返回**true**；如果指定的文件已经存在，则返回 **false**

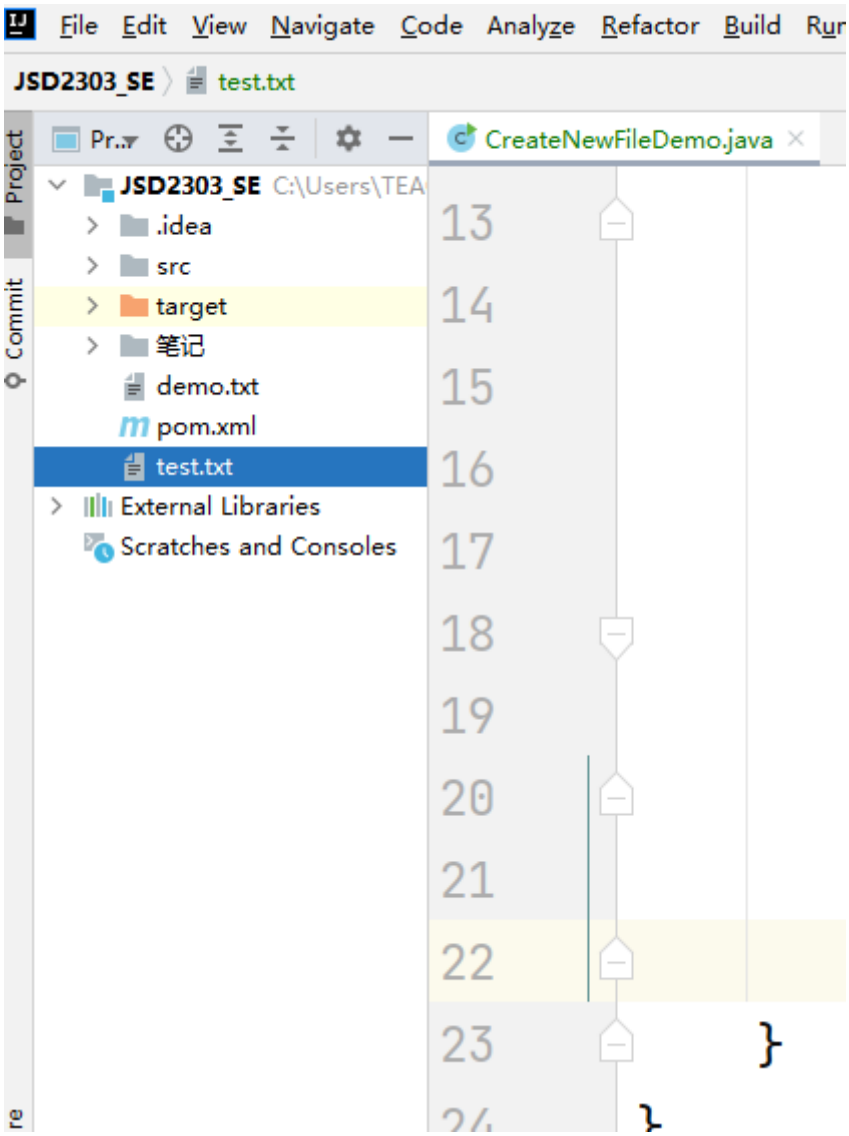
例

```java
package file;

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

/**
 * 使用File对象新建一个文件
 */
public class CreateNewFileDemo {
    public static void main(String[] args) throws IOException {
        /*
            在当前项目目录下新建一个名为:test.txt的文件
         */
```

```
        //1创建一个File对象来表示该路径
        File file = new File("./test.txt");
        //2创建这个文件  返回true:文件被创建  false:文件已经存在无法再创建
        boolean success = file.createNewFile();
        if(success){
            System.out.println("创建成功");
        }else{
            System.out.println("该文件已存在");
        }
    }
}
```



练习

在当前项目目录下新建100个文件，名字为test1.txt--test100.txt

```
package file;

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

/**
```

```
 * 在当前项目目录下新建100个文件
 * 名字为test1.txt--test100.txt
 *
 * 提示:
 * 1:使用for循环，循环变量控制在1-100
 * 2:可以将循环变量拼接起来形成文件名
 * 3:使用File对象表示这个文件名并将它创建出来
 */
public class Test1 {
    public static void main(String[] args) throws IOException {
        for(int i=1;i<=100;i++){
            File file = new File("./test"+i+".txt");
            file.createNewFile();
        }
        System.out.println("创建完毕！");
    }
}
```

文件的删除

方法

boolean delete()

删除当前File对象表示的文件。

当且仅当成功删除文件或目录时，返回**true**；否则返回**false**

例

```
package file;

import java.io.File;

/**
 * 删除一个文件
 */
public class DeleteFileDemo {
    public static void main(String[] args) {
        /*
            将当前项目目录下的test.txt文件删除
         */
        //1创建File对象表示要删除的文件
        //在相对路径中"./"是可以忽略不写的，默认就是从"./"开始
//        File file = new File("./test.txt");
        File file = new File("test.txt");

        file.delete();
        System.out.println("文件已删除");
    }
}
```

练习

删除当前项目目录下100个文件,test1.txt--test100.txt

```
package file;

import java.io.File;

/**
 * 删除当前项目目录下100个文件
 * test1.txt--test100.txt
 */
public class Test2 {
    public static void main(String[] args) {
        for(int i=1;i<=100;i++){
            File file = new File("./test"+i+".txt");
            file.delete();
        }
        System.out.println("创建完毕! ");
    }
}
```

创建与删除目录

目录的创建

方法

boolean mkdir()

将当前File表示的目录创建出来。**该方法要求当前File表示的目录所在的父目录必须存在，否则创建失败**

当且仅当已创建目录时，返回 **true**；否则返回 **false**

例

```
package file;

import java.io.File;

/**
 * 创建一个目录
 */
public class MkDirDemo {
    public static void main(String[] args) {
        //在当前项目目录下新建一个名为:demo的目录
        File dir = new File("demo");
        /*
            boolean exists()
            判断当前File对象表示的文件或目录是否已经存在，如果已经存在则返回true

            boolean mkdir()
            将当前File对象表示的目录创建出来，当且仅当目录成功创建时返回true
```

```
         */
        if(!dir.exists()) {
            dir.mkdir();
            System.out.println("该目录已创建");
        }else{
            System.out.println("该目录已存在");
        }

    }
}
```

创建多级目录

方法

boolean mkdirs()

创建当前File对象表示的目录，并且会**自动创建所有不存在的父目录**

当且仅当已创建目录以及所有必需的父目录时，返回**true**；否则返回 **false**

实际开发中推荐用该方法创建目录

例

```
package file;

import java.io.File;

/**
 * 创建多级目录
 */
public class MkDirsDemo {
    public static void main(String[] args) {
        /*
            在当前目录下新建:a/b/c/d/e/f目录
         */
        //当前dir表达式的是f目录，只不过f是在前面对应的里面
        File dir = new File("a/b/c/d/e/f");
        if(dir.exists()){
            System.out.println("该目录已存在!");
        }else{
            /*
                创建失败
                原因:mkdir()方法在创建目录时要求该目录所在的父目录必须真实存在
                    否则创建失败
             */
//            dir.mkdir();
            /*
                mkdirs()方法在创建目录时会自动将不存在的父目录一同创建出来
                实际开发中推荐用这个方法创建目录
             */
            dir.mkdirs();
            System.out.println("该目录已创建!");
        }
```

```
    }
}
```

目录的删除

方法

boolean delete()

与删除文件是同一个方法，如果File表示的是目录也可以被删除

删除目录时要求该目录必须是一个空目录，否则删除失败

例

```
package file;

import java.io.File;

/**
 * 删除目录
 */
public class DeleteDirDemo {
    public static void main(String[] args) {
        //将当前项目目录下的demo目录删除
        //File dir = new File("./demo");
        File dir = new File("./a");//删除失败，因为a不是一个空目录
        if(dir.exists()){
            dir.delete();
            System.out.println("该目录已删除");
        }else{
            System.out.println("该目录不存在");
        }
    }
}
```

访问目录内容

访问一个目录中的所有子项

方法

FIle[] listFiles()

获取当前File对象表示的目录中的所有子项。返回的数组中每个FIle对象表示该目录中的一个子项。

相关方法

boolean isFile()

判断当前File对象表达的是否为一个真实存在的文件

boolean isDirectory()

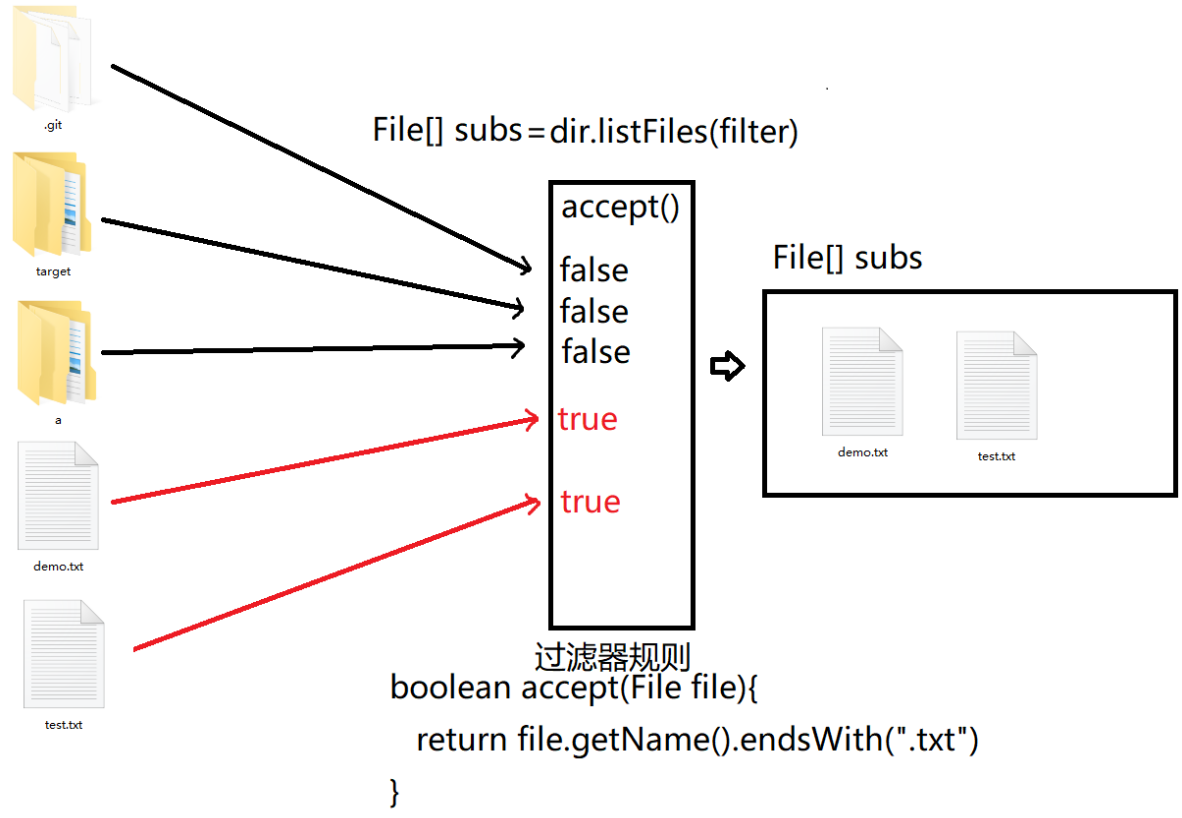
判断当前File对象表达的是否为一个真实存在的目录

例

```
package file;

import java.io.File;

/**
 * 获取一个目录中的所有子项
 */
public class ListFilesDemo {
    public static void main(String[] args) {
        //将当前目录中的所有子项列出
        //1创建File对象表示当前目录
        File dir = new File(".");
        if(dir.isDirectory()) {//存在且确实为一个目录
            //2列出该目录中的所有子项
            File[] subs = dir.listFiles();
            System.out.println("一共有"+subs.length+"个子项");
            //遍历数组，获取每一个子项
            for(File sub : subs){
                String name = sub.getName();//获取该File对象表示的文件或目录的名字
                System.out.println(name);
            }
        }
    }
}
```



练习

定义一个过滤器，获取当前目录下子项名字中含有字母"a"的所有子项

```
package file;

import java.io.File;
import java.io.FileFilter;

/**
 * 定义一个过滤器，获取当前目录下子项名字中含有字母"a"的所有子项
 *
 * indexOf("a")是否>=0，因为当字符串不含有"a"时indexOf返回值为-1
 *
 * String提供了直接判断是否包含指定内容的方法
 * boolean contains(String str)
 * 当前字符串包含参数str表示的字符串内容时则返回true
 */
public class Test3 {
    public static void main(String[] args) {
        File dir = new File(".");
        if(dir.isDirectory()){
//            FileFilter filter = new FileFilter() {
//                public boolean accept(File f) {
//                    return f.getName().contains("a");
//                }
//            };
//            File[] subs = dir.listFiles(filter);
            /*
                FileFilter接口只有一个抽象方法
                因此可以使用lambda表达式创建
             */
            File[] subs = dir.listFiles(f->f.getName().contains("a"));
            for(File sub : subs){
                System.out.println(sub.getName());
            }
        }
    }
}
```

JAVA IO

基本概念

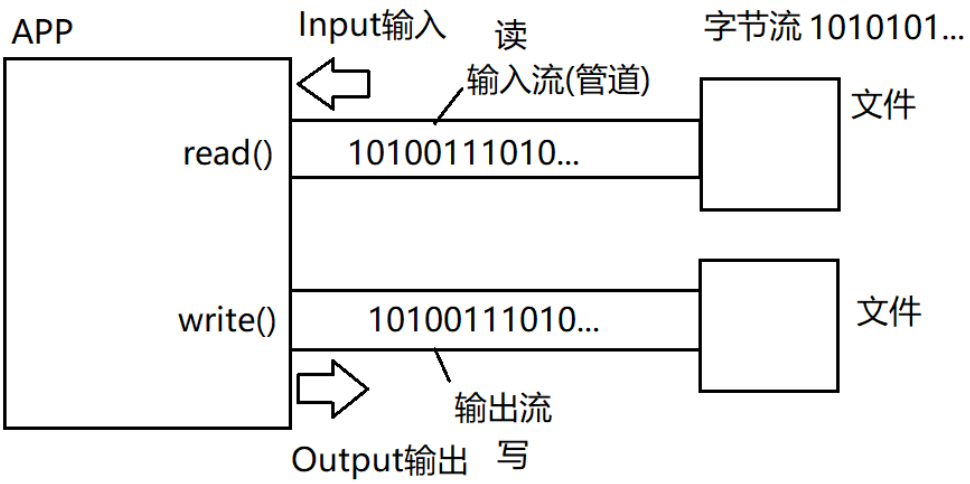
I/O 这里的I和O指的是输入与输出

- 输入:Input　用来读取数据的
- 输出:Output　用来写出数据的

流的概念

java将输入与输出比喻为"流"，英文:**Stream**.

就像生活中的"电流","水流"一样,它是以同一个方向顺序移动的过程.只不过这里流动的是字节(2进制数据).所以在IO中有输入流和输出流之分,我们理解他们是连接程序与另一端的"管道",用于获取或发送数据到另一端.



超类

- **java.io.InputStream**是所有字节输入流的超类，里面定义了所有字节输入流都必须具备的读取字节的方法
  - int read()

    读取一个字节，以int形式返回，该int值的”低八位”有效，若返回值为-1则表示EOF

  - int read(byte[] data)

    尝试最多读取给定数组的length个字节并存入该数组，返回值为实际读取到的字节量。
- **java.io.OutputStream**是所有字节输出流的超类，里面定义了所有字节输出流都必须具备的写出字节的方法
  - void write(int d)

    写出一个字节,写的是给定的int的”低八位”

  - void write(byte[] data)

    将给定的字节数组中的所有字节全部写出

  - void write(byte[] data,int offset,int len)

    将给定的字节数组中从offset处开始的连续len个字节写出

文件流

概念

文件流是用来链接我们的程序与文件之间的"管道",用来读写文件数据的流。

文件流分为

- 文件输入流java.io.FileInputStream:读取文件数据的流
- 文件输出流java.io.FileOutputStream:写入文件数据的流
- 文件流是继承自InputStream和OutputStream

文件输出流

java.io.FileOutputStream使用文件输出流向文件中写入数据

构造器

```
FileOutputStream(String path)
创建文件输出流对指定的path路径表示的文件进行写操作，如果该文件不存在则将其创建出来

FileOutputStream(File file)
创建文件输出流对指定的file对象表示的文件进行写操作，如果该文件不存在则将其创建出来
```

例

```
package io;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * JAVA IO
 *
 */
public class FOSDemo {
    public static void main(String[] args) throws IOException {
        /*
            FileOutputStream(String path)
            FileOutputStream(File file)
         */
        //向当前项目目录下的文件fos.dat中写入数据
        FileOutputStream fos = new FileOutputStream("./fos.dat");
//        File file = new File("./fos.dat");
//        FileOutputStream fos = new FileOutputStream(file);
        /*
            void write(int d)
            用来向文件中写入1个字节

            计算机底层只有2进制。1和0

                          8421
          0000 0          0001    1
          0001 1          0010    2
```

```
          0010 2          0100    4
          0011 3          1000    8
          0100 4
          0101 5          1110    14
          0110 6
          0111 7         01111111  +127       1字节  1byte
          1000 8         10000000  -
          1001 9
          1010 10
          1011 11
          1100 12
          1101 13
          1110 14
          1111 15


          write方法会将给定的int值对应的2进制的"低八位"写出
          fos.write(1)

          int型1的2进制:
          00000000 00000000 00000000 00000001
                                     ^^^^^^^^
                                     写出的数据
          fos.dat文件中内容:
          00000001

          write(2)
          2的2进制
          00000000 00000000 00000000 00000010
                                     ^^^^^^^^
                                     写出的数据

          fos.dat文件中内容:
          00000001 00000010
          */
        fos.write(1);
        fos.write(2);

        //当IO操作完毕后要关闭
        fos.close();


    }
}
```

总结:

File类

File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

- 1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等
- 2:创建和删除文件或目录
- 3:访问一个目录中的子项

常用构造器:

- File(String pathname)
- File(File parent,String name)可参考文档了解

常用方法:

- length()：返回一个long值，表示占用的磁盘空间，单位为字节。
- canRead()：File表示的文件或目录是否可读
- canWrite()：File表示的文件或目录是否可写
- isHidden()：File表示的文件或目录是否为隐藏的
- createNewFile()：创建一个新文件，如果指定的文件所在的目录不存在会抛出异常 java.io.FileNotFoundException
- mkdir：创建一个目录
- mkdirs：创建一个目录，并且会将所有不存在的父目录一同创建出来，推荐使用。
- delete()：删除当前文件或目录，如果目录不是空的则删除失败。
- exists()：判断File表示的文件或目录是否真实存在。true:存在 false:不存在
- isFile()：判断当前File表示的是否为一个文件。
- isDirectory()：判断当前File表示的是否为一个目录
- listFiles()：获取File表示的目录中的所有子项
- listFiles(FileFilter filter)：获取File表示的目录中满足filter过滤器要求的所有子项