## Java 工程师成神之路 | 2019正式版

***1***

**基础篇**

01

 面向对象

#### ****→ 什么是面向对象****

面向对象、面向过程

面向对象的三大基本特征和五大基本原则.

答: 面向对象是向现实世界模型的自然延伸,这是一种”万物皆对象”的编程思想,在现实生活中的任何物体都可以归为一个事物,而每一个个体都是一个事物的实例。

面向对象有三大特征：封装、继承、多态。

五大基本原则：单一职责原则（SRP）,开放封闭原则(OCP),里氏替换原则(DIP),接口隔离原则(ISP)。

#### → 平台无关性

Java 如何实现的平台无关

JVM 还支持哪些语言（Kotlin、Groovy、JRuby、Jython、Scala）

#### → 值传递

值传递、引用传递

为什么说 Java 中只有值传递？

答： 1.值传递的就是传递变量的值。2.引用传递就是传递变量的地址。

#### → 封装、继承、多态

什么是多态、方法重写与重载？

Java 的继承与实现

构造函数与默认构造函数

类变量、成员变量和局部变量

成员变量和方法作用域

重载与重写是Java多态性的不同表现。

重写是父类与子类之间多态性的表现，在运行时起作用（如实现动态绑定）

重载是一个类中多态性的表现，在编译时起作用（如实现静态绑定）

02 Java 基础知识

#### → 基本数据类型

8 种基本数据类型：整型、浮点型、布尔型、字符型.

整型中 byte、short、int、long 的取值范围

什么是浮点型？什么是单精度和双精度？为什么不能用浮点型表示金额？

答案:

btye: -2^7 -1(127) ~ 2^8(128)

short: -2^15-1(3万两千多) ~2^15

int: -2^31-1 ~ 2^31

long: -2^63 ~ 2^63

由于浮点数表示的这种“不精确性”或者说是“近似性”， 对于精确度要求不高的运算还行，“使用BigDecimal来做精确运算”。

#### → 自动拆装箱

什么是包装类型、什么是基本类型、什么是自动拆装箱

Integer 的缓存机制

答案:

基本数据类型是直接在栈内存中创建的,将数值直接存储在栈上。

Java是一个面向对象的语言,基本数据类型不具有对象的性质,为了与其他对象”接轨”就出现了包装类型，为其添加属性和方法，丰富了基本类型的操作。

Integer是对(-128~127)有缓存,在JVM缓存的时候,数据在(-128~127)之间的数字便被缓存到了本地内存中,如果初始化-128~127之间的数字,将会直接从内存中取出,不需要创建新对象。

补充： Java中的==用于判断两个操作是否相等，如果操作是基本数据类型，则判断数值是否相等；如果操作是对象，则判断两个对象的地址是否相等（也就是引用是否相等）。

#### → String

字符串的不可变性

JDK 6 和 JDK 7 中 substring 的原理及区别、

replaceFirst、replaceAll、replace 区别、

String 对“+”的重载、字符串拼接的几种方式和区别

String.valueOf 和 Integer.toString 的区别、

switch 对 String 的支持

字符串池、常量池（运行时常量池、Class 常量池）、intern

答案:

字符串不可变性是因为String类中的关键字是private final char value[];,value是String封装的数组。

JDK6中substring会引发内存泄漏, JDK7中的substring内部new了一个新对象,因此避免了内存泄漏的问题。

ReplaceFirst替换第一次出现的字符，replaceAll替换符合规则的所有字符，replace支持字符的替换也支持字符串的替换。

#### → 熟悉 Java 中各种关键字

transient、instanceof、final、static、volatile、synchronized、const 原理及用法。

答案：Java的transient关键字,只需要实现Serilizable接口,将不需要序列化的属性前添加关键字transient,序列化的时候这个属性就不会被序列化到指定的目的中。

istanceof是java的一个二元操作符，和==，>,<是同一类的用法,是java保留关键字,作用是判断它左边的对象是否是右边的实例，返回布尔类型。

Volatile关键字涉及到并发编程的概念，在并发编程中，我们通常会遇到以下三个问题：原子性问题，可见性问题，有序性问题。

原子性：对基本数据类型的变量的读取和赋值操作是原子性操作，即这些操作是不可被中断的，要么执行，要么不执行。

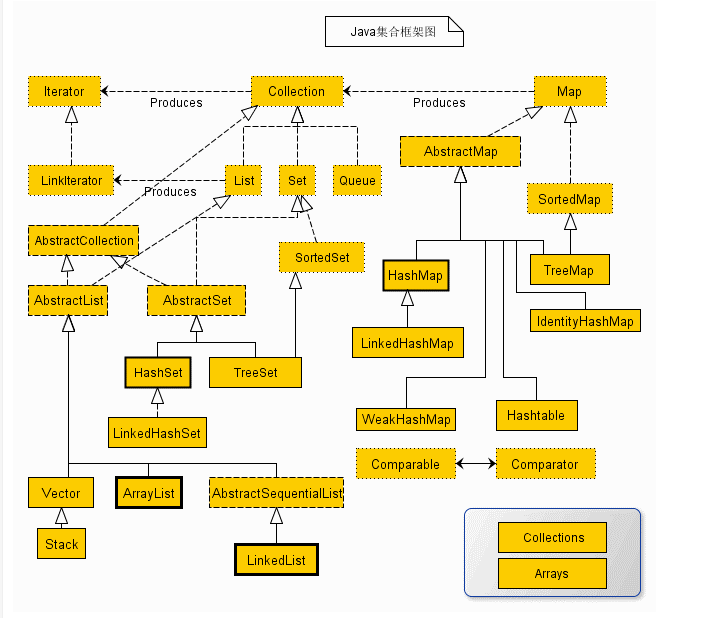
可见性：当一个共享变量被volatile修饰时，它会保证修改的值立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。

有序性：在java内存模型中，允许编译器和处理器对指令进行重排序，但是重排序过程不会影响到单线程程序的执行，却会影响到多线程并发执行的正确性。

Volatile关键字无法保证原子性（其他可以），这时候就引入了synchronized关键字,synchronized关键字是防止多个线程同时执行一段代码，有时候会很影响程序的执行效率。但是volatile关键字无法替代synchronized关键字。

Const是Java预留关键字，用于后期扩展用，用法跟final相似，不常用。

#### → 集合类



常用集合类的使用、ArrayList 和 LinkedList 和 Vector 的区别 、SynchronizedList 和 Vector 的区别、HashMap、HashTable、ConcurrentHashMap 区别、

Set 和 List 区别？Set 如何保证元素不重复？

Java 8 中 stream 相关用法、apache 集合处理工具类的使用、不同版本的 JDK 中 HashMap 的实现的区别以及原因

Collection 和 Collections 区别

Arrays.asList 获得的 List 使用时需要注意什么

Enumeration 和 Iterator 区别

fail-fast 和 fail-safe

CopyOnWriteArrayList、ConcurrentSkipListMap；

遍历集合：http://www.runoob.com/java/java-collections.html

集合类学习资料参考。

#### → 枚举

枚举的用法、枚举的实现、枚举与单例、Enum 类

Java 枚举如何比较

switch 对枚举的支持

枚举的序列化如何实现

枚举的线程安全性问题

#### → IO

字符流、字节流、输入流、输出流、

同步、异步、阻塞、非阻塞、Linux 5 种 IO 模型

BIO、NIO 和 AIO 的区别、三种 IO 的用法与原理、netty

#### → 反射

反射与工厂模式、反射有什么用

Class 类、java.lang.reflect.\*

反射是什么呢？当我们的程序在运行时，需要动态的加载一些类这些类可能之前用不到所以不用加载到jvm，而是在运行时根据需要才加载，这样的好处对于服务器来说不言而喻，举个例子我们的项目底层有时是用mysql，有时用oracle，需要动态地根据实际情况加载驱动类，这个时候反射就有用了，假设 com.java.dbtest.myqlConnection，com.java.dbtest.oracleConnection这两个类我们要用，这时候我们的程序就写得比较动态化，通过Class tc = Class.forName("com.java.dbtest.TestConnection");通过类的全类名让jvm在服务器中找到并加载这个类，而如果是oracle则传入的参数就变成另一个了。

#### → 动态代理

静态代理、动态代理

动态代理和反射的关系

动态代理的几种实现方式

AOP

答案： 实现动态代理有两种方式：1.JDK的动态代理 2.cglib动态代理.

JDK动态代理是由java内部的反射机制来实现的,cglib动态代理底层借助asm依赖库来实现的。

#### → 序列化

什么是序列化与反序列化、为什么序列化、序列化底层原理、序列化与单例模式、protobuf、为什么说序列化并不安全。

Java 序列化是指把 Java 对象转换为字节序列的过程便于保存在内存、文件、数据库中，ObjectOutputStream类的 writeObject() 方法可以实现序列化。反序列化是指把字节序列恢复为 Java 对象的过程，ObjectInputStream 类的 readObject() 方法用于反序列化。

#### ****→ 注解****

元注解、自定义注解、Java 中常用注解使用、注解与反射的结合

Spring 常用注解

答：

一.组件类注解 （4个）

@Commpent：标准一个普通的Spring Bean类。

@Repository：标注一个DAO组件类。

@Service：标注一个业务逻辑组件类。

@Controller：标注一个控制器组件类。

二.装配bean时常用的注解（4个）

@Autowired： 用于类的属性、构造器、方法进行注值。

@Resource： 不属于Spring注解，但是作用于Autowired类似。

@PostConstruct: 方法实现初始化之前的操作。

@PreDestroy： 方法实现销毁bean之前进行的操作。

三．@Component 和 @Configuration 和 @Bean注解。

四．Spring MVC模块注解(四个)

@Controller ：表明该类会作为与前端作交互的控制层组件，通过服务接口定义的提供访问应用程序的一种行为，解释用户的输入，将其转换成一个模型然后将试图呈献给用户。

@RequestMapping ： 这个注解用于将url映射到整个处理类或者特定的处理请求的方法。可以只用通配符！

@RequestParam ：将请求的参数绑定到方法中的参数上，有required参数，默认情况下，required=true，也就是改参数必须要传。如果改参数可以传可不传，可以配置required=false。

@PathVariable ： 该注解用于方法修饰方法参数，会将修饰的方法参数变为可供使用的uri变量（可用于动态绑定）。

#### → JMS

什么是 Java 消息服务、JMS 消息传送模型

#### → JMX

java.lang.management.\*、 javax.management.\*

#### → 泛型

泛型与继承、类型擦除、泛型中 KTVE? object 等的含义、泛型各种用法

限定通配符和非限定通配符、上下界限定符 extends 和 super

List<Object> 和原始类型 List 之间的区别?

List<?> 和 List<Object> 之间的区别是什么?

#### → 单元测试

junit、mock、mockito、内存数据库（h2）

#### → 正则表达式

java.lang.util.regex.\*

#### → 常用的 Java 工具库

commons.lang、commons.\*...、 guava-libraries、 netty

#### → API & SPI

API、API 和 SPI 的关系和区别

如何定义 SPI、SPI 的实现原理

#### → 异常

异常类型、正确处理异常、自定义异常

Error 和 Exception

异常链、try-with-resources

finally 和 return 的执行顺序

#### → 时间处理

时区、冬令时和夏令时、时间戳、Java 中时间 API

格林威治时间、CET,UTC,GMT,CST 几种常见时间的含义和关系

SimpleDateFormat 的线程安全性问题

Java 8 中的时间处理

如何在东八区的计算机上获取美国时间

#### → 编码方式

Unicode、有了 Unicode 为啥还需要 UTF-8

GBK、GB2312、GB18030 之间的区别

UTF8、UTF16、UTF32 区别

URL 编解码、Big Endian 和 Little Endian

如何解决乱码问题

#### → 语法糖

Java 中语法糖原理、解语法糖

语法糖：switch 支持 String 与枚举、泛型、自动装箱与拆箱、方法变长参数、枚举、内部类、条件编译、 断言、数值字面量、for-each、try-with-resource、Lambda 表达式

03 阅读源代码

String、Integer、Long、Enum、

BigDecimal、ThreadLocal、ClassLoader & URLClassLoader、

ArrayList & LinkedList、

HashMap & LinkedHashMap & TreeMap & CouncurrentHashMap、HashSet & LinkedHashSet & TreeSet

04 Java 并发编程

#### ****→ 并发与并行****

什么是并发、什么是并行

并发与并行的区别?

答案:

并发: 一个处理器同时处理多个任务。

并行：多个处理器或者多核的处理器同时处理多个不同的任务。

#### ****→ 什么是线程，与进程的区别？****

线程的实现、线程的状态、优先级、线程调度、创建线程的多种方式、守护线程

线程与进程的区别？

一个程序有至少有一个进程，一个进程至少有一个线程，线程是进程的最小粒度的体现。（互斥锁，内存共享，信号量）

#### → 线程池

自己设计线程池、submit() 和 execute()、线程池原理

为什么不允许使用 Executors 创建线程池？

答案： 阿里开发手册上写着关于线程创建的问题：

【强制】线程池不允许使用Executors去创建，而是通过ThreadPoolExecutor的方式,这样的处理方式让写的同学更加明确线程池的运行规则,规避资源耗尽的风险。

原因： 1）FixedThreadPool和SingleThreadPool方法:

允许的请求队列长度为Integer.MAX\_VALUE,可能会堆积大量的请求,从而导致OOM。

2）CachedThreadPool和ScheduledThreadPool方法：

允许创建的线程数量为Integer.MAX\_VALUE,可能会创建大量的线程,从而导致OOM。（OutOfMemoryError）线程创建使用的不是堆内存,而是实际物理内存。

#### → 线程安全

死锁、死锁如何排查、线程安全和内存模型的关系

答案：线程的三个特性： 原子性，可见性，有序性。

volatile和synchronize关键子；

volatile具有可见性,禁止指令重排,但是不具备原子性特性。

#### → 锁

CAS、乐观锁与悲观锁、数据库相关锁机制、分布式锁、偏向锁、轻量级锁、重量级锁、monitor、

锁优化、锁消除、锁粗化、自旋锁、可重入锁、阻塞锁、死锁。

答案： 乐观锁与悲观锁·

乐观锁一般假设数据不会造成冲突，在数据提交更新的时候，才会正式对数据的冲突进行检测，如果发生冲突，则返回错误信息，让用户自己去处理；实现乐观锁的方式是：1）使用版本号 2）使用时间戳。

悲观锁并发实际上采用“先取锁再访问”的保守策略，为数据处理的安全提供了保证。

#### → 死锁

什么是死锁

死锁如何解决？

答案： 死锁：一个进程集合中的多线程因为竞争资源，而造成互相等待的现象。

原因：1）系统资源不足 2）进程运行推进顺序不合适 3）资源分配不当等。

解决死锁的最简单办法 1）系统重启 2）撤销进程，回收他们占有的资源。

#### → synchronized

synchronized 是如何实现的？

synchronized 和 lock 之间关系、不使用 synchronized 如何实现一个线程安全的单例

synchronized 和原子性、可见性和有序性之间的关系

#### → volatile

happens-before、内存屏障、编译器指令重排和 CPU 指令重

volatile 的实现原理

volatile 和原子性、可见性和有序性之间的关系

有了 symchronized 为什么还需要 volatile

#### ****→ sleep 和 wait****

参考答案:

1. sleep方法让出CPU,不释放锁,到指定时间后CPU再回到指定线程继续往下执行.
2. wait方法是指当前线程暂时退出同步资源锁,只有调用notify/notifyAll唤醒指定线程或所有线程,才会进入线程池,再次参与线程的调度。

#### ****→ wait 和 notify****

#### ****→ notify 和 notifyAll****

#### ****→ ThreadLocal****

#### ****→ 写一个死锁的程序****

#### ****→ 写代码来解决生产者消费者问题****

#### ****→ 并方包****

Thread、Runnable、Callable、ReentrantLock、ReentrantReadWriteLock、Atomic\*、Semaphore、CountDownLatch、ConcurrentHashMap、Executors

***2***

**底层篇**

01JVM

#### → JVM 内存结构

class 文件格式、运行时数据区：堆、栈、方法区、直接内存、运行时常量池、

堆和栈区别

Java 中的对象一定在堆上分配吗？

#### ****→ Java 内存模型****

计算机内存模型、缓存一致性、MESI 协议

可见性、原子性、顺序性、happens-before、

内存屏障、synchronized、volatile、final、锁

#### → 垃圾回收

GC 算法：标记清除、引用计数、复制、标记压缩、分代回收、增量式回收

GC 参数、对象存活的判定、垃圾收集器（CMS、G1、ZGC、Epsilon）

#### → JVM 参数及调优

-Xmx、-Xmn、-Xms、Xss、-XX:SurvivorRatio、

-XX:PermSize、-XX:MaxPermSize、-XX:MaxTenuringThreshold

#### → Java 对象模型

oop-klass、对象头

#### → HotSpot

即时编译器、编译优化

#### → 虚拟机性能监控与故障处理工具

jps, jstack, jmap, jstat, jconsole, jinfo, jhat, javap, btrace, TProfiler

Arthas

02 类加载机制

classLoader、类加载过程、双亲委派（破坏双亲委派）、模块化（jboss modules、osgi、jigsaw）

03 编译与反编译

什么是编译（前端编译、后端编译）、什么是反编译

JIT、JIT 优化（逃逸分析、栈上分配、标量替换、锁优化）

编译工具：javac

反编译工具：javap 、jad 、CRF

***3***

**进阶篇**

01 Java 底层知识

#### ****→ 字节码、class 文件格式****

#### ****→ CPU 缓存，L1，L2，L3 和伪共享****

#### ****→ 尾递归****

#### ****→ 位运算****

用位运算实现加、减、乘、除、取余

02 设计模式

设计模式的六大原则：

开闭原则（Open Close Principle）、里氏代换原则（Liskov Substitution Principle）、依赖倒转原则（Dependence Inversion Principle）

接口隔离原则（Interface Segregation Principle）、迪米特法则（最少知道原则）（Demeter Principle）、合成复用原则（Composite Reuse Principle）

**→ 了解 23 种设计模式**

创建型模式：单例模式、抽象工厂模式、建造者模式、工厂模式、原型模式。

结构型模式：适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。

行为型模式：模版方法模式、命令模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式（Interpreter 模式）、状态模式、策略模式、职责链模式(责任链模式)、访问者模式。

#### ****→ 设计模式****

单例的七种写法：懒汉——线程不安全、懒汉——线程安全、饿汉、饿汉——变种、静态内部类、枚举、双重校验锁

工厂模式、适配器模式、策略模式、模板方法模式、观察者模式、外观模式、代理模式等必会

Spring 中用到了那些设计模式，讲一下自己对于这些设计模式的理解？

参考答案：

代理模式：在AOP中被使用最多，比如jdk动态代理和Cglib动态代理。

适配器模式：在Aop中使用Advice（前置）类增强被代理类的功能，在代理类的方法前，设置拦截器，通过拦截器增强代理方法的功能，实现面向切面编程。

单列模式：在Spring配置文件中定义bean的时候默认的是单例模式。

工厂模式：BeanFactory用来创建对象的实例，Spring下的bean默认为singleton，可以通过singleton=“true|false”指定。 （暂时这么多）

#### ****→**** ****不用 synchronized 和 lock，实现线程安全的单例模式？****

#### ****→ 实现 AOP****

#### ****→ 实现 IOC****

#### ****→ nio 和 reactor 设计模式、****

参考答案：nio叫做非阻塞io，Nio有Selector类，类似于一个观察者，只要我们把需要探知的socketchannel告诉Selector,我们接着做别的事情,当有事件发生的时候,它就会通知我们,传回一组SelectionKey,我们读取这些Key,就会获得我们刚刚注册过的socketchannel,然后读取数据,并处理数据。

03 网络编程知识

**→ tcp、udp、http、https 等常用协议**

三次握手与四次关闭、流量控制和拥塞控制、OSI 七层模型、tcp 粘包与拆包

#### ****→ http/1.0 http/1.1 http/2 之前的区别****

http 中 get 和 post 区别

常见的 web 请求返回的状态码

404、302、301、500分别代表什么

#### ****→ http/3****

#### ****→ Java RMI，Socket，HttpClient****

#### ****→ cookie 与 session****

cookie 被禁用，如何实现 session

#### ****→ 用 Java 写一个简单的静态文件的 HTTP 服务器****

#### ****→ 了解 nginx 和 apache 服务器的特性并搭建一个对应的服务器****

#### ****→ 用 Java 实现 FTP、SMTP 协议****

#### ****→ 进程间通讯的方式****

#### ****→ 什么是 CDN？如果实现？****

#### ****→ DNS****

#### 什么是 DNS 、记录类型: A 记录、CNAME 记录、AAAA 记录等

域名解析、根域名服务器

DNS 污染、DNS 劫持、公共 DNS：114 DNS、Google DNS、OpenDNS

#### ****→ 反向代理****

正向代理、反向代理

反向代理服务器

04 框架知识

**→ Servlet**

生命周期

线程安全问题

filter 和 listener

web.xml 中常用配置及作用

#### ****→ Hibernate****

什么是 OR Mapping？

参考答案： OR Mapping就是对象关系映射,把对象映射到数据库表,把字段映射数据库的列。

#### Hibernate 的懒加载

Hibernate 的缓存机制

参考答案：

Hibernate一级缓存是Session缓存；Hibernate二级缓存是SessionoFactory级别的缓存。

Hibernate / Ibatis / MyBatis 之间的区别？

参考答案：

Mybatis的优势：

Mybatis可以进行更为细致的SQL优化，可以减少查询字段。

Mybatis容易掌握，而Hibernate门槛高。

Hibernate的优势：

Hibernate的DAO层来发比Mybatis简单,Mybatis需要维护SQL和结果映射表。

Hibernate对对象的维护和缓存要比Mybatis好，对增删改查的对象的维护要方便。

Hibernate数据库移植性很好，Mybatis的数据库移植性差，不同的数据库需要些不同的SQL。

Hibernate有更好的二级缓存机制，可以使用第三方缓存。Mybatis本身提供的缓存机制不佳。

#### ****→ Spring****

#### Bean 的初始化

#### AOP 原理

#### 实现 Spring 的IOC

#### Spring 四种依赖注入方式

#### ****→ Spring MVC****

什么是 MVC

Spring mvc 与 Struts mvc 的区别

#### ****→ Spring Boot****

Spring Boot 2.0、起步依赖、自动配置、

Spring Boot 的 starter 原理，自己实现一个 starter

#### ****→ Spring Security****

### ****→ Spring Cloud****

服务发现与注册：Eureka、Zookeeper、Consul

负载均衡：Feign、Spring Cloud Loadbalance

服务配置：Spring Cloud Config

服务限流与熔断：Hystrix

服务链路追踪：Dapper

服务网关、安全、消息

05 应用服务器知识

**→ JBoss**

#### ****→ tomcat****

#### ****→ jetty****

#### ****→ Weblogic****

06 工具

**→ git & svn**

#### ****→ maven & gradle****

#### ****→ Intellij IDEA****

常用插件：Maven Helper 、FindBugs-IDEA、阿里巴巴代码规约检测、GsonFormat

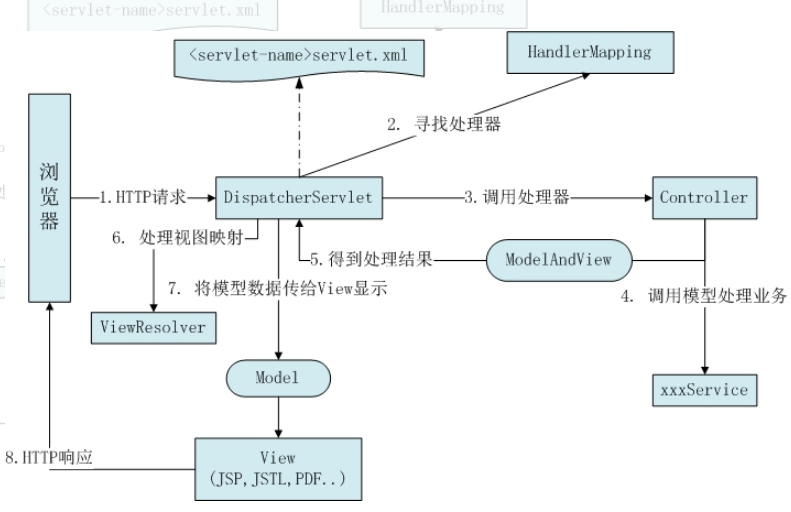
Lombok plugin、.ignore、Mybatis plugin

## ***6*** 广州面试题 2019-3-3

SpringMVC 相关面试题

https://www.jianshu.com/p/ce85b9e7d650

https://blog.csdn.net/cswhale/article/details/16941281



1. 什么是SpringMVC?

参考答案:

Spring是一个模块,基于MVC的一个框架。MVC是一种设计模式。M代表

Model; V代表view; C代表Controller。M指的是模型, 即dao层和service层,V指的是视图, 即jsp,html等。C指的是Controller, 即控制器。

1. SpringMVC的工作原理是什么？

参考答案:

1. **客户端请求提交到DispatcherServlet。**
2. **由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller。**
3. **DispatcherServlet将请求提交到Controller**
4. **Controller调用业务逻辑处理后，返回ModelAndView**
5. **DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图。**
6. **视图负责将结果显示到客户端。**
7. SpringMVC的优缺点？

答：1.方便解耦，简化开发,职责鲜明，分工明确。

        2.AOP的很好支持，方便面向切面编程。

        3.对主流的框架提供了很好的集成支持。

        4.Spring属于低侵入，代码污染极低。

说一下自己对 IOC 、AOP 的理解？ （ok）

Spring 中用到了那些设计模式，讲一下自己对于这些设计模式的理解（ok）

Spring Bean 的作用域和生命周期了解吗？（ok）

参考答案：

Spring 事务中的隔离级别？

参考答案：有四种事务隔离级别：

read\_uncommitted: 完全没有隔离，会出现脏读、不可重复读和幻读。

read\_committed: 简单隔离，不会出现脏读。

repeatable\_read: 中度隔离，不会出现脏读、不可重复读。

serializable: 完全隔离，不会出现脏读、不可重复读和幻读。

Spring 事务中的事务传播行为？

参考答案：

required: 如果当前有事务，就使用当前的事务，如果没有，就开启新事务。

supports: 如果当前有事务，就使用当前事务，如果没有，则不开启事务。

requires\_new: 如果当前有事务，则挂起当前事务，并开启新事务；否则直接开启新事务。

not\_supported: 如果当前有事务，则挂起当前事务。

nested：如果当前有事务，则开启一个嵌套事务；如果当前没有事务，开启新事务。

mandatory: 如果当前没有事务，则抛出异常。

never：如果当前有事务，则抛出异常。

手写一个 LRU 算法

知道那些排序算法，简单介绍一下快排的原理，能不能手写一下快排

参考答案：

 快速排序（Quicksort）是对冒泡排序的一种改进。由C. A. R. Hoare在1962年提出。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

String 为什么是不可变的？String为啥要设计为不可变的？（ok）

Arraylist 与 LinkedList 异同 （ok）

HashMap的底层实现？（github）

参考答案：

JDK1.8之前HashMap底层实现是数组和链表结合在一起使用，也就是链表散列。

JDK1.8之后，当链表长度大于阈值（默认8）时，将链表转化为红黑树，以减少搜索时间。

HashMap 的长度为什么是2的幂次方？

参考答案：为了能让HashMap存取高效，尽量减少碰撞，也就是尽量把数据分配均匀。

ConcurrentHashMap 和 Hashtable 的区别？

参考答案：

1）JDK1.7的时候ConcurrentHashMap对整个桶数组进行了分割分段（Segment），每把锁只锁容器其中一部分数据，多线程访问容器里不同数据段的数据，就不会存在锁竞争，提高并发访问率。JDK1.8的时候，直接使用Node数组+链表+红黑树的数据结构来实现，并发控制使用synchronized和CAS来操作。

2）Hashtable使用synchronized来保证线程安全，效率非常低下。

ConcurrentHashMap线程安全的具体实现方式/底层具体实现？

参考答案：

JDK1.7

首先将数据分为一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据时，其他段的数据也能被其他线程访问。

**ConcurrentHashMap 是由 Segment 数组结构和 HashEntry 数组结构组成**。

Segment 实现了 ReentrantLock,所以 Segment 是一种可重入锁，扮演锁的角色。HashEntry 用于存储键值对数据。

static class Segment<K,V> extends ReentrantLock implements Serializable {

}

一个 ConcurrentHashMap 里包含一个 Segment 数组。Segment 的结构和HashMap类似，是一种数组和链表结构，一个 Segment 包含一个 HashEntry 数组，每个 HashEntry 是一个链表结构的元素，每个 Segment 守护着一个HashEntry数组里的元素，当对 HashEntry 数组的数据进行修改时，必须首先获得对应的 Segment的锁。

JDK1.8

ConcurrentHashMap取消了Segment分段锁，采用CAS和synchronized来保证并发安全。数据结构跟HashMap1.8的结构类似，数组+链表/红黑二叉树。

synchronized只锁定当前链表或红黑二叉树的首节点，这样只要hash不冲突，就不会产生并发，效率又提升N倍。