今日目标

能够说出File对象的创建方式

能够说出File类获取名称的方法名称

能够说出File类获取绝对路径的方法名称

能够说出File类获取文件大小的方法名称

能够说出File类判断是否是文件的方法名称

能够说出File类判断是否是文件夹的方法名称

能够辨别相对路径和绝对路径

能够遍历文件夹

能够解释递归的含义

能够使用递归的方式计算5的阶乘

能够说出File类获取绝对路径的方法名称

能够说出使用递归会内存溢出隐患的原因

# File

## ## File类简介

回顾之前学习的字符流作用(FileReader/FileWriter)

将程序中的数据保存到文件中

将文件中的数据读取到程序中

**注意:不能往文件夹读写内容**

File类的作用

File通过路径来表示一个文件或文件夹,File类可以用来新建文件或文件夹,删除文件或文件夹,修改文件或文件夹,获取文件和文件夹的相关信息

## ## File类的静态成员(了解)

1.static String pathSeparator

路径分割符 在windows中是 ; 在Linux中是 :

2.static String separator

一个路径中文件和文件夹之间的分隔符 windows中 \ 在Linux中 /

我们在开发中直接是用 \ 拼接字符串即可, 在工作中建议使用separator.方便部署到Linux中

"C:" + File.separator + "MyFileTest"

Windows -> "C:\\MyFileTest"

Linux -> "C:/MyFileTest"

代码

public class Demo02 {

public static void main(String[] args) {

// 多个路径之间的分割符path环境变量中

// 路径分割符 在windows中是 ; 在Linux中是 :

System.*out*.println(File.*pathSeparator*);

// 一个路径中文件和文件夹之间的分隔符 windows中 \ 在Linux中 /

System.*out*.println(File.*separator*);

/\*

在开发中经常要写路径: "C:\Users"

在公司里面:建议使用 File.separator,

\*/

String path = "C:" + File.*separator* + "Users";

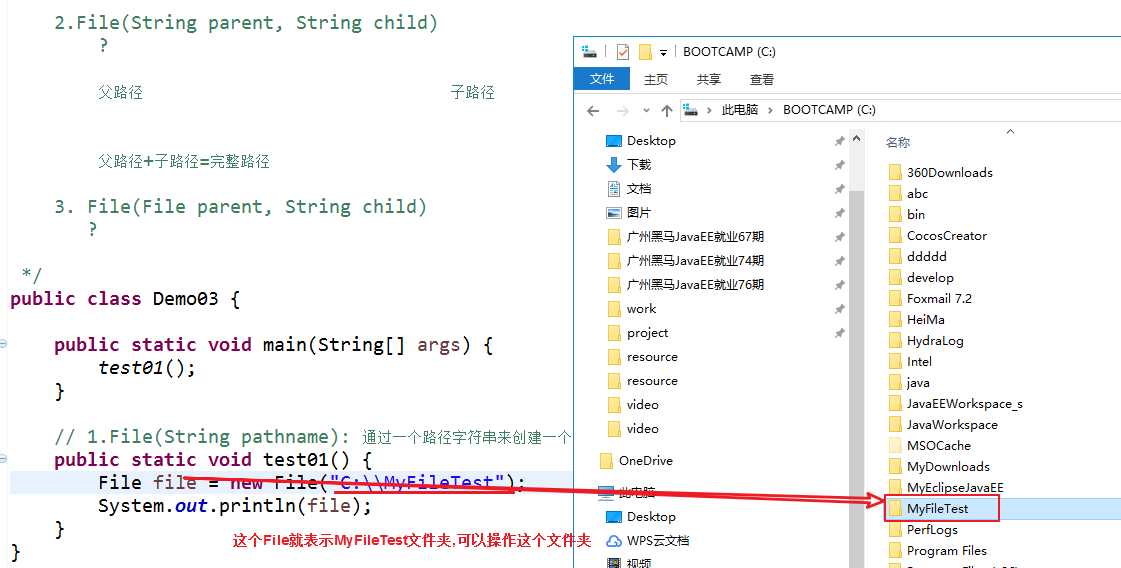
// Windows: "C:" + \ + "Users";

// Linux: "C:" + / + "Users";

}

}

## ## File构造方法



1.File(String pathname)

通过一个路径字符串来创建一个File对象

2.File(String parent, String child)

通过父路径和子路径来创建File对象

C:\MyFileTest\高清无码\苍老师\美女1.jpg

父路径 子路径

C:\MyFileTest\高清无码\苍老师 美女1.jpg

C:\MyFileTest\高清无码 苍老师\美女1.jpg

C:\MyFileTest 高清无码\苍老师\美女1.jpg

父路径+子路径=完整路径

3. File(File parent, String child)

通过父路径File对象和子路径字符串来创建File对象

注意

file表示的文件/文件夹不一定存在.我们通过方法来取操作这个路径

代码

public class Demo03 {

public static void main(String[] args) {

// test01();

// test02();

*test03*();

}

// 3. File(File parent, String child): 通过父路径File对象和子路径字符串来创建File对象

public static void test03() {

File parent = new File("C:\\MyFileTest\\高清无码");

File file = new File(parent, "苍老师\\美女1.jpg");

System.*out*.println(file);

}

// 2.File(String parent, String child): 通过父路径和子路径来创建File对象

public static void test02() {

// File file = new File("C:\\MyFileTest\\高清无码\\苍老师 ", "美女1.jpg");

File file = new File("C:\\MyFileTest", "高清无码\\苍老师\\美女1.jpg");

System.*out*.println(file);

}

// 1.File(String pathname): 通过一个路径字符串来创建一个File对象

public static void test01() {

// file表示的文件/文件夹不一定存在.我们通过方法来取操作这个路径

File file = new File("C:\\MyFileTest12344");

System.*out*.println(file);

}

}

## ## File功能介绍

1.创建功能

2.删除功能

3.获取功能

4.判断功能

5.list功能 (重要)

## ## File创建功能

文件名规律

C:\MyFileTest\ddd\cc.txt

最后一个\右边的作为文件名

最后一个\左边的作为文件夹

1.创建文件

boolean createNewFile()

2.创建文件夹

boolean mkdir()

\*\*\*\*\*boolean mkdirs(): 创建文件夹,父路径不存在也会帮我们创建父路径

注意大小写问题

**windows中文件名不区分大小写**

**到底创建的是文件还是文件夹,不能看文件名,而要看调用的方法**

代码

public class Demo04 {

public static void main(String[] args) throws IOException {

// method01();

// method02();

// method03();

*method04*();

}

private static void method04() throws IOException {

File file = new File("C:\\MyFileTest\\新建文件夹");

boolean b = file.createNewFile();

System.*out*.println(b);

}

// boolean mkdirs(): 创建文件夹,父路径不存在也会帮我们创建父路径

private static void method03() throws IOException {

File file = new File("C:\\MyFileTest\\ccc\\ddd\\Aaa");

boolean b = file.mkdirs(); // make directory

System.*out*.println(b);

}

// 2.创建文件夹: boolean mkdir()

private static void method02() throws IOException {

// 父路径不存在,创建不成功

// File file = new File("C:\\MyFileTest\\ccc\\ddd\\aaa");

File file = new File("C:\\MyFileTest\\标清有码");

boolean b = file.mkdir(); // make directory

System.*out*.println(b);

}

// 1.创建文件: boolean createNewFile()

private static void method01() throws IOException {

File file = new File("C:\\MyFileTest\\abc.txt");

boolean b = file.createNewFile();

System.*out*.println(b);

}

}

## ## File删除功能

删除文件和文件夹

boolean delete()

注意

文件夹中有内容不能直接删除,先要删除文件夹里面的所有内容

代码

public class Demo05 {

public static void main(String[] args) {

// method01();

// method02();

*method03*();

}

// 删除文件夹: 文件夹中有内容不能直接删除,先要删除文件夹里面的所有内容

private static void method03() {

// 先要删除文件夹里面的所有内容

File f = new File("C:\\MyFileTest\\ccc\\123.txt");

f.delete();

File file = new File("C:\\MyFileTest\\ccc"); // 有内容的文件夹

boolean b = file.delete();

System.*out*.println(b);

}

// 删除文件夹

private static void method02() {

File file = new File("C:\\MyFileTest\\标清有码"); // 空文件夹

boolean b = file.delete();

System.*out*.println(b);

}

// 删除文件

private static void method01() {

File file = new File("C:\\MyFileTest\\abc.txt"); // 文件

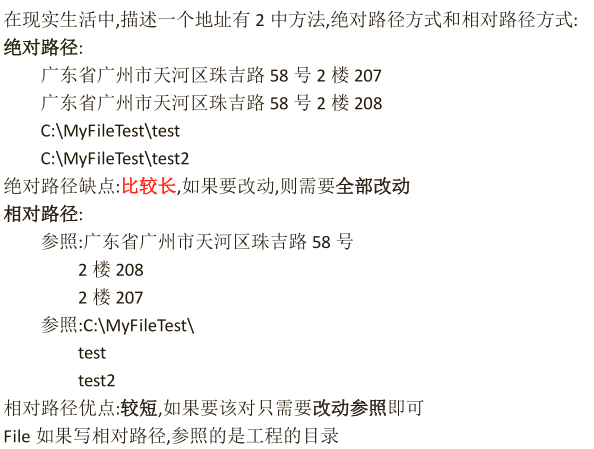
boolean b = file.delete();

System.*out*.println(b);

}

}

## ## 回顾相对路径和绝对路径



## ## File获取功能

1.String getPath(): 获取File对象的路径

\*\*\*\*\* 2.String getName(): 获取名称,最后一个\右边的

\*\*\*\*\* 3.String getParent(): 获取到父路径最后一个\左边的

4.File getParentFile(): 获取到父路径最后一个\左边的,以File的形式返回

\*\*\*\*\* 5.String getAbsolutePath(): 获取绝对路径字符串

6.File getAbsoluteFile(): 获取绝对路径File对象

\*\*\*\*\* 7. long length(): 获取文件大小

注意

length只能获取文件,文件夹是不准的

代码

public class Demo06 {

public static void main(String[] args) {

// test01();

// test02();

// test03();

// test04();

// test05();

// test06();

*test07*();

}

// \*\*\*\*\* 7. long length(): 获取文件大小

public static void test07() {

// File file = new File("C:\\MyFileTest\\abc\\b.jpg");

File file = new File("C:\\MyFileTest\\abc");

long length = file.length(); // 单位是字节byte

System.*out*.println(length);

}

// 6.File getAbsoluteFile(): 获取绝对路径File对象

public static void test06() {

File file = new File("a.txt"); // 不是从盘符开始的,叫相对路径

File absoluteFile = file.getAbsoluteFile();

System.*out*.println(absoluteFile); // C:\Users\zhangping\work\广州黑马JavaEE就业76期\day10\code\day10-File-递归\a.txt

}

// \*\*\*\*\* 5.String getAbsolutePath(): 获取绝对路径字符串

public static void test05() {

File file = new File("a.txt"); // 不是从盘符开始的,叫相对路径

// 获取绝对路径字符串

String absolutePath = file.getAbsolutePath();

System.*out*.println(absolutePath); // C:\Users\zhangping\work\广州黑马JavaEE就业76期\day10\code\day10-File-递归\a.txt

}

// 4.File getParentFile(): 获取到父路径最后一个\左边的,以File的形式返回

public static void test04() {

File file = new File("C:\\MyFileTest\\高清无码\\苍老师\\ChangLiang.java");

File parentFile = file.getParentFile(); // C:\MyFileTest\高清无码\苍老师

System.*out*.println(parentFile);

}

// \*\*\*\*\* 3.String getParent(): 获取到父路径最后一个\左边的

public static void test03() {

File file = new File("C:\\MyFileTest\\高清无码\\苍老师\\ChangLiang.java");

String parent = file.getParent(); // C:\MyFileTest\高清无码\苍老师

System.*out*.println(parent);

}

// \*\*\*\*\* 2.String getName(): 获取名称,最后一个\右边的

public static void test02() {

File f = new File("C:\\MyFileTest\\高清无码\\苍老师\\ChangLiang.java");

String name = f.getName(); // ChangLiang.java

System.*out*.println(name);

}

// 1.String getPath(): 获取File对象的路径

public static void test01() {

File f = new File("C:\\MyFileTest\\高清无码\\小泉老师");

String path = f.getPath();

System.*out*.println(path);

}

}

## ## File判断功能

**\*\*\*\*\*** 1. boolean exists(): 判断对应的文件或文件夹是否存在

**\*\*\*\*\*** 2. boolean isDirectory(): 判断是否是文件夹

**\*\*\*\*\*** 3. boolean isFile(): 判断是否是文件

代码

public class Demo07 {

public static void main(String[] args) {

// test01();

// test02();

*test03*();

}

// \*\*\*\*\* 3. boolean isFile(): 判断是否是文件, 存在的文件才返回true

public static void test03() {

// File file = new File("C:\\MyFileTest\\b.jpg"); // 存在的文件

// File file = new File("C:\\MyFileTest\\高清无码"); // 存在的文件夹

File file = new File("C:\\MyFileTest\\高清无码1"); // 不存在

boolean b = file.isFile();

System.*out*.println(b);

}

// \*\*\*\*\* 2. boolean isDirectory(): 判断是否是文件夹, 存在的文件夹才返回true

public static void test02() {

// File file = new File("C:\\MyFileTest\\高清无码"); // 存在的文件夹

// File file = new File("C:\\MyFileTest\\b.jpg"); // 存在的文件

File file = new File("C:\\MyFileTest\\高清无码1"); // 不存在

boolean b = file.isDirectory();

System.*out*.println(b);

}

// \*\*\*\*\* 1. boolean exists(): 判断对应的文件或文件夹是否存在

public static void test01() {

// File file = new File("C:\\MyFileTest\\高清无码"); // 存在的文件夹

File file = new File("C:\\MyFileTest\\b.jpg"); // 存在的文件

// File file = new File("C:\\MyFileTest\\高清无码1"); // 不存在

boolean b = file.exists();

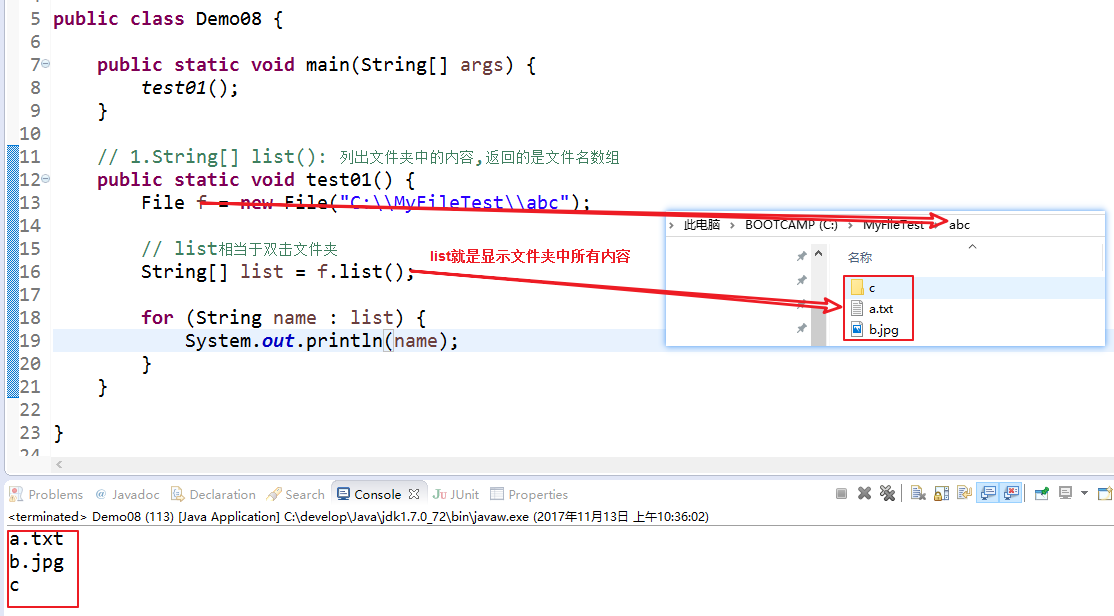
System.*out*.println(b);

}

}

## ## File的list功能(重要)

list功能列出文件夹中的内容.**相当于双击文件夹**



1.String[] list()

列出文件夹中的内容,返回的是文件名数组

2.File[] listFiles()

列出文件夹中的内容,返回的是File对象数组



注意

列出文件夹里面的内容.如果内容里面是文件夹.不会帮我们列出来

## ## 列出符合需求的文件

需求:列出指定目录里面的.java文件

分析

1. 定义一个File关联指定文件夹
2. 使用list/listFiles功能列出所有的内容
3. 遍历所有内容,拿到每个File对象
4. 判断如果是文件夹不用管
5. 如果是文件,拿到文件名
6. 判断文件名的后缀是否是.java结尾

代码

public class Demo09 {

public static void main(String[] args) {

// 1. 定义一个File关联指定文件夹

File file = new File("C:\\MyFileTest\\cba");

// 2. 使用list/listFiles功能列出所有的内容

File[] listFiles = file.listFiles();

// 3. 遍历所有内容,拿到每个File对象

for (File f : listFiles) {

// 4. 判断如果是文件夹不用管

// 5. 如果是文件,拿到文件名

if (f.isFile()) {

String fileName = f.getName();

// 6. 判断文件名的后缀是否是.java结尾

if (fileName.endsWith(".java")) {

System.*out*.println(fileName);

}

}

}

}

}

## ## 文件过滤器(难点.重点)

需求:列出指定目录里面的.java文件

以前listFiles()直接将所有的内容都返回,然后自己for循环根据条件拿到符合要求的

文件过滤器方法

File[] listFiles(FileFilter filter):直接返回满足FileFilter的内容

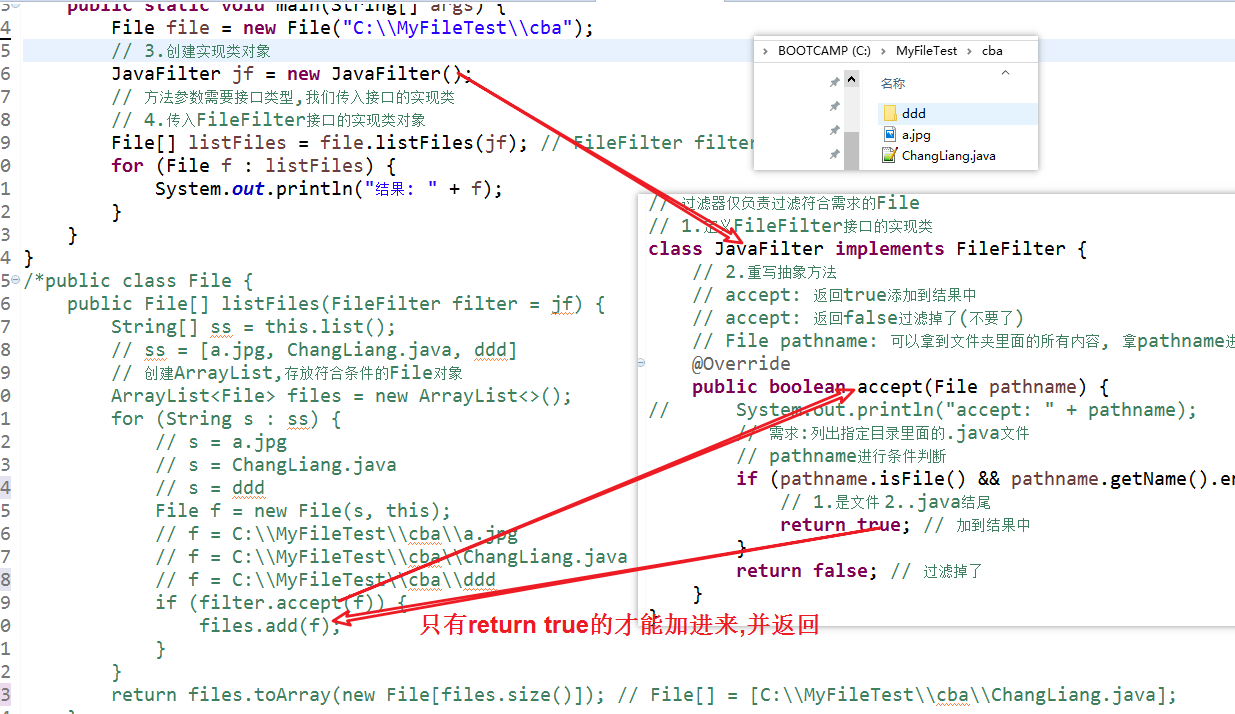
文件过滤器的使用步骤

1. 定义FileFilter接口的实现类
2. 重写抽象方法,满足条件的返回true
3. 创建实现类对象
4. 传入FileFilter接口的实现类对象
5. listFiles返回的结果就是符合条件的File

文件过滤器的作用

只返回符合要求的内容

过滤器源码分析



代码

public class Demo10 {

public static void main(String[] args) {

File file = new File("C:\\MyFileTest\\cba");

// 3.创建实现类对象

JavaFilter jf = new JavaFilter();

// 方法参数需要接口类型,我们传入接口的实现类

// 4.传入FileFilter接口的实现类对象

File[] listFiles = file.listFiles(jf); // FileFilter filter = new JavaFilter();

for (File f : listFiles) {

System.*out*.println("结果: " + f);

}

}

}

/\*public class File {

public File[] listFiles(FileFilter filter = jf) {

String[] ss = this.list();

// ss = [a.jpg, ChangLiang.java, ddd]

// 创建ArrayList,存放符合条件的File对象

ArrayList<File> files = new ArrayList<>();

for (String s : ss) {

// s = a.jpg

// s = ChangLiang.java

// s = ddd

File f = new File(s, this);

// f = C:\\MyFileTest\\cba\\a.jpg

// f = C:\\MyFileTest\\cba\\ChangLiang.java // 满足条件,返回true

// f = C:\\MyFileTest\\cba\\ddd

if (filter.accept(f)) {

files.add(f);

}

}

return files.toArray(new File[files.size()]); // File[] = [C:\\MyFileTest\\cba\\ChangLiang.java];

}

}

\*/

// 过滤器仅负责过滤符合需求的File

// 1.定义FileFilter接口的实现类

class JavaFilter implements FileFilter {

// 2.重写抽象方法

// accept: 返回true添加到结果中

// accept: 返回false过滤掉了(不要了)

// File pathname: 可以拿到文件夹里面的所有内容, 拿pathname进行条件判断

@Override

public boolean accept(File pathname) {

// System.out.println("accept: " + pathname);

// 需求:列出指定目录里面的.java文件

// pathname进行条件判断

if (pathname.isFile() && pathname.getName().endsWith(".java")) {

// 1.是文件 2..java结尾

return true; // 加到结果中

}

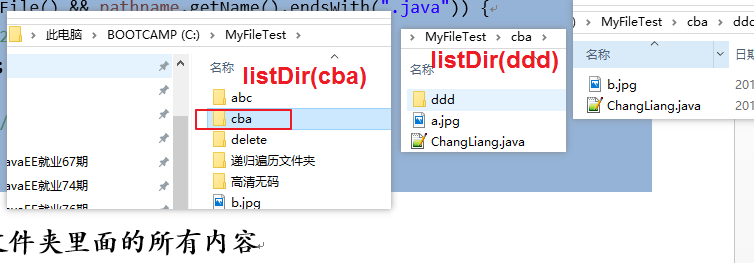
return false; // 过滤掉了

}

}

## ## 递归遍历文件夹里面的所有内容

需求: 遍历文件夹里面的所有内容,如果文件夹里面有文件夹,也需要将这个文件夹里面的内容遍历出来



分析

1. 定义一个File关联到对应的文件夹
2. 定义一个方法listDir(File file),列出对应file里面的内容
3. 调用listFiles功能,列出所有内容
4. 遍历每个File对象
5. 如果是文件,直接输出
6. 如果是文件夹,调用listDir(ddd)

代码

/\*

需求: 遍历文件夹里面的所有内容,如果文件夹里面有文件夹,也需要将这个文件夹里面的内容遍历出来

分析

1. 定义一个File关联到对应的文件夹

2. 定义一个方法listDir(File file),列出对应file里面的内容

3. 调用listFiles功能,列出所有内容

4. 遍历每个File对象

5. 如果是文件,直接输出

6. 如果是文件夹,调用listDir(ddd)

\*/

**public** **class** Demo11 {

**public** **static** **void** main(String[] args) {

// 1. 定义一个File关联到对应的文件夹

File file = **new** File("C:\\MyFileTest\\cba");

*listDir*(file);

}

// 2. 定义一个方法listDir(File file),列出对应file里面的内容

**public** **static** **void** listDir(File file) {

// 3. 调用listFiles功能,列出所有内容

File[] listFiles = file.listFiles();

// 4. 遍历每个File对象

**for** (File f : listFiles) {

// 5. 如果是文件,直接输出

**if** (f.isFile()) {

System.***out***.println(f);

} **else** {

// 6. 如果是文件夹,调用listDir(ddd)

*listDir*(f);

}

}

}

}

## ## 递归的概述

递归分为两种

1.直接递归

方法自己调用自己 StackOverflowError: 栈溢出

2.间接递归

A() -> B() -> C() -> A()

递归的注意事项

1. 递归次数不能太多
2. 递归需要出口
3. 构造方法不能递归

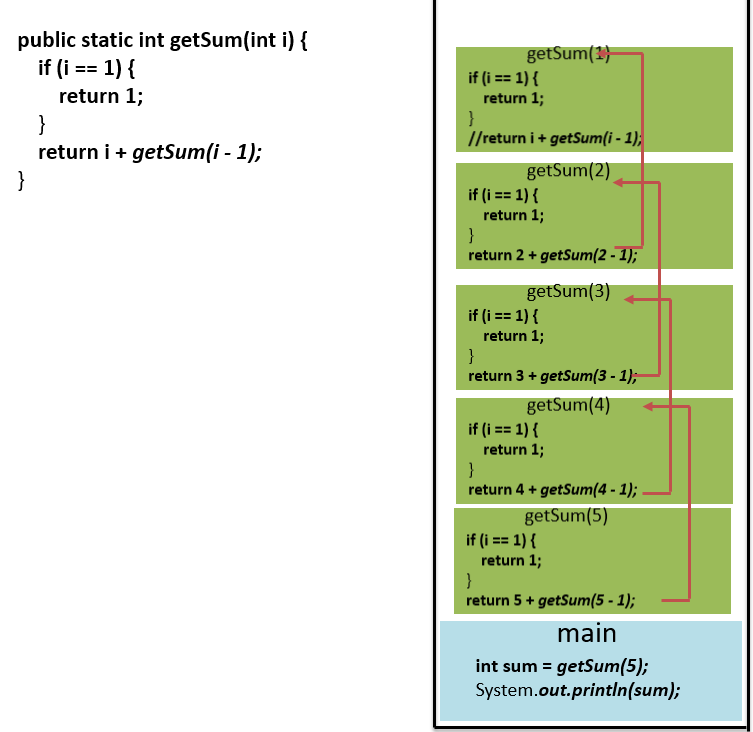
## ## 递归求1 -5的和

5+ 1-4的和

5+4 +1-3的和

5+4+3 +1-2的和

5+4+3+2 + 1-1的和



代码

public class Demo13 {

public static void main(String[] args) {

// System.out.println(getSum1());

System.*out*.println(*getSum*(100));

}

// 递归求1 -5的和: 递归是自己调用自己,参数不一样

// 1.分析递归的规律, 2.根据规律写代码

// 规律: 1到n的和 = n + (1到n-1的和)

// 这个方法就是求1到n的和

public static int getSum(int n) {

if (n == 1) {

return 1;

}

// 1到n的和 = n + (1到n-1的和)

return n + *getSum*(n - 1);

}

public static int getSum1() {

int sum = 0;

for (int i = 1; i <= 5; i++) {

sum += i;

}

return sum;

}

}

能够说出使用递归会内存溢出隐患的原因

递归是方法的不停进栈,方法进入栈中是需要消耗栈内存的.递归次数多了.栈空间就不够了.就溢出了

## ## 递归求5! (1\*2\*3\*4\*5)

!阶乘

## ## 递归实现斐波那契数列

不死神兔

汉诺塔问题

需求:

有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，

假如兔子都不死，问第二十个月的兔子对数为多少？

分析:找到规律(+表示新生的兔子)

总对数 详情

第一个月 1 1

第二个月 1 1

第三个月 2 1 +1

第四个月 3 1 1 +1

第五个月 5 1 1 1 +1 +1

第六个月 8 1 1 1 1 1 +1 +1 +1

规律

第一个月和第二个月都是1对

从第三个月开始,这个月的兔子对数 = 上个月的对数 + 上上个月的对数

代码

public class Demo14 {

public static void main(String[] args) {

int fb = *fb*(20);

System.*out*.println(fb);

}

/\*

有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，

假如兔子都不死，问第二十个月的兔子对数为多少？

第一个月和第二个月都是1对

从第三个月开始,这个月的兔子对数 = 上个月的对数 + 上上个月的对数

\*/

// 定义一个方法求某个月的兔子对数

public static int fb(int n) {

if (n == 1 || n == 2) {

return 1;

}

// 从第三个月开始,这个月的兔子对数 = 上个月的对数 + 上上个月的对数

return *fb*(n - 1) + *fb*(n - 2);

}

}

## ## 递归好不好

不好

1. 递归的规律不好找
2. 根据递归的规律写代码不好写
3. 递归是方法不停的进栈,容易导致栈溢出,速度不够快
4. 能用其他替换的尽量使用其他替换
5. 但是有些地方无法替换.没办法