栈帧是用于支持虚拟机进行方法调用和方法执行的数据结构，是虚拟机运行时数据区中的虚拟机栈的栈元素。栈帧存储了方法的**局部变量表，操作数栈，动态链接以及方法返回地址**等信息。每一个方法的调用开始到结束都对应着一个栈帧在虚拟机栈里从入栈到出栈的过程。每一个栈帧都包括了局部变量表，操作数栈，动态链接，方法返回地址和一些额外的附加信息。**在编译程序代码的时候，栈帧中需要多大的局部变量表，多深的操作数栈都已经完全确定了并写入到方法表的Code属性中。因此一个栈帧需要分配多少内存，不是受程序运行期变量数据决定，而是仅仅取决于具体的虚拟机实现。**
一个线程中的方法调用链可能会很长，很多方法都同时处于执行状态。对于执行引擎来说，在活动线程中，只有位于栈顶的栈帧才是有效的，称为当前栈帧

**1.局部变量表**
局部变量表是一组变量的存储空间，用于存放方法参数和方法内部定义的局部表变量，在Java编程成class文件时，就在方法的Code属性中的max_locals数据项中定义了该方法需要分配局部变量表的最大容量。
局部变量表的容量已变量槽（Slot）为最小单位，而且虚拟机中没有决定Slot的内存占用大小，只是说每个Slot都应该能存放一个boolean, byte, char, int , short, float, reference或者returnAddress类型的数据，这8中数据类型，都可以使用32位或者更小的物理内存来存放，但这种描述明确的指出每个Slot占用32位长度的内存空间有些差别，允许slot的长度随着处理器，操作系统或虚拟机的不同而存在变化。对于reference类型，虚拟机并没有明确指出这种引用应有的结构和长度，但是一般来说，虚拟机实现至少都应该通过这种引用做到两点，一是从此引用中直接或者间接地查找到对象在Java堆中的数据存放的起始地址索引。而是此引用中直接或间接地查找到对象所属数据类型在方法区中的存储类型信息。而ReturnAddress类型目前已经很少见了，指向了一条字节码指令地址。对于64位的数据类型，虚拟机会使用高位对其的方式为其分配两个连续的Slot空间。java语言中明确规定reference类型可能是32位可能是64位，而64位的数据只有long和double两种，这里的做法就是把long和double类型的数据分割为两次32为读写的做法。不过由于局部变量表建立在成的堆栈上，是线程私有的数据，无论读写两个连续的slot是否是原子操作，都不会引起数据安全问题。在方法执行的时候，虚拟机是使用局部变量表完成参数变量列表的传递过程，如果执行的是实例方法(非static)，那局部变量表中的第0位索引的slot默认是用于传递方法所属实例的引用，即通过关键字this来访问到这个隐含参数，其余的参数则按照

参数表顺序排列，占用从1开始的局部变量Slot。关于局部变量表还有一点可能会对实际开发有影响，就是局部变不像之前介绍的类变量一样会被赋初值，如int a;会被赋初值为0，如果局部变量没有赋予初始值是不能使用的，这段代码是不能运行的

**2.操作数栈**

操作数栈也常被称为操作栈，他是一个后入先出的栈，同局部变量一样，操作数栈的最大深度也在编译的时候写入到Code属性的max_stacks数据项中。操作数栈的每一个元素可以是任意的java数据类型。32位数据类型所占用的栈容量是1,64位数据项占用的站容量是2。

两个栈帧作为虚拟机栈的元素是完全相互独立的，但是虚拟机进行了优化，令两个虚拟机栈出现了一部分重叠，让下面的虚拟机的部分操作数栈与上面的栈帧的部分局部表变量表重叠在一起。java虚拟机的解释机制那个引擎称为“基于栈的执行引擎”，其中所指的“栈”就是操作数栈

**3. 动态链接**

每个栈帧都包含一个指向运行时常量池中**该栈帧所属方法的引用**，持有这个引用是为了支持方法调用过程中的动态链接。

在Class文件的常亮池中存在大量符号引用，字节码中的方法调用指令就是以常量池中指向方法的符号引用作为参数。这些符号引用一部分在类加载阶段或者第一次使用的时候转成了直接引用这种称为静态解析，另外一部分是在每一次运行期间转化为直接引用，这部分称为动态链接

**4. 方法返回地址**

当一个方法执行后，有两种退出方式，第一种方式是遇到人一个方法返回的字节码指令，这个时候可能会有返回值传递给上层调用者，是否有返回值和返回值类型将根据遇到何种方法返回指令来决定的。这种称谓正常完成出口

另一种退出是，在方法执行过程中遇到了异常，并且这个异常没有再方法体内得到处理，无论是java虚拟机内存产生的异常，还是代码中通过athrow字节码指令产生的异常导致的退出称为异常完成出口，是不会给上层调用者任何返回的

无论任何方式的退出，在方法退出之后，**都需要返回到方法被调用的位置，程序才能继续执行**。方法返回时可能需要在栈帧中保存一些信息，用来恢复它上层调用方法的执行状态。一般来说，正常退出调用者pc计数器的值就可以作为返回地址，栈帧中很可能保存这个计数器值。如果是异常退出，返回地址是通过异常处

理器表来确定的，栈帧中不会保存这些信息。方法退出过程实际上就等同于把当前栈帧出站，因此退出时可能执行的操作有：恢复上层方法的局部变量表和操作数栈，把返回值压入调用者栈帧的操作数栈中，调整pc计数器的值以指向方法调用指令后的一条指令。