什么是递归

递归：简单来说就是自己调用自己

如：

main(){

a();

}

a(){

a();

}

-递归的分类：

-递归可以分为两种，直接递归和间接递归

-直接递归：方法自己调用自己；

-简介递归：方法间接调用自己；

如：A方法调用B方法，B方法再调用A方法；

a(){

b();

}

b(){

a();

}

-注意事项：

-递归一定要有条件判定（即递归出口），保证递归可以停止下来，否则会发生栈溢出；

-在递归中虽然有出口，但递归次数不能太多，否则也会发生栈溢出；

-构造方法禁止递归。

-递归的使用前提：

-当调用方法的时候，方法的主体不变，每次调用的参数也不同，可以使用递归

-代码示例：

public class Main {

public static void main(String[] args){

// a();

b();

}

// 3.构造方法,禁止递归

// 编译报错:构造方法是创建对象使用的,不能让对象一直创建下去

//

// public Demo01DiGui()

// {

// //Demo01DiGui();

// }

//2.在递归中虽然有限定条件，但是递归次数不能太多。否则也会发生栈内存溢出。

// 4993

// Exception in thread "main" java.lang.StackOverflowError

private static void b(int i){

System.out.println(i);

//添加一个递归结束的条件,i==5000的时候结束

if(i==5000){

return;//结束方法

}

b(++i);

}

//1.递归一定要有条件限定，保证递归能够停止下来，否则会发生栈内存溢出。Exception in thread "main"

//java.lang.StackOverflowError

private static void a(){

System.out.println("a方法");

a();

}

}

对于递归问题的思考方式：

在初学递归的时候, 看到一个递归实现, 我们总是难免陷入不停的验证之中，比如上面提及的阶乘，求解Factorial(n)时，我们总会情不自禁的发问，Factorial(n-1)可以求出正确的答案么？接着我们就会再用Factorial(n-2)去验证，，，不停地往下验证直到Factorial(0)。

对递归这样的不适应，和我们平时习惯的思维方式有关。我们习惯的思维是：已知Factorial(0)，乘上 1 就等于Factorial(1)，再乘以 2 就等于Factorial(2)，，，直到乘到 n。

而递归和我们的思维方式正好相反。

那我们怎么判断这个递归计算是否是正确的呢？Paul Graham 提到一种方法，如下：

如果下面这两点是成立的，我们就知道这个递归对于所有的 n 都是正确的。

当 n=0,1 时，结果正确；

假设递归对于 n 是正确的，同时对于 n+1 也正确。

这种方法很像数学归纳法，也是递归正确的思考方式，上述的第 1 点称为基本情况，第 2 点称为通用情况。

在递归中，我们通常把第 1 点称为终止条件，因为这样更容易理解，其作用就是终止递归，防止递归无限地运行下去。

递归示例：

计算1~n的和

思路分析：n的累和其实可以看作是一个数列，该数列的递推公式为：

N(n) = n + N(n-1);

当n = 1 时为递归出口所以可以将累和的操作定义成一个方法，递归调用。

代码如下：

public static void main(String[] args) {

//计算1~num的和，使用递归完成

int num = 5;

// 调用求和的方法

int sum = getSum(num);

// 输出结果

System.out.println(sum);

}

/\*

通过递归算法实现.

参数列表:int

返回值类型: int

\*/

public static int getSum(int num) {

/\*

num为1时,方法返回1,

相当于是方法的出口,num总有是1的情况

\*/

if (num == 1) {

return 1;

}

/\*

num不为1时,方法返回 num +(num‐1)的累和

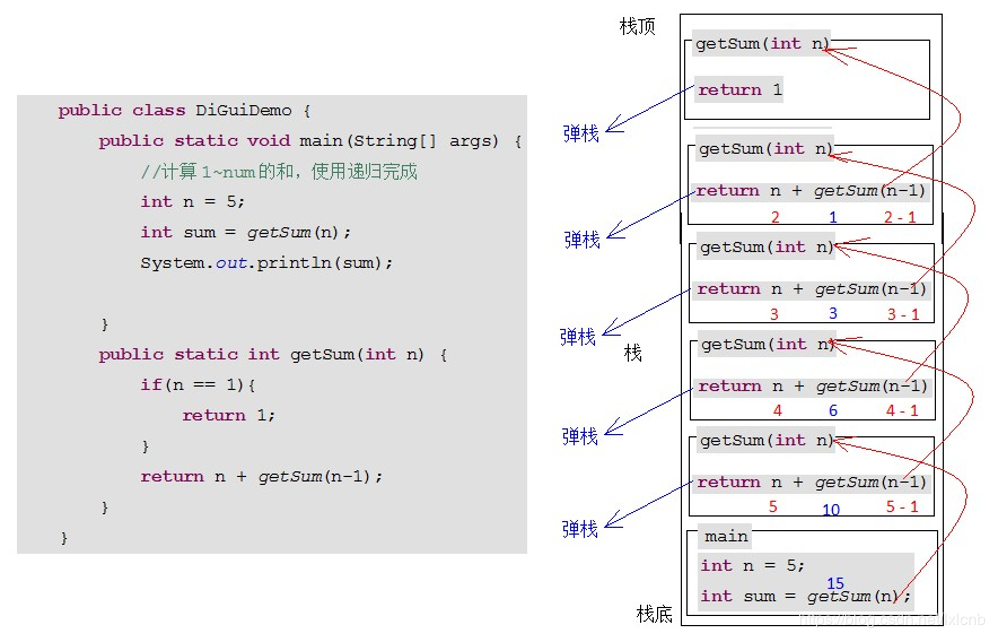
递归调用getSum方法

\*/

return num + getSum(num-1);

}

代码执行图解：



什么时候该用递归

当我们遇到一个问题时，我们是怎么判断该题用递归来解决的？

问题可用递归来解决需具备的条件：

子问题需与原问题为同样的事，且规模更小；

程序停止条件。

————————————————

版权声明：本文为CSDN博主「lxlcnb」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/lxlcnb/article/details/102519781