c422分享会

进程的内存模型

天津师范大学
计算机1601
王小多

1. 引言
2. 程序的内存模型（不完整）
3. 函数和栈的关系
4. 程序的内存模型（堆）

1.引言

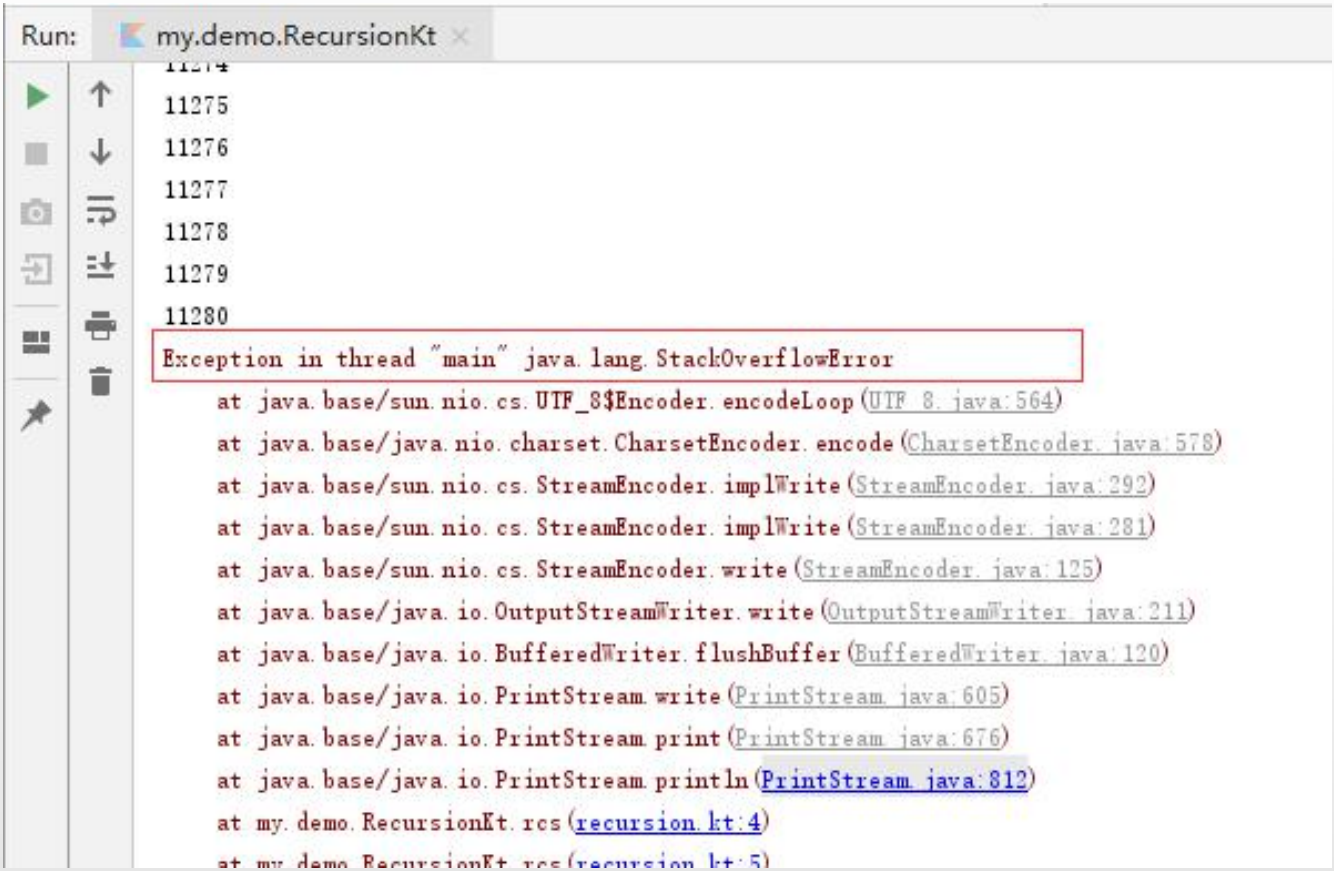
最近玩kotlin，一不小心，写了个臭代码
类似下面这个，运行在java虚拟机上

```
package my.demo

val i =1
fun rcs(i:Int){
    println(i)
    rcs( i: i+1)
}

fun main()
{
    rcs(i)
}
```

kotlin可以运行在宿主机也可以运行在java虚拟机

1.引言

结果如下


Run:
my.demo.RecursionKt
11275
11276
11277
11278
11279
11280
Exception in thread "main" java.lang.StackOverflowError
at java.base/sun.nio.cs.UTF_8$Encoder.encodeLoop(UTF_8.java:564)
at java.base/java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:578)
at java.base/sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:292)
at java.base/sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:281)
at java.base/sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)
at java.base/java.io.OutputStreamWriter.write(OutputStreamWriter.java:211)
at java.base/java.io.BufferedWriter.flushBuffer(BufferedWriter.java:120)
at java.base/java.io.PrintStream.write(PrintStream.java:605)
at java.base/java.io.PrintStream.print(PrintStream.java:676)
at java.base/java.io.PrintStream.println(PrintStream.java:812)
at my.demo.RecursionKt.rcs(recursion.kt:4)

1.引言

问题：

1. 函数递归次数过多，为什么引起栈溢出。

2. 变量和栈的关系与函数和栈的关系有没有什么相似的地方。

3. 函数之间不能相互访问彼此内部的变量，是为什么。

4. 所有函数都可以访问全局变量和全局常量是为什么。

2.进程的内存模型

一个简单的c语言程序在执行时，它在内存空间中被分为两部分：

1. **只读区域**
2. **读写区域**

1. 只读区域用来存储**全局常量**和**函数的执行体**。内容大小在开始执行时就确定下来。

2. 读写区域则是可变的，大小不固定。它的内容随程序的运行而不同。进程初始化时，大小为**全局变量和**程序中**所有静态变量**的大小。

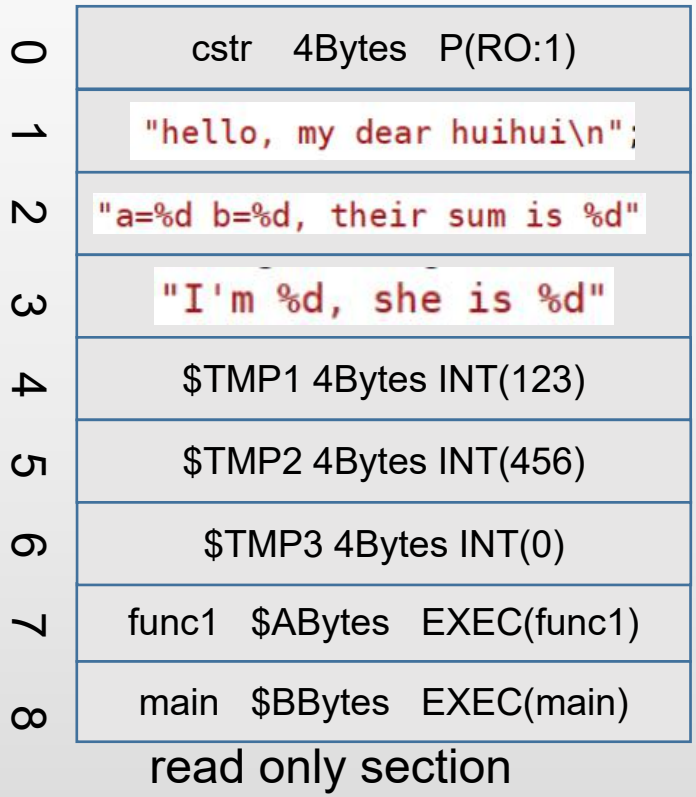
接下来，用一个c语言示例来表示。

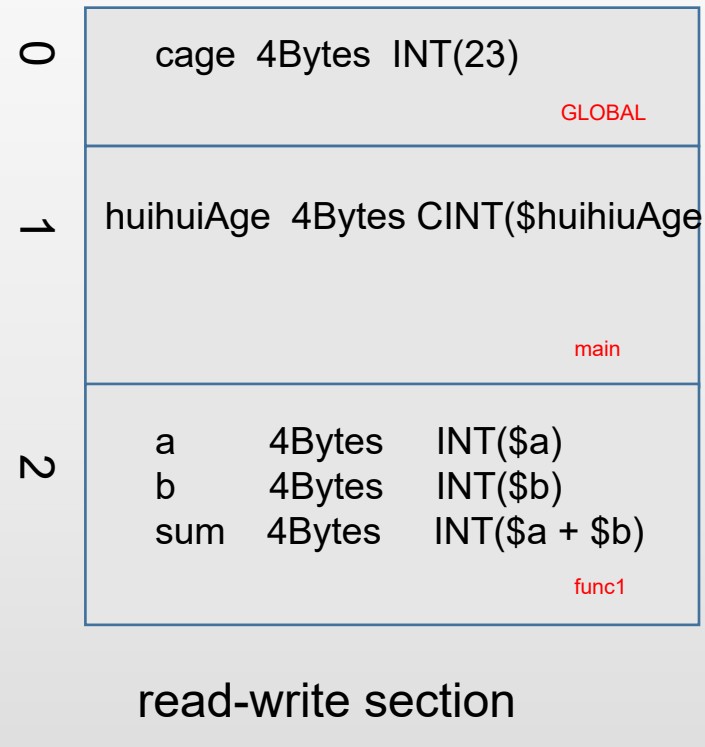
2.进程的内存模型

```c
#include<stdio.h>
const char* cstr = "hello, my dear huihui\n";
int cage = 23;
int func1(int a, int b)
{
    int sum = a + b;
    printf("a=%d b=%d, their sum is %d", a, b, sum);
}
int main(void)
{
    printf(cstr);
    const int huihuiAge = cage - 1;
    printf("I'm %d, she is %d", cage, huihuiAge);

    func1(123, 456);
    return 0;
}
```

下面先进行一个概览

```c
#include<stdio.h>
const char* cstr = "hello, my dear huihui\n";
int cage = 23;
int func1(int a, int b)
{
    int sum = a + b;
    printf("a=%d b=%d, their sum is %d", a, b, sum);
    return sum;
}
int main(void)
{
    printf(cstr);
    const int huihuiAge = cage - 1;
    printf("I'm %d, she is %d", cage, huihuiAge);

    func1(123, 456);
    return 0;
}
```



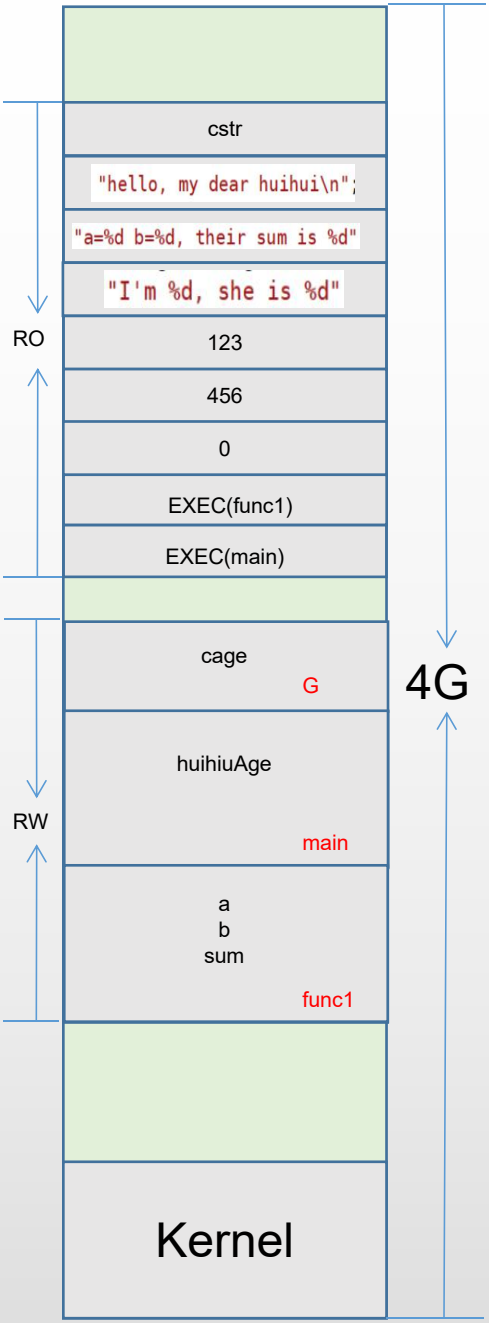


2.进程的内存模型

在实际运行时，进程的内存情况如右图

接下来，放大招。

我们从程序载入内存开始执行，一步一步分析。

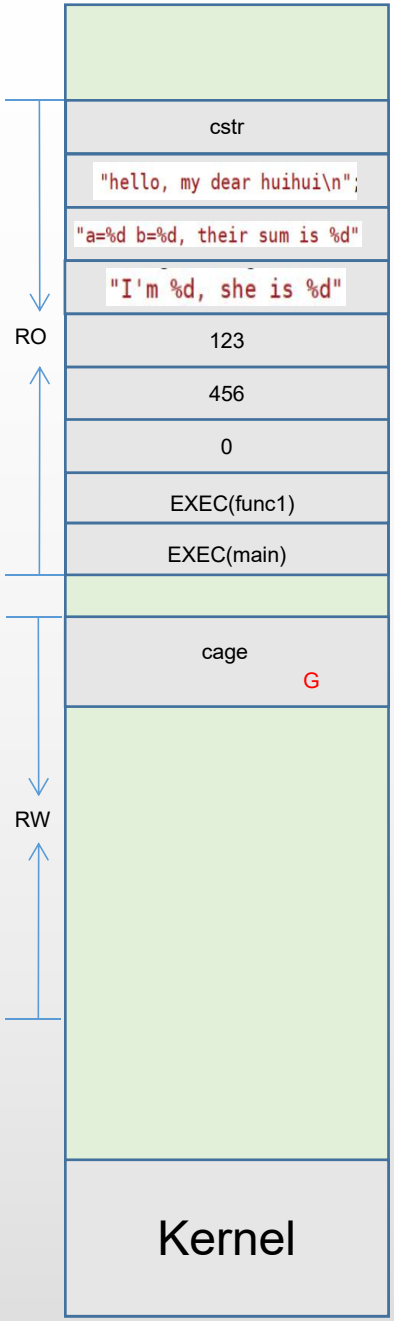


2.进程的内存模型

1.程序刚刚载入内存时，首先将**全局常量**和**函数执行体**放到RO（只读）区域。

将全局变量放到RW（读写）区域。

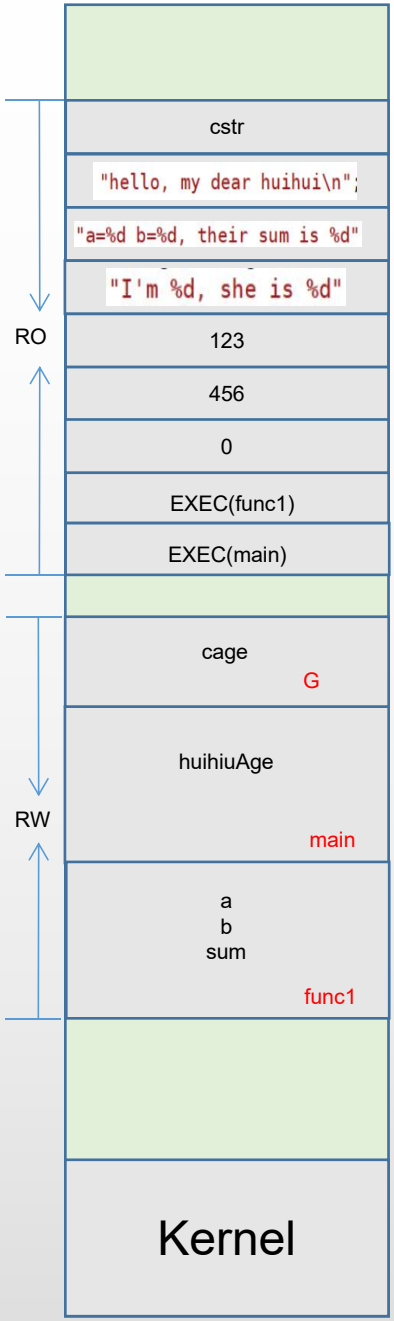
这些动作都在main()执行前完成，属于初始化工作



2.进程的内存模型

2. 开始执行main()，同时将执行过程中声明的**变量**和**常量**存入内存。

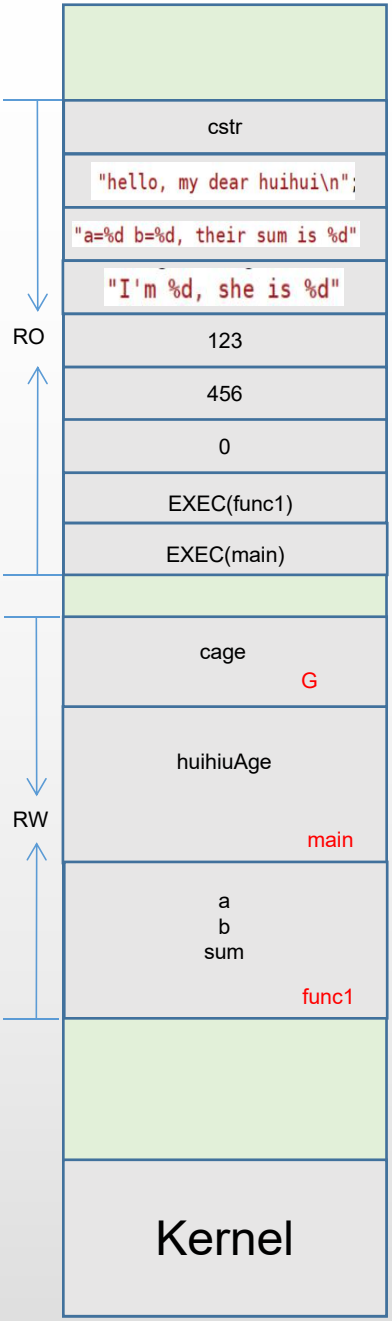
3.main()调用func1()，将func1()执行时声明的变量和常量存入内存。



2.进程的内存模型

4.func1()执行结束，返回mian，清除func1()占用的内存。

5.main()结束，进程结束，所有内存被清空

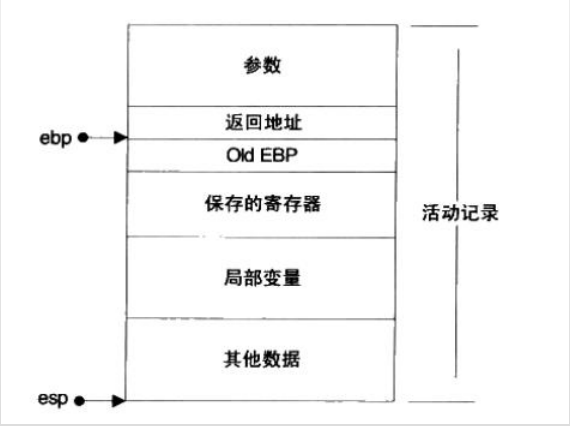


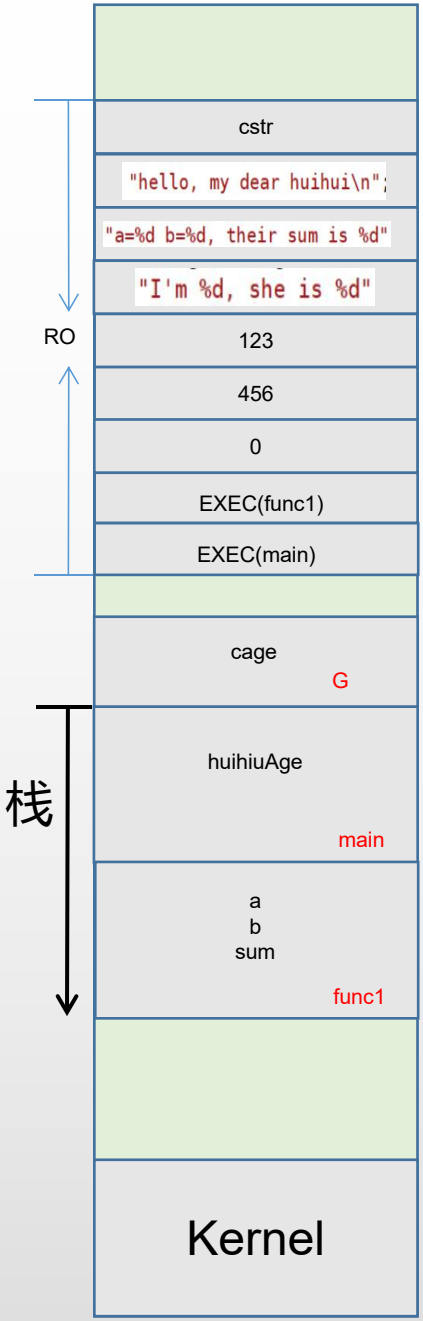
2.进程的内存模型

结合这个过程，我们发现，函数体中的变量和全局变量的进驻内存和从内存中清除的过程，就像是一个入栈、出栈的过程。

自然而然我们就把程序执行时，用来保存函数中变量的这个区域叫做**栈**。

需要说明的是，栈中不仅仅保存函数中的变量，还有调用当前函数时保存的上一函数使用的寄存器、返回地址等等信息。





1.引言

问题：

1. 函数递归次数过多，为什么引起栈溢出。

2. 变量和栈的关系与函数和栈的关系有没有什么相似的地方。

3. 函数之间不能相互访问彼此内部的变量，是为什么。

4. 所有函数都可以访问全局变量和全局常量是为什么。

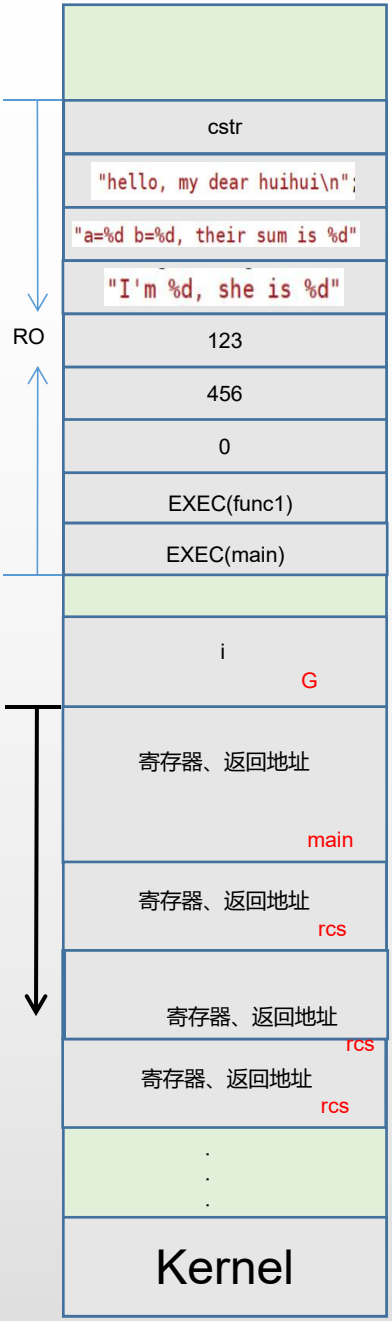
3.函数和栈的关系

结合上一节内容，我们就知道程序使用栈来保存函数中的内容。

那么当我们创建很多函数时，比如这个递归函数

```
package my.demo

val i =1
fun rcs(i:Int){
    println(i)
    rcs( i: i+1)
}

fun main()
{
    rcs(i)
}
```

栈只被使用，而没有释放，慢慢的，程序存储空间就不足了。



1.引言

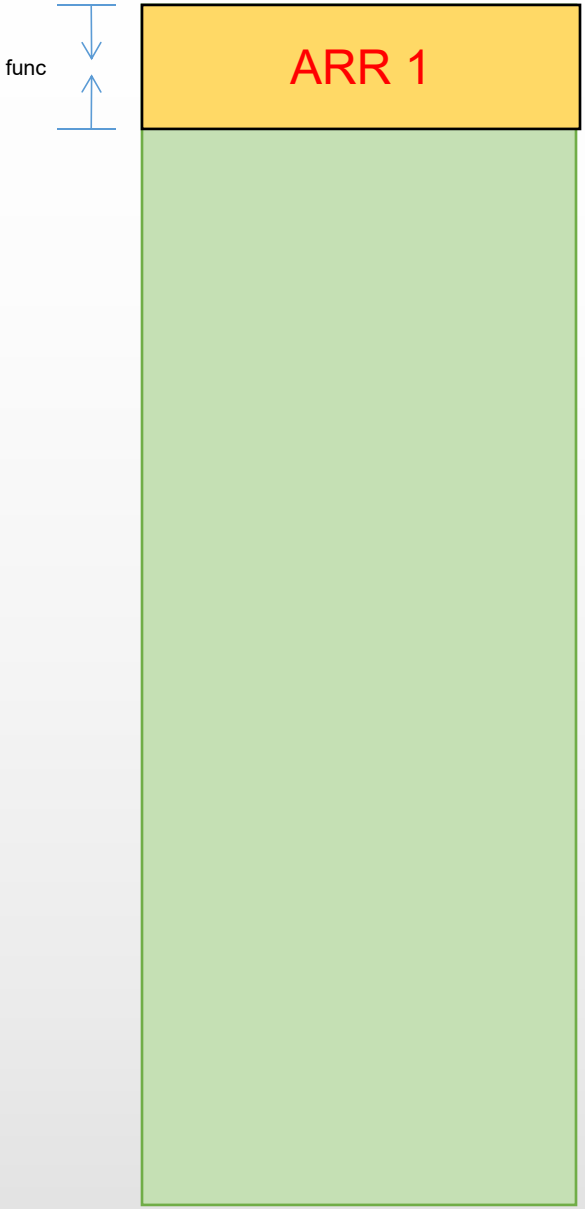
问题：

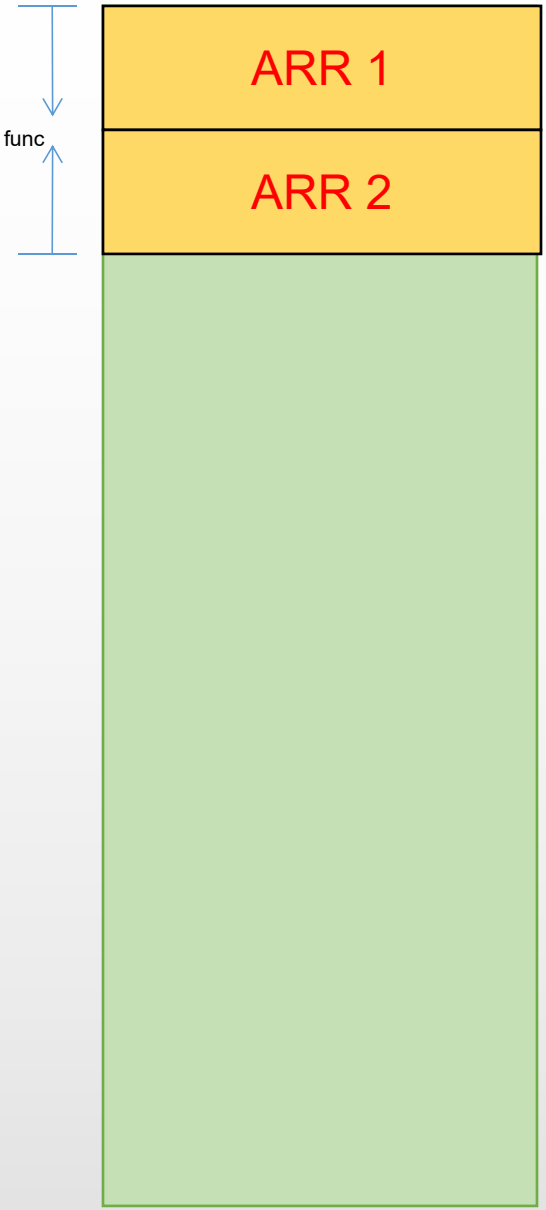
1. 函数递归次数过多，为什么引起栈溢出。

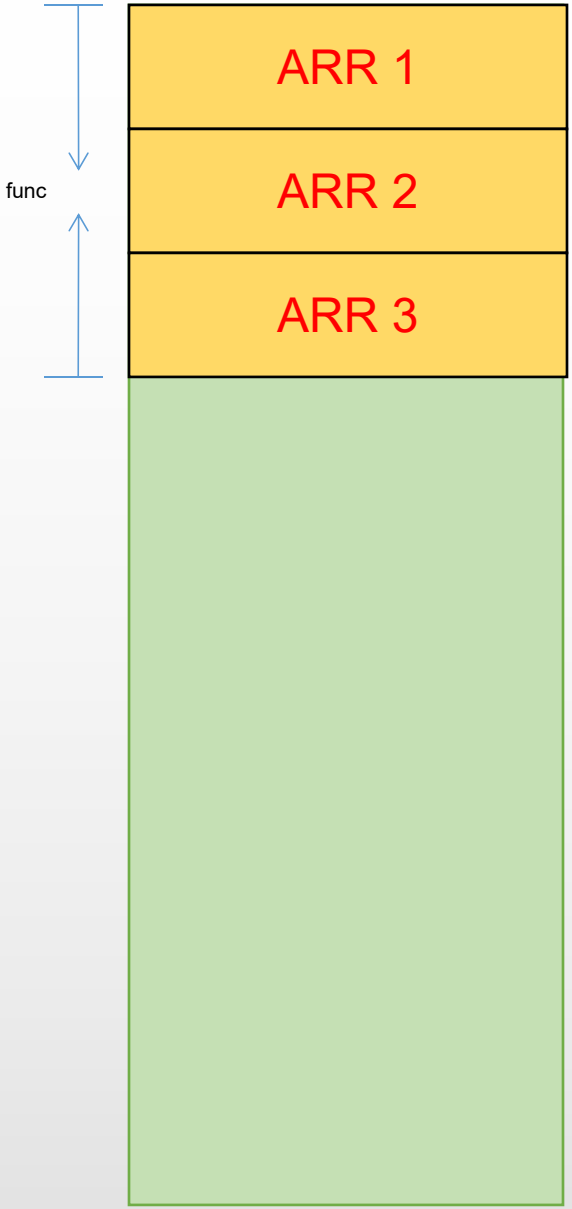
2. 变量和栈的关系与函数和栈的关系有没有什么相似的地方。

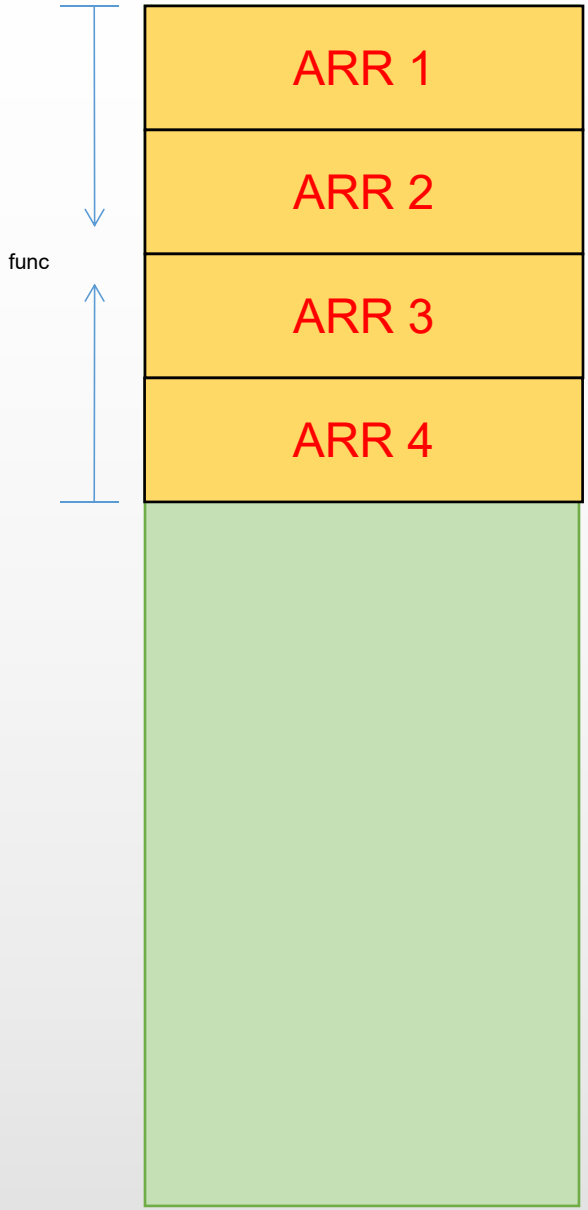
3. 函数之间不能相互访问彼此内部的变量，是为什么。

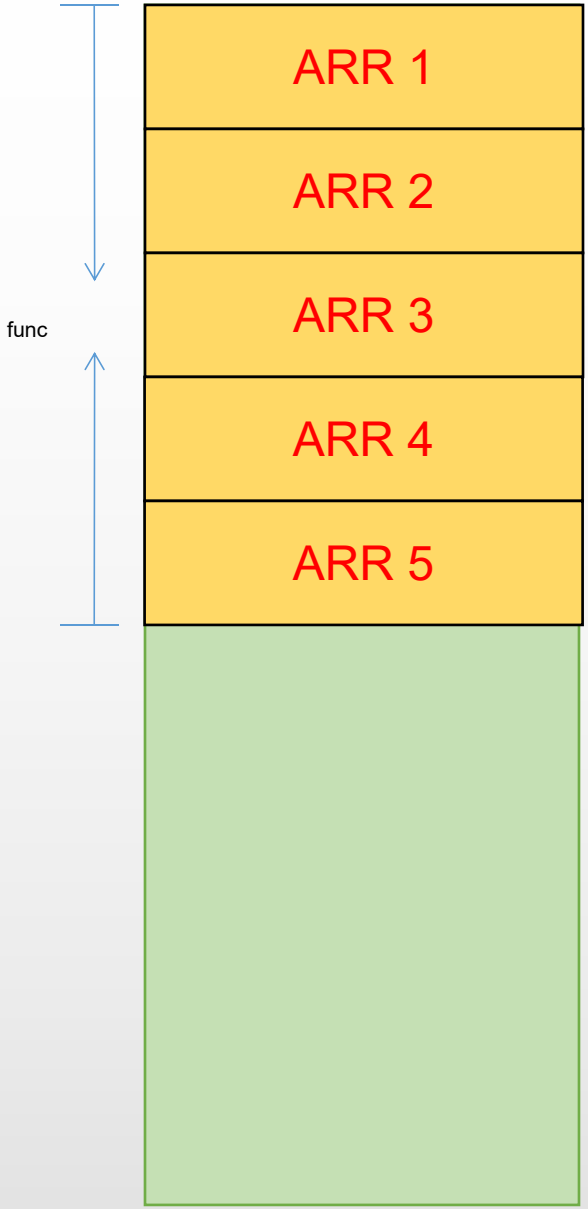
4. 所有函数都可以访问全局变量和全局常量是为什么。

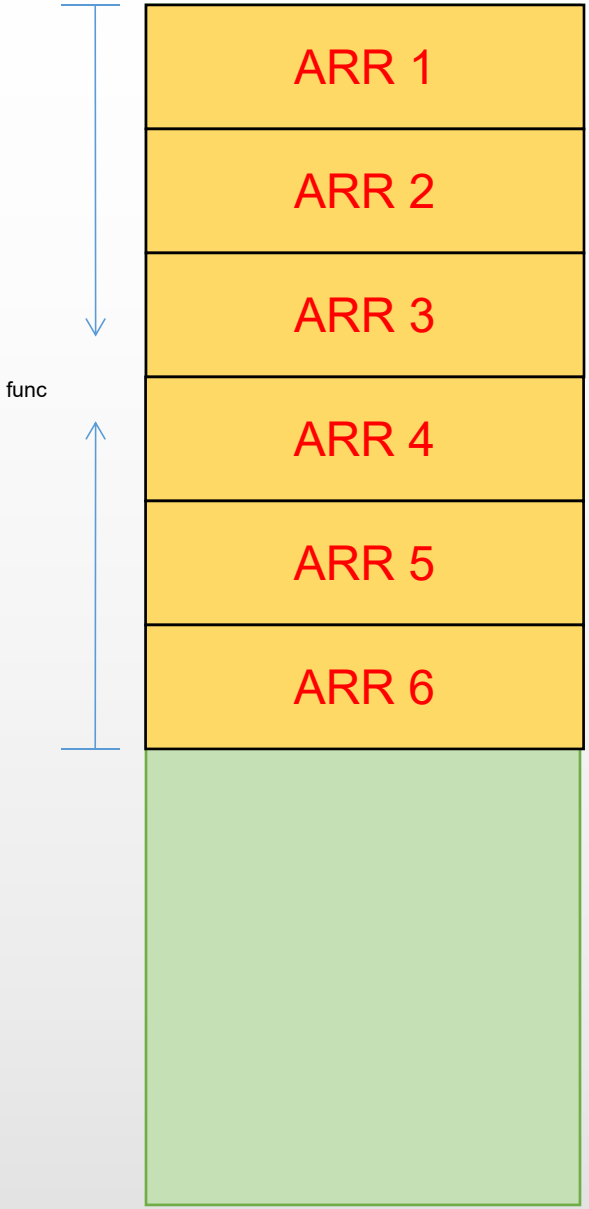

func
ARR 1


ARR 1
ARR 2
func


ARR 1
ARR 2
ARR 3
func


ARR 1
ARR 2
ARR 3
ARR 4
func


ARR 1
ARR 2
ARR 3
ARR 4
ARR 5
func


ARR 1
ARR 2
ARR 3
func
ARR 4
ARR 5
ARR 6

写程序，函数中声明变量，并且查看地址

1.引言

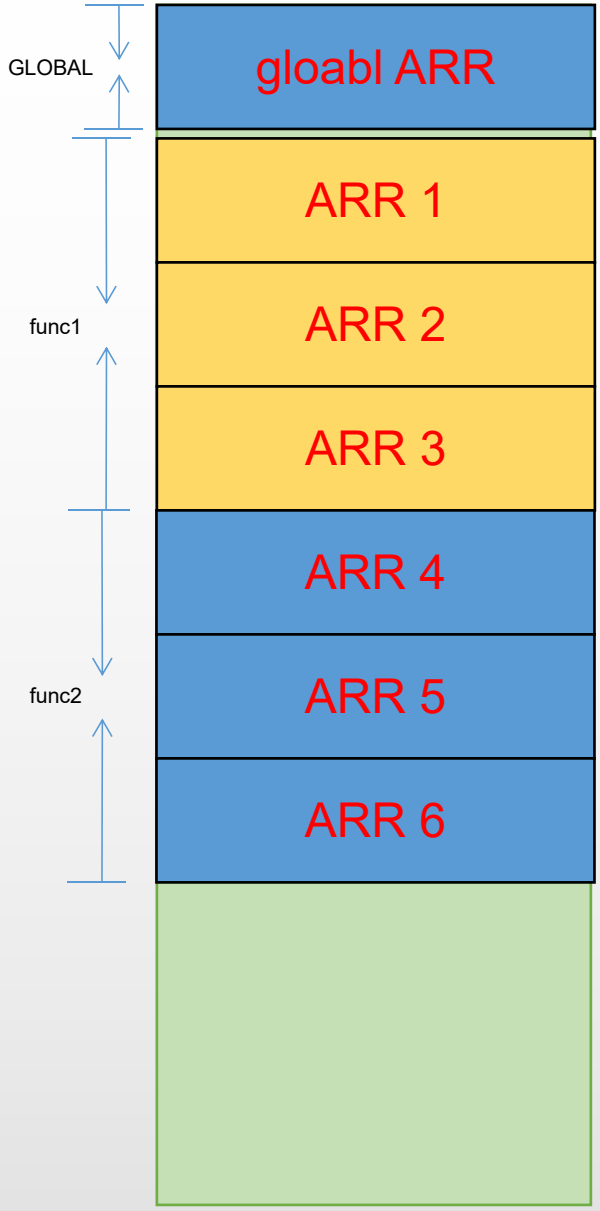
问题：

1. 函数递归次数过多，为什么引起栈溢出。

2. 变量和栈的关系与函数和栈的关系有没有什么相似的地方。

3. 函数之间不能相互访问彼此内部的变量，是为什么。

4. 所有函数都可以访问全局变量和全局常量是为什么。



简单的讲，函数执行时，能访问的地址范围
就是当前函数的栈空间，和全局变量以及RO存储空间。

1.引言

问题：

1. 函数递归次数过多，为什么引起栈溢出。

2. 变量和栈的关系与函数和栈的关系有没有什么相似的地方。

3. 函数之间不能相互访问彼此内部的变量，是为什么。

4. 所有函数都可以访问全局变量和全局常量是为什么。

答：全局变量和常量并不属于函数栈。
他们属于RO或者RW中非函数栈的内存空间。

4.程序的内存模型

第 2 节讲的那种模型，只是程序运行空间中的一部分。
内存空间中，还存在一块区域叫做堆。

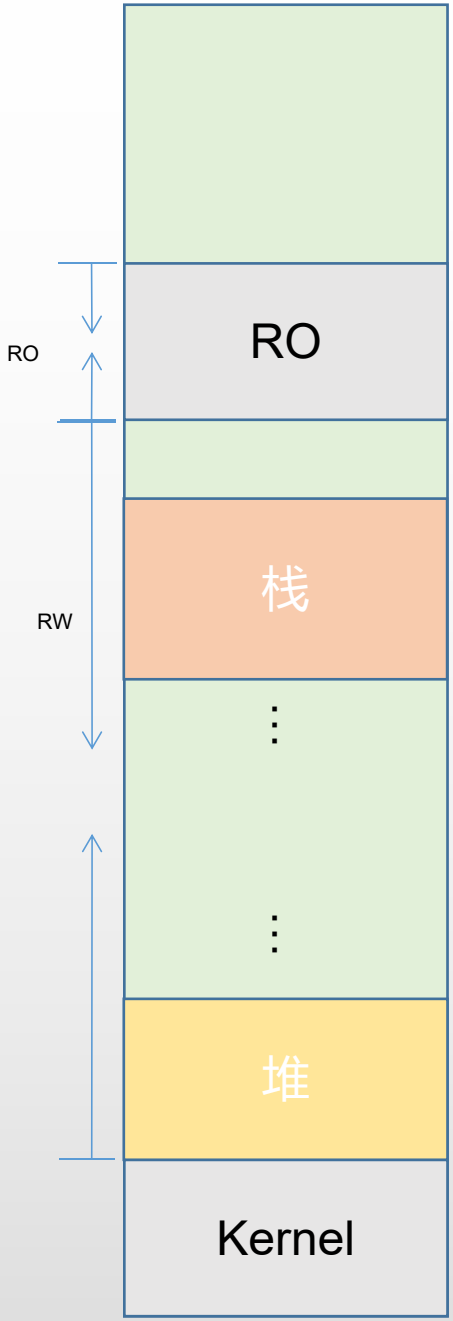
4.程序的内存模型

内存空间中，栈向下生长，堆向上生长。

栈内存放的是函数栈

堆中存放的是自由分配的内存空间

比如：c语言中，使用malloc()进行分配的内存地址空间，就来自堆。



4.程序的内存模型

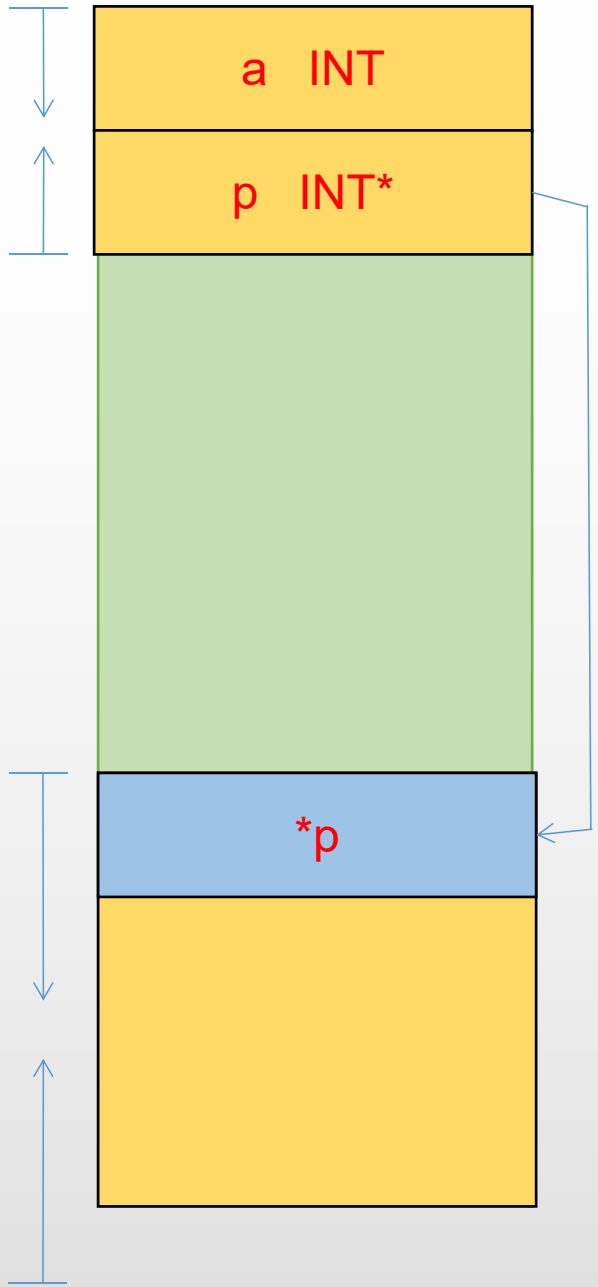
写程序如下

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  int main(void)
4  {
5      int a=123;
6      int* p = (int*)malloc(sizeof(int));
7      return 0;
8  }
```

4.程序的内存模型

变量及地址空间分配

```c
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int a=123;
    int* p = (int*)malloc(sizeof(int));
    return 0;
}
```



但这里需要注意，堆并不是函数自己堆，是所有函数共同使用的，访问时只需要使用指针即可

java中的一段代码

```
String str = new String();
```

java声称自己没有指针
但实际上java中到处都使用指针