1.什么是MVC框架？传统MVC框架存在的问题是什么？

MVC框架是为了解决传统MVC模式(Jsp + Servlet + JavaBean)的一些问题而出现的框架。

传统MVC模式存在问题：

1. 所有的Servlet和Servlet映射都要配置在web.xml中，如果项目太大，web.xml就太庞大，并且不能实现模块化管理。虽然servelt3.0能使用注解，但是需要建立的类太多了.
2. Servlet的主要功能就是接受参数、调用逻辑、跳转页面，比如像其他字符编码、文件上传等功能也要写在Servlet中 。
3. 接受参数比较麻烦

如： String name = request.getParameter(“name”)

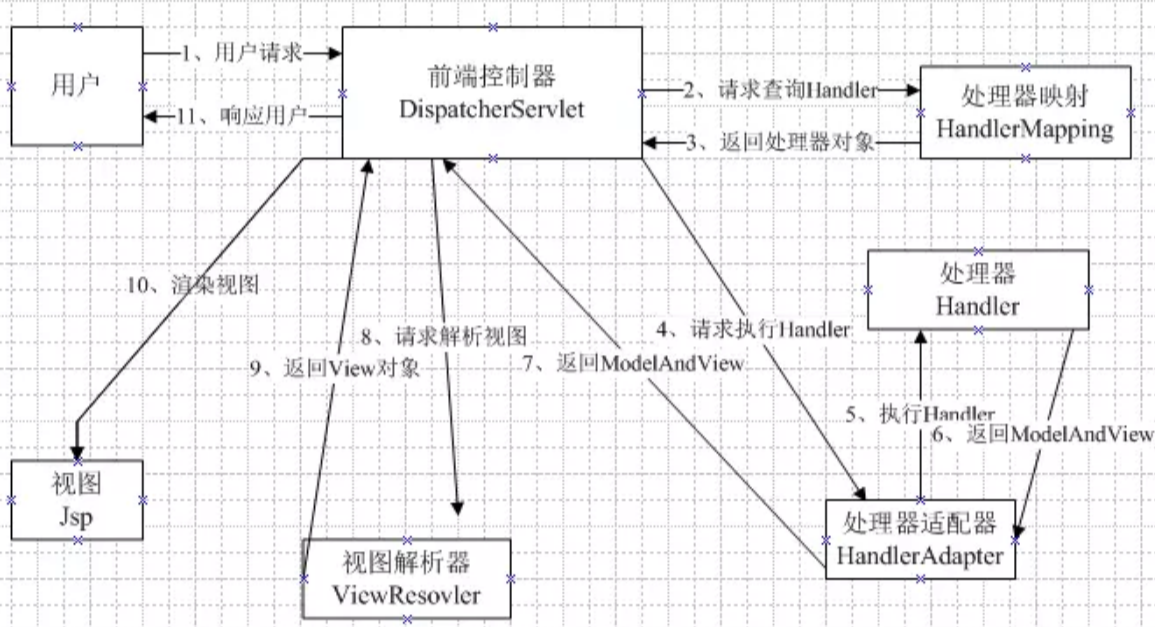
User user=new User();

user.setName(name))， 只能单个接收，接收完成后转换封装model.

1. 跳转页面方式比较单一(forword,redirect),并且当我的页面名称发生改变时需要修改Servlet源代码.

2简单介绍下你对springMVC特点的理解?

1. 它是基于组件技术的.全部的应用对象,无论控制器和视图,还是业务对象之类的都是java组件.并且和Spring提供的其他基础结构紧密集成.
2. 不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于S.ervlet的)
3. 可以任意使用各种视图技术,而不仅仅局限于JSP
4. 支持各种请求资源的映射策略
5. 它应是易于扩展的
6. 简单的谈一下SpringMVC的工作流程？★★★★★



流程

1、用户发送请求至前端控制器DispatcherServlet

2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4、DispatcherServlet调用HandlerAdapter处理器适配器

5、HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、Controller执行完成返回ModelAndView

7、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9、ViewReslover解析后返回具体View

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、DispatcherServlet响应用户

4.SpringMVC与Struts2的主要区别？

1. spring mvc核心控制器是Servlet，而Struts2是Filter。
2. Spring Mvc是基于方法设计，而Sturts是基于对象，每次发一次请求都会实例一个action，每个action都会被注入 属性，而Spring更像Servlet一样，只有一个实例，每次请求执行对应的方法即可(注意：由于是单例实例，所以应当避免全局变量的修改，这样会产生线程安全问题)。
3. 管理方式：大部分的公司的核心架构中，就会使用到spring,而spring mvc又是spring中的一个模块，所以spring对于spring mvc的控制器管理更加简单方便，而且提供了全 注解方式进行管理，各种功能的注解都比较全面，使用简单，而struts2需要采用XML很多的配置参数来管理（虽然也可以采用注解，但是几乎没有公司那 样使用）。
4. 参数传递：Struts2中自身提供多种参数接受，其实都是通过（ValueStack）进行传递和赋值，而SpringMvc是通过方法的参数进行接收。
5. 学习难度：Struts更加很多新的技术点，比如拦截器、值栈及OGNL表达式，学习成本较高，springmvc 比较简单，很较少的时间都能上手。
6. intercepter 的实现机制：struts有以自己的interceptor机制，spring mvc用的是独立的AOP方式。这样导致struts的配置文件量还是比spring mvc大，虽然struts的配置能继承，所以我觉得论使用上来讲，spring mvc使用更加简洁，开发效率Spring MVC确实比struts2高。
7. spring mvc处理ajax请求,直接通过返回数据，方法中使用注解@ResponseBody，spring mvc自动帮我们对象转换为JSON数据。而struts2是通过插件的方式进行处理
8. 在SpringMVC流行起来之前，Struts2在MVC框架中占核心地位，随着SpringMVC的出现，SpringMVC慢慢的取代struts2,但是很多企业都是原来搭建的框架，使用Struts2较多。

5.如何解决POST请求中文乱码问题，GET的又如何处理呢？

在web.xml中加入：

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>**org.springframework.web.filter.CharacterEncodingFilter**</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceResponseEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>.

以上可以解决post请求乱码问题。

对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：



另外一种方法对参数进行重新编码：

String params=request.getParamter(“userName”);

String userName = new String(**params.getBytes(“ISO8859-1”)**,“utf-8”)

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

6.说出SpringMVC常用的5个注解？如何使用 SpringMVC完成JSON操作？：

常用的 5 个注解

@RequestMapping 、 @PathVariable 、 @RequestParam 、 @RequestBoy 、

@ResponseBody

如何使用 SpringMVC 完成 JSON 操作：

①. 配置 MappingJacksonHttpMessageConverter

②. 使用 @RequestBody 注解或 ResponseEntity 作为返回值

7. Spring 支持的事务管理类型有哪些？你在项目中使用哪种方式？怎么理解全局事务和局部事务？

Spring 支持编程式事务管理和声明式事务管理。许多 Spring 框架的用户选择声明式事务管理，因为这种方式和应用程序的关联较少，因此更加符合轻量级容器的概念。声明式事务管理要优于编程式事务管理，尽管在灵活性方面它弱于编程式事务管理，因为编程式事务允许你通过代码控制业务。

声明式事务又分为两种： **a、基于XML的声明式事务** b、基于注解的声明式事务

事务分为全局事务和局部事务。全局事务由应用服务器管理，需要底层服务器 JTA 支持（如 WebLogic、 WildFly等）。局部事务和底层采用的持久化方案有关，例如使用 JDBC 进行持久化时，需要使用 Connetion 对象来操作事务；

8.简单介绍下你对mybatis的理解？

1. mybatis配置

2. SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。

3. mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

4. 通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂

5. 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。

6. mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。

7. Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。

8. Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。

9. Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

10.SpringMvc的控制器是不是单例模式,如果是,有什么问题,怎么解决?

是单例模式,所以在多线程访问的时候有线程安全问题,不要用同步,会影响性能的,解决方案是在控制器里面不能写字段。

11. （1）@RequestMapping注解用在类上面有什么作用？（2）怎么样把某个请求映射到特定的方法上面？

（1）@RequestMapping注解用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

（2）如何把某个请求映射到特定的方法上面方案：直接在方法上面加上注解@RequestMapping,并且在这个注解里面写上要拦截的路径

12（1）如果在拦截请求中,我想拦截get方式提交的方法,怎么配置？

(2) 如果在拦截请求中,我想拦截提交参数中包含"type=test"字符串,怎么配置?

可以在@RequestMapping注解里面加上method=RequestMethod.GET

可以在@RequestMapping注解里面加上params=“type=test”

MyBatis部分

1讲下对MyBatis的缓存的理解？

MyBatis的缓存分为一级缓存(SqlSession)和二级缓存(SqlSessionFactory),

一级缓存放在session里面,默认就有，需要在同一个连接对象中使用;

二级缓存放在它的命名空间里,默认是打开的,使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置

2说一下orm与jdbc的区别？

jdbc只是一个java操作数据库的规范接口而已

orm 不过是一种思想，对象关系映射。

ORM：是对象关系模型，如hibernate,让你以面向对象的方式去编程。封装了JDBC.

JDBC：是从底层访问数据库服务器。一般银行，金融行业为了安全起见，直接用JDBC访问

3.JDBC编程有哪些不足之处，MyBatis是如何解决这些问题的？

① 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

② Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

③ 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决： Mybatis自动将java对象映射至sql语句。

④ 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决：Mybatis自动将sql执行结果映射至java对象。

4.MyBatis编程步骤是什么样的？

① 创建SqlSessionFactory

② 通过SqlSessionFactory创建SqlSession

③ 通过sqlsession执行数据库操作

④ 调用session.commit()提交事务

⑤ 调用session.close()关闭会话

5、 mybatis中${value}与#{} 的区别是什么？

* ${}：字符串替换，sql中的${}替换成？，有效的防止Sql语句注入。
* #{}：预编译处理，把#{}直接替换成变量的值，不做任何转换。  
  总结：一般用#{}来进行列的代替

6、 JDBC 与数据库交互的流程是什么?

加载数据库驱动

获取数据库连接

创建statement 对象

设置SQL语句

设置SQL语句参数

使用Statement对象执行SQL语句

获取结果集 解析结果集

关闭资源

7.解释一下什么是 aop？

AOP是Spring提供的关键特性之一。AOP即面向切面编程，是OOP编程的有效补充。

使用AOP技术，可以将一些系统性相关的编程工作，独立提取出来，独立实现，然后通过切面切入进系统。

8.解释一下什么是 ioc？

ioc（inverse of control ）控制反转：所谓控制反转就是把对象（bean）对象和维护对象（bean）之间的关系的权利转移到Sqring容器中去了（ApplicationContext.xml）而程序本身不在维护了

9、spring 的事务隔离级别

<https://www.cnblogs.com/wj0816/p/8474743.html>

1） PROPAGATION\_REQUIRED ，默认的spring事务传播级别，使用该级别的特点是，如果上下文中已经存在事务，那么就加入到事务中执行，如果当前上下文中不存在事务，则新建事务执行。所以这个级别通常能满足处理大多数的业务场景。

2）PROPAGATION\_SUPPORTS ，从字面意思就知道，supports，支持，该传播级别的特点是，如果上下文存在事务，则支持事务加入事务，如果没有事务，则使用非事务的方式执行。所以说，并非所有的包在transactionTemplate.execute中的代码都会有事务支持。这个通常是用来处理那些并非原子性的非核心业务逻辑操作。应用场景较少。

3）PROPAGATION\_MANDATORY ， 该级别的事务要求上下文中必须要存在事务，否则就会抛出异常！配置该方式的传播级别是有效的控制上下文调用代码遗漏添加事务控制的保证手段。比如一段代码不能单独被调用执行，但是一旦被调用，就必须有事务包含的情况，就可以使用这个传播级别。

4）PROPAGATION\_REQUIRES\_NEW ，从字面即可知道，new，每次都要一个新事务，该传播级别的特点是，每次都会新建一个事务，并且同时将上下文中的事务挂起，执行当前新建事务完成以后，上下文事务恢复再执行。

这是一个很有用的传播级别，举一个应用场景：现在有一个发送100个红包的操作，在发送之前，要做一些系统的初始化、验证、数据记录操作，然后发送100封红包，然后再记录发送日志，发送日志要求100%的准确，如果日志不准确，那么整个父事务逻辑需要回滚。

怎么处理整个业务需求呢？就是通过这个PROPAGATION\_REQUIRES\_NEW 级别的事务传播控制就可以完成。发送红包的子事务不会直接影响到父事务的提交和回滚。

5）PROPAGATION\_NOT\_SUPPORTED ，这个也可以从字面得知，not supported ，不支持，当前级别的特点就是上下文中存在事务，则挂起事务，执行当前逻辑，结束后恢复上下文的事务。

这个级别有什么好处？可以帮助你将事务极可能的缩小。我们知道一个事务越大，它存在的风险也就越多。所以在处理事务的过程中，要保证尽可能的缩小范围。比如一段代码，是每次逻辑操作都必须调用的，比如循环1000次的某个非核心业务逻辑操作。这样的代码如果包在事务中，势必造成事务太大，导致出现一些难以考虑周全的异常情况。所以这个事务这个级别的传播级别就派上用场了。用当前级别的事务模板抱起来就可以了。

6）PROPAGATION\_NEVER ，该事务更严格，上面一个事务传播级别只是不支持而已，有事务就挂起，而PROPAGATION\_NEVER传播级别要求上下文中不能存在事务，一旦有事务，就抛出runtime异常，强制停止执行！这个级别上辈子跟事务有仇。

7）PROPAGATION\_NESTED ，字面也可知道，nested，嵌套级别事务。该传播级别特征是，如果上下文中存在事务，则嵌套事务执行，如果不存在事务，则新建事务

# redis篇

1.redis 是什么？都有哪些使用场景？

Redis是一个开源的 基于内存的 key—value型 单线程 数据库，

支持string、list、set、zset和hash类型数据。

默认端口:6379

默认数据库数量:16

适用场景：1).数据高并发的读写

2).海量数据的读写，对扩展性要求高的数据

2.redis 为什么是单线程的？

（1）纯内存操作；

（2）多线程仍然会有上下文切换的损耗，虽然比进程切换损耗小；

（3）采用了非阻塞I/O多路复用机制

<https://my.oschina.net/hanchao/blog/3057400>

3.什么是缓存穿透？怎么解决？

<https://blog.csdn.net/muyi_amen/article/details/80229647>

缓存穿透，是指查询一个数据库一定不存在的数据。正常的使用缓存流程大致是，数据查询先进行缓存查询，如果key不存在或者key已经过期，再对数据库进行查询，并把查询到的对象，放进缓存。如果数据库查询对象为空，则不放进缓存。

缓存穿透解决方案：

1. 再web服务器启动时，提前将有可能被频繁并发访问的数据写入缓存。
2. 规范key的命名，并且统一缓存查询和写入的入口。这样，在入口处，对key的规范进行检测。–这样预防恶意的key被拦截。
3. Synchronized双重检测机制，这时我们就需要使用同步（Synchronized）机制，在同步代码块前查询一下缓存是否存在对应的key，然后同步代码块里面再次查询缓存里是否有要查询的key。 这样“双重检测”的目的，还是避免并发场景下导致的没有意义的数据库的访问（也是一种严格避免穿透的方案）。这一步会导致排队，但是第一步中我们说过，为了避免大量的排队，可以提前将可以预知的大量请求提前写入缓存。
4. 不管数据库中是否有数据，都在缓存中保存对应的key，值为空就行。–这样是为了避免数据库中没有这个数据，导致的平凡穿透缓存对数据库进行访问。
5. 第4步中的空值如果太多，也会导致内存耗尽。导致不必要的内存消耗。这样就要定期的清理空值的key。避免内存被恶意占满。导致正常的功能不能缓存数据。

6.redis 支持的数据类型有哪些？

支持string、list、set、zset和hash类型数据

9.怎么保证缓存和数据库数据的一致性？

<https://www.jianshu.com/p/2936a5c65e6b>

1. redis 持久化有几种方式？

<https://www.cnblogs.com/AndyAo/p/8135980.html>

redis提供两种方式进行持久化，

一种是RDB持久化（原理是将Reids在内存中的数据库记录定时 dump到磁盘上的RDB持久化），

另外一种是AOF（append only file）持久化（原理是将Reids的操作日志以追加的方式写入文件）

1. redis 怎么实现分布式锁？

<https://mp.weixin.qq.com/s/8fdBKAyHZrfHmSajXT_dnA>

<https://blog.csdn.net/tuesdayma/article/details/82751790>

13.redis 如何做内存优化？

<https://www.cnblogs.com/jandison/p/6902396.html>

1. redis 淘汰策略有哪些？

<https://www.cnblogs.com/Xuuuuuu/p/10796004.html>

1）voltile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

2）volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

3）volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

4）allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

5）allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

6）no-enviction（驱逐）：禁止驱逐数据

15.redis 常见的性能问题有哪些？该如何解决？

　1).Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以Master最好不要写内存快照。

　2).Master AOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF文件过大会影响Master重启的恢复速度。Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。

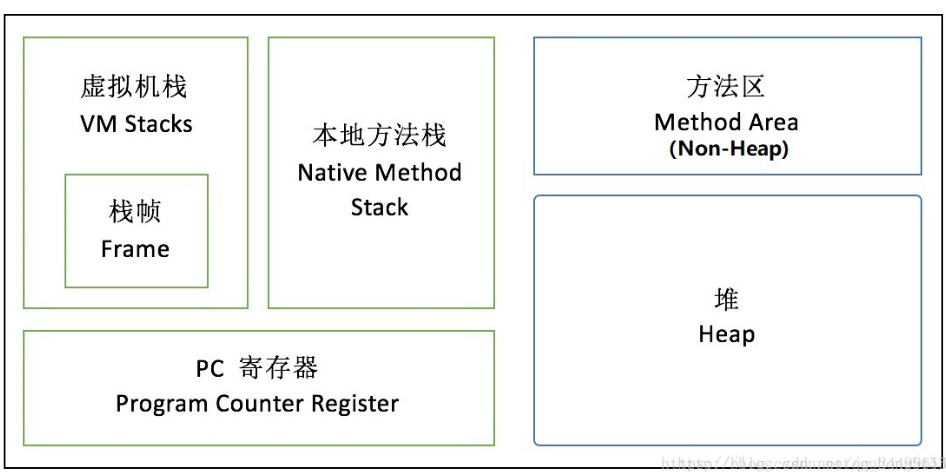
3).Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，出现短暂服务暂停现象。

4). Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内

# 四、JVM 虚拟机

1、JVM内存分为哪几部分?各个部分的作用是什么?

****Java虚拟机内存的五大区域****   
JVM内存区域分为五个部分，分别是堆，方法区，虚拟机栈，本地方法栈，程序计数器。



1)、堆。 堆是Java对象的存储区域，任何用new字段分配的Java对象实例和数组，都被分配在堆上，Java堆可使用-Xms -Xmx进行内存控制，值得一提的是从JDK1.7版本之后，运行时常量池从方法区移到了堆上。

2)、方法区。它用于存储已被虚拟机加载的类信息，常量，静态变量，即时编译器编译后的代码等数据，方法区在JDK1.7版本及以前被称为永久代，从JDK1.8永久代被移除。

3)、虚拟机栈。虚拟机栈中执行每个方法的时候，都会创建一个栈帧用于存储局部变量表，操作数栈，动态链接，方法出口等信息。

4)、本地方法栈。与虚拟机栈发挥的作用相似，相比于虚拟机栈为Java方法服务，本地方法栈为虚拟机使用的Native方法服务，执行每个本地方法的时候，都会创建一个栈帧用于存储局部变量表，操作数栈，动态链接，方法出口等信息。

5)、程序计数器。指示Java虚拟机下一条需要执行的字节码指令。

以上五个区域是Java虚拟机内存划分情况，其中方法区和堆被JVM中多个线程共享，比如类的静态常量就被存放在方法区，供类对象之间共享，虚拟机栈，本地方法栈，pc寄存器是每个线程独立拥有的，不会与其他线程共享。

所以Java在通过new创建一个类对象实例的时候，一方面会在虚拟机栈中创建一个该对象的引用，另一方面会在堆上创建类对象的实例，然后将对象引用指向该对象的实例。对象引用存放在每一个方法对应的栈帧中。

**2、说一下堆栈的区别**

栈内存:栈内存首先是一片内存区域，存储的都是局部变量，凡是定义在方法中的都是局部变量（方法外的是全局变量），for循环内部定义的也是局部变量，是先加载函数才能进行局部变量的定义，所以方法先进栈，然后再定义变量，变量有自己的作用域，一旦离开作用域，变量就会被释放。栈内存的更新速度很快，因为局部变量的生命周期都很短。

堆内存:存储的是数组和对象（其实数组就是对象），凡是new建立的都是在堆中，堆中存放的都是实体（对象），实体用于封装数据，而且是封装多个（实体的多个属性），如果一个数据消失，这个实体也没有消失，还可以用，所以堆是不会随时释放的，但是栈不一样，栈里存放的都是单个变量，变量被释放了，那就没有了。堆里的实体虽然不会被释放，但是会被当成垃圾，Java有垃圾回收机制不定时的收取。

3、jvm 有哪些垃圾回收算法

<https://www.cnblogs.com/aspirant/p/8662690.html>

1)、引用计数法

引用计数法实现简单，效率较高，在大部分情况下是一个不错的算法。其原理是：给对象添加一个引用计数器，每当有一个地方引用该对象时，计数器加1，当引用失效时，计数器减1，当计数器值为0时表示该对象不再被使用。需要注意的是：引用计数法很难解决对象之间相互循环引用的问题，主流Java虚拟机没有选用引用计数法来管理内存。

1. 、标记-清除算法(Mark-Sweep）
2. 、复制算法(Copying)
3. 标记-整理算法(Mark-compact)
4. 分代收集算法

5.1)、年轻代（Young Generation）的回收算法 (回收主要以Copying为主)

5.2)、年老代（Old Generation）的回收算法（回收主要以Mark-Compact为主）

5.3)、持久代（Permanent Generation）(也就是方法区)的回收算法

1. **GC是什么时候触发的**

由于对象进行了分代处理，因此垃圾回收区域、时间也不一样。GC有两种类型：Scavenge GC和Full GC。

6.1 Scavenge GC

一般情况下，当新对象生成，并且在Eden申请空间失败时，就会触发Scavenge GC，对Eden区域进行GC，清除非存活对象，并且把尚且存活的对象移动到Survivor区。然后整理Survivor的两个区。这种方式的GC是对年轻代的Eden区进行，不会影响到年老代。因为大部分对象都是从Eden区开始的，同时Eden区不会分配的很大，所以Eden区的GC会频繁进行。因而，一般在这里需要使用速度快、效率高的算法，使Eden去能尽快空闲出来

6.2 Full GC

对整个堆进行整理，包括Young、Tenured和Perm。Full GC因为需要对整个堆进行回收，所以比Scavenge GC要慢，因此应该尽可能减少Full GC的次数。在对JVM调优的过程中，很大一部分工作就是对于Full GC的调节。有如下原因可能导致Full GC：

a) 年老代（Tenured）被写满；

b) 持久代（Perm）被写满；

c) System.gc()被显示调用；

d) 上一次GC之后Heap的各域分配策略动态变化；