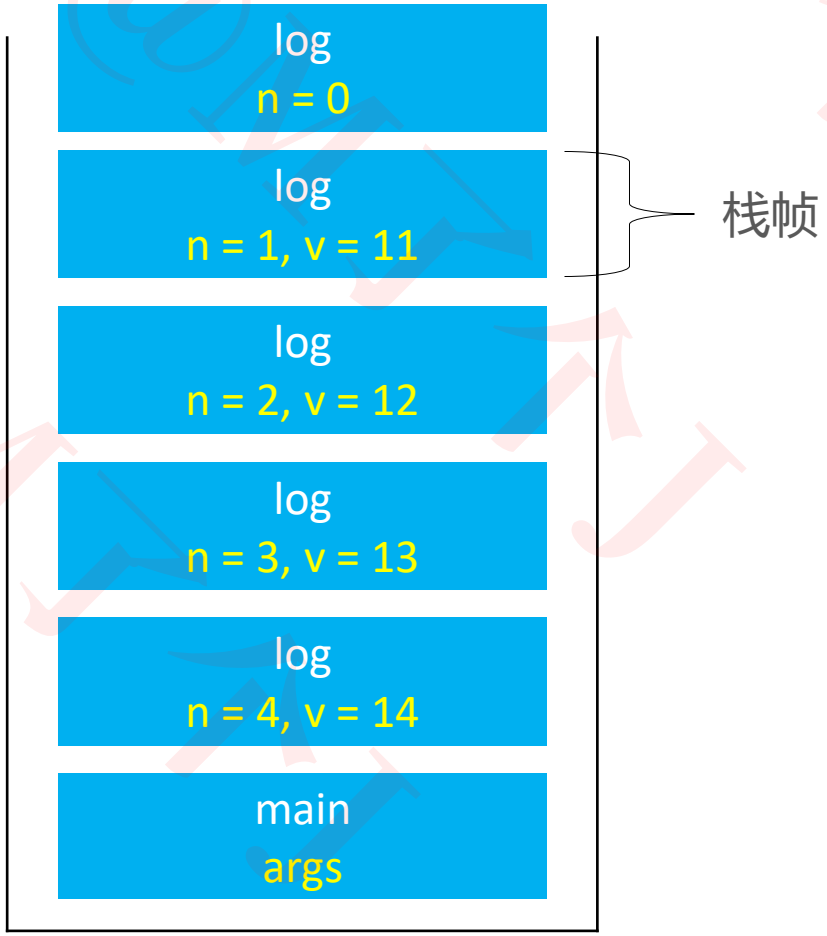
递归转非递归

- 递归调用的过程中，会将每一次调用的参数、局部变量都保存在了对应的栈帧（Stack Frame）中

```java
public static void main(String[] args) {
    log(4);
}

static void log(int n) {
    if (n < 1) return;
    log(n - 1);
    int v = n + 10;
    System.out.println(v);
}
```



- 若递归调用深度较大，会占用比较多的栈空间，甚至会导致栈溢出
- 在有些时候，递归会存在大量的重复计算，性能非常差
- 这时可以考虑将递归转为非递归（递归100%可以转换成非递归）

小码哥教育 SEEMYGO

递归转非递归

- 递归转非递归的万能方法
  - 自己维护一个栈，来保存参数、局部变量
  - 但是空间复杂度依然没有得到优化

```
static class Frame {
    int n;
    int v;
    Frame(int n, int v) {
        this.n = n;
        this.v = v;
    }
}
```

```
static void log(int n) {
    Stack<Frame> frames = new Stack<>();
    while (n > 0) {
        frames.push(new Frame(n, n + 10));
        n--;
    }
    while (!frames.isEmpty()) {
        Frame frame = frames.pop();
        System.out.println(frame.v);
    }
}
```

递归转非递归

- 在某些时候，也可以重复使用一组相同的变量来保存每个栈帧的内容

```
static void log(int n) {
    for (int i = 1; i <= n; i++) {
        System.out.println(i + 10);
    }
}
```

- 这里重复使用变量 i 保存原来栈帧中的参数
- 空间复杂度从 O(n) 降到了 O(1)