1、什么是 Spring?

Spring 是一个轻量级的 DI / IoC 和 AOP 容器的开源框架，用于简化 java 开发。

2、Spring 中的两个理念

IOC（控制反转）：将原本在程序中手动创建对象的控制权，交由 Spring 框架来管理；
正控：若要使用某个对象，需要自己去负责对象的创建；
反控：若要使用某个对象，只需要从 Spring 容器中获取需要使用的对象，不关心对象的创建过程，也就是把创建对象的控制权反转给了 Spring 框架；

AOP（面向切面编程）：核心业务功能和切面功能分别独立进行开发，然后把切面功能和核心业务功能 "编织" 在一起

3、什么是 DI

注入依赖，通过配置把依赖属性注入到对象。

4、Spring 中 Ioc 的设计

BeanFactory（最低层的接口）
ApplicationContext（继承 BeanFactory，其还有很多子类）

5、Spring 中 Bean 的实例方式

（1）类构造器实例；
（2）使用静态工厂实例；
（3）使用实例工厂实例；

6、Spring 中 Bean 的生命周期及作用域

（1）Bean 的定义；
（2）Bean 的初始化；
（3）Bean 的调用；
（4）Bean 的销毁；
作用域：singleton、prototype、request、session、global session

7、Bean 的注入方式

（1）接口注入；
（2）构造器注入；
（3）Set 方法注入；

8、Spring 的事务管理

对一系列的数据进行统一提交或回滚操作，如果插入成功则一起成功，如果其中一条出现异常则全部回滚；

9、Spring 是如何处理线程并发问题

使用 ThrealLocal 解决线程安全问题，每个线程中有独立的变量副本，从而隔离多线程对数据的访问冲突；

10、AOP 中的几个名词的解析

切面：一个关注点的模块化，使用@Aspect 注解；
连接点：程序执行中某个特定的点；
切点：匹配连接点的断言；
通知：切面的某个特定连接点上执行的动作；
目标对象：被一个或多个切面通知的对象；
织入：把切面连接到通知对象上；

11、AOP 中通知的类型

前置通知、后置通知、结果返回通知、环绕通知、异常抛出后通知

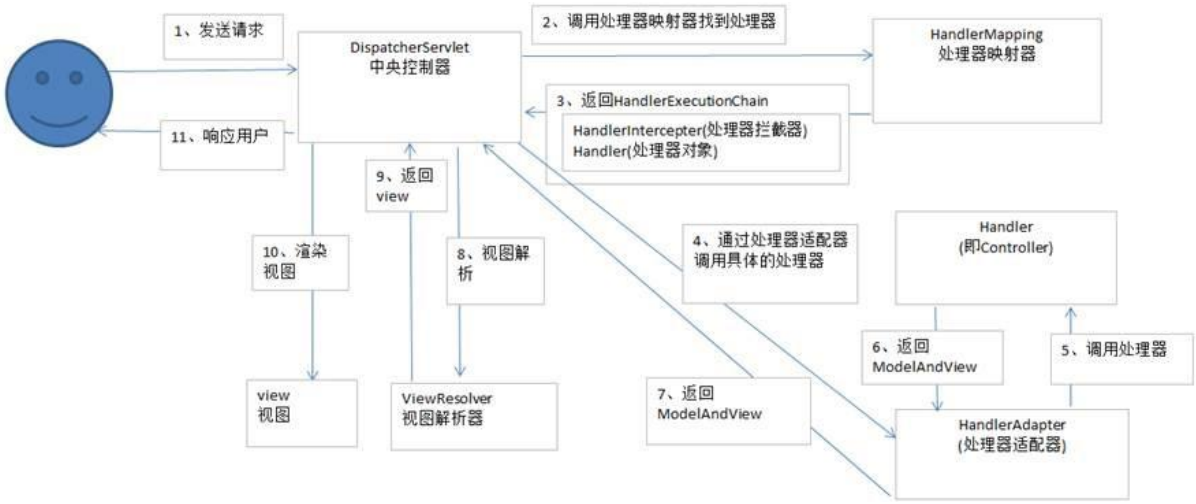
12、Spring 中的核心类

BeanFactory：提供 bean 的获取；
BeanWrapper：提供统一的 Get 和 Set 方法；

13、什么是 SpringMVC

Web 层的视图控制框架，用于替代 servlet 的

14、SpringMVC 的工作流程



(1) 用户发请求至前端控制器 DispatcherServlet；
(2) DispatcherServlet 调用映射处理器 HadlerMapping 进行请求映射，返回映射对象；
(3) DispatcherServlet 调用 HandlerAdapter，根据映射对象匹配对应的 Controller 进行业务处理并返回 ModelAndView；
(4) DispatcherServlet 调用视图解析器对 ModelAndView 进行解析、渲染，生成视图 View，最后响应给用户。

15、SpringMVC 与 Struct2 的区别

（1）入口不同，mvc 是 Servlet，Struct2 是 Filter；
（2）生命周期不同，mvc 的 Controller 是单例，Struct2 的 Action 是多例；

16、SpringMVC 如何实现 RESTFul 服务
（1）引入 json 数据格式的处理包；
（2）开启注解驱动<mvc:annotation-dirven />；
（3）使用@RequestBody、@@ResponseBody 注解；

17、SpringMVC 如何定义过滤器、拦截器
拦截器：实现 HandleInterceptor，并重写 preHandler、postHandler、afterCompletion 方法，然后把该类配置到 Spring 中，并设置拦截路径；（WebMvcConfigurerAdapter）
过滤器：实现 Filter，重写 doFilter 方法，然后注册到 Spring 中，用@WebFilter 的注解，

18、Spring 整合 Springmvc（这个有毒，被问过）
在 web.Xml 中配置 dispatcherServlet，并载入 spring-mvc 的配置文件

19、Spring 中的设计模式（9 个）
单例模式
简单工厂模式
工厂模式
代理模式
适配器模式
装饰者模式
观察这模式
策略模式
模板模式

20、AOP 用的是什么模式
代理模式（JDK 动态代理、CGLIB 代理）

21、Hibernate 的三种状态
瞬时状态：没有持久化 OID，未与 Hibernate Session 关联对象；
游离状态：有持久化 OID，但未与当前 Hibernate Session 关联；
持久状态：有持久化 OID，与当前 Hibernate Session 关联且 Session 未关闭；

22、Hibernate 中 HQL 与 SQL 区别
HQL：面向对象
SQL：面向数据表

23、Hibernate 的延迟加载
设置 lazy 属性（true/false）
当真正需要数据的时候才真正执行数据加载操作（实体对象延迟加载、集合延迟

加载、属性延迟加载）

24、Hibernate 的缓存机制

一级缓存：默认开启，基于 session，session 关闭时缓存会自动清楚；
二级缓存：默认不开启，基于应用程序，所有 session 都可以使用（在 hibernate.cfg.xml 需要开启缓存、开启查询缓存、配置缓存框架、指定需缓存的类）

25、Hibernate 的三种查询方式

SQL
HQL
Criteria

26、Hibernate 中状态转化

瞬时 -> 持久　保存、更新
瞬时 -> 游离　对象设置 ID

持久 -> 瞬时　删除操作
持久 -> 游离　 evict()、close()、clear()

游离 -> 瞬时 对象 ID 设置为 null
游离 -> 持久 保存、更新

27、Hibernate 的核心类

Configration（加载配置）
SessionFactory（获取 session）
Session(数据操作)
Query（查询对象，写 SQL 的）
Transation（事务操作:commit，rollback）
Criteria（多条件查询）

28、Hibernate 的工作原理

（1）读取并解析配置文件
（2）读取并解析映射信息，创建 SessionFactory;
（3）打开 Session
（4）创建 Transation
（5）持久化操作
（6）提交事务
（7）关闭 Session
（8）关闭 SessionFactory

29、Hibernate 中 sorted collection 与 ordered collection 区别

sorted collection：在内存中通过 java 比较器进行排序
ordered collection：在数据库中通过 order by 进行排序

30、如何优化 Hibernate

（1）优化数据库设计
（2）HQL 优化
（3）API 的正确使用
（4）主配置参数（日志、查询缓存、fetch_size、batch_size 等）
（5）映射文件优化（ID 生成策略、二级缓存、延迟加载、关联优化）
（6）缓存管理

31、JDBC、Hibernate、Mybatis 区别

JDBC

- 手动 SQL
- delete、update、insert 不能直接传入对象，需把值一个个拼接到 sql
- select 返回的是 resultSet，要手动封装对象

Mybatis

- 手动 SQL
- delete、update、insert 能直接传入对象
- select 返回的对象

Hibernate

- 不用写 SQL，自动封装
- delete、update、insert 能直接传入对象
- select 返回的对象

32、Hibernate 中一些关键字

inverse：表关系控制反转，默认 false，一般在 one 的一方设置
cascade：级联操作，一个表改变另一个也改变，none、all、save-up、delete
fetch：懒加载，select、join、subselect
batch-size:select 是一次抓取一条数据，而 subselect 是抓取所有的记录。而 batch-size 是一个折中点。它可以设置抓取的记录的大小。类似于批量的操作

33、Hibernate 解决并发出现的数据库读取问题

（1）悲观锁：锁数据库；
做法 1：SQL 使用 forupdate
做法 2：Hibernate 的加锁模式有：
LockMode.NONE ： 无锁机制。
LockMode.WRITE ： Hibernate 在 Insert 和 Update 记录的时候会自动获取。
LockMode.READ ： Hibernate 在读取记录的时候会自动获取。
以上这三种锁机制一般由 Hibernate 内部使用，如 Hibernate 为了保证 Update 过程中对象不会被外界修改，会在 save 方法实现中自动为目标对象加上 WRITE 锁。
LockMode.UPGRADE ： 利用数据库的 for update 子句加锁。
LockMode. UPGRADE_NOWAIT ： Oracle 的特定实现，利用 Oracle 的 for update nowait 子句实现加锁。

上面这两种锁机制是我们在应用层较为常用的，加锁一般通过以下方法实现：
Criteria.setLockMode
Query.setLockMode
Session.lock
（2）乐观锁：锁表
大多是基于数据版本（Version）记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个"version"字段来实现。
乐观锁的工作原理：读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。
Hibernate 为乐观锁提供了三 种实现：
1.基于 version----------最常用
2.基于 timestamp----------较常用
3.为遗留项目添加添加乐观锁------不常用
基于 version
做法一：映射文件
<versionname="version" column="VERSION" type="integer" />
做法二：注解
在对应的 version 字段用注解@version，即可
基于 timestamp
<timestamp name="version" column="version" />

34、Hibernate 的事务类型
本地事务：数据库的本地且限于单个线程内的事务，单一数据源；
全局事务：资源管理器管理和协调的事务，可以跨越多个数据库和进程
全局事务是二次提交协议，第一次是预提交，当提交返回都为 true 是才会执行第二次提交，这次才会发生数据变动。
实现：
1.如果使用的是本地事务（JDBC 事务）
<property name="hibernate.current_session_context_class">thread</property>
2. 如果使用的是全局事务（JTA 事务）
<property name="hibernate.current_session_context_class">jta</property>

35、Mybatis 中#与$的区别
${}:属于静态文本替换
#{}：属于预编译的，Mybatis 会把它变成？，然后按序给 sql 的？占位符设值

36、Mybatis 中的标签
常用：<select | insert | update | delete />
关于参数集：<resultMap /> 、<result />、<sql />、<include />、<collection />、<association />
9 个动态标签：<if />、<trim />、<where />、<foreach />、<choose />、

<ortherwise />、<set />、<bind />、<when />

37、Mybatis 的工作原理

Mybatis 是 JDK 动态代理的，运行时为 DAO 层接口生成代理对象，代理对象会拦截 DAO 层接口的方法，转而执行 MappedStatement 所代表的 SQL，然后再将 SQl 执行结果返回

38、Mybatis 的 DAO 层方法可以重载吗？

不行,因为是类的权限名+方法名的保存和寻找策略

39、Mybatis 工作流程

（1）读取配置文件,配置文件右数据库连接信息和 Mapper 映射文件或者 Mapper 包

（2）有了这些信息就能创建 SqlSessionFactorySqlSessionFactory 的生命周期是程序级,程序运行的时候建立起来,程序结束的时候消亡

（3）SqlSessionFactory 建立 SqlSession,目的执行 sql 语句，SqlSession 是过程级,一个方法中建立,方法结束应该关闭

（4）当用户使用 mapper.xml 文件中配置的的方法时，mybatis 首先会解析 sql 动态标签为对应数据库 sql 语句的形式，并将其封装进 MapperStatement 对象，然后通过 executor 将 sql 注入数据库执行，并返回结果。

（5）将返回的结果通过映射，包装成 java 对象

40、SpringBoot 是什么

SpringBoot 是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手

41、SpringBoot 的核心配置文件

application.yml: SpringBoot 自动化的配置

bootstrap.yml：属性不可被覆盖，使用 SpringCloud 的 config 时需要用它

42、SpringBoot 的核心注解

@SpringBootApplication：启动类注解

@EnableAutoConfigration：自动配置

@ComponentScan：组件扫描

@SpringBootConfigration：配置文件

43、如何理解 starters

启动器的意思，简化管理依赖的引入

44、如何在 SpringBoot 启动时运行指定代码

- 实现 ApplicationRunner 接口
- 实现 CommandLineRunner 接口
- @PostStruct 注解

45、SpringBoot 实现不同环境的配置

配置 spring.profiles.active

46、SpringBoot 配置的加载顺序

(1) properties
(2) yml 文件
(3) 系统参数
(4) 运行指令参数

47、SpringBoot 如何兼容旧项目的配置

使用@ImportResource 注解

48、SpringBoot2.x 与 1.x 的区别

(1) jdk 版本升级
(2) 第三方类库更新
(3) 配置属性绑定
(4) 响应式 Spring 编程
(5) Http2.0 版本的支持

49、SpringBoot 项目的保护策略

(1) Https 协议
(2) 开启跨域保护
(3) 使用内容安全策略防止 xss 攻击
(4) SpringBoot 版本的升级

50、SpringBoot 的热部署

引入 spring-boot-devtools 依赖，然后把 idea 的运行策略改为热更新

51、SpringBoot 的执行流程

(1) 启动计时器；
(2) 实例化 SpringBootListeners，调用 starting()启动监听；
(3) 创建 Enviroment，并加载到监听其中；
(4) 创建 Banner，并加载到监听其中；
(5) 创建 Argument，并加载到监听器中；
(6) 创将 ConfigrationApliicationContext；
(7) 把 Bean 加载到 ConfigrationApliicationContext；
(8) 通过 prepareContext 把 listeners、bannner、enviroment、argument 的实例加载到上下文中；
(9) 调用 refresh()刷新上下文；
(10) 执行自定义的 ApplicationRuner 和 CommandLineRunner 的代码
(11) 调用 finished（）结束监听；
(12) 结束计时器；

52、SpringBoot 全局异常拦截

@ControllerAdvice+@ExceptionHandler

53、jar 与 war 区别

jar：开发时要引用的依赖，只包含 class 文件
war：web 应用程序，可直接运行 web 模块

54、多线程的应用场景

(1) 后台任务，例如定时备份文件、长时间的业务执行
(2) 异步处理，例如记录日志
(3) 分布式计算

55、线程、进程的区别

线程：CPU 调度和分派的基本单位
进程：系统调度和分派的基本单位，一个进程可以有多个线程

56、实现多线程的方式

(1) 继承 Thread 类
(2) 实现 Runnable 接口
(3) 线程池 ExecutorService

57、继承 Thread 与实现 Runnable 的区别

(1) 类是单继承，接口可以实现多个接口；
(2) 实现 Runnable 适合多个线程共享代码；
(3) 实现 Runnable 可以增强程序的健壮性，代码共享，数据独立；

58、为什么要用线程池

控制线程数量，减少线程创建和销毁的开销

59、线程池参数

(1) 核心线程数 corePoolSize
(2) 最大线程数 maxPoolSize
(3) 最大空闲时间 keepAliveTime
(4) 任务队列 workQueue
(5) 线程工厂 threadFactory
(6) 拒绝策略 rejectExecutionHandle

60、线程的的状态

运行、等待、阻塞、死亡

61、start()与 run()

start：线程的启动
run：线程的业务代码的执行

62、实现线程安全的方式

- 同步锁：Synchronized
- Lock，方法：lock（）、unlock（）、interrupt（）

63、Synchornzied 与 Lock 区别

Synchronized 是关键字，不可中断等待，出现异常会自动释放锁，资源竞争不激烈时性能优；
Lock 是类，可中断等待，需要手动在 finally{}代码块中释放，资源竞争性能优；

64、volatile 关键字的作用

确保可见性和一致性
当我们使用 volatile 关键字去修饰变量的时候，所以线程都会直接读取该变量并且不缓存它。这就确保了线程读取到的变量是同内存中是一致的。

65、与 Lock 相关的类、接口

ReentranLock（重入锁，类）
ReadWriteLock（读写锁，接口）

66、什么是线程安全

多线程环境下执行的结果与单线程一直，可视为线程安全

67、乐观锁和悲观锁

悲观锁：每次拿数据时都认为会修改，每次都会上锁
乐观锁：每次拿数据时不认为会修改，所以不上锁，只有当更新操作是去获取最新版本的数据之后再做修改操作

68、多线程共享数据

- 如果执行的代码块相同，则可以在 Runnable 对象中设置共享数据；
- 如果执行的代码块不相同，这可以把共享数据设计封装成对象，里面包含线程对数据的操作方法，然后把这个对象作为参数传递给 Runnable 对象

69、死锁

两个或以上的线程竞争同一资源而出现的相互等待的场景

70、死锁发生的条件

(1) 互斥条件：一个资源只能同时被一个线程使用
(2) 请求与保持条件：一个进程因请求资源而阻塞，对已获得的资源不释放
(3) 不剥夺条件：进程获取资源后只有在使用结束后释放，不能强行剥夺
(4) 循环等待条件：若干线程形成环形等待资源关系

71、活锁

两个或以上的线程因状态更改方向一样导致不能进行下去，例如，走廊两个人相互避让但方向一致，导致都不能通过，相对于死锁，就是状态可变

72、避免死锁的算法（银行家算法）

银行系统的现金贷款而出名。一个银行家向一群客户发放信用卡，每个客户有不同的信用额度。每个客户可以提出信用额度内的任意额度的请求，直到额度用完后再一次性还款。银行家承诺每个客户最终都能获得自己需要的额度。所谓“最终”，是说银行家可以先挂起某个额度请求较大的客户的请求，优先满足小额度的请求，等小额度的请求还款后，再处理挂起的请求。这样，资金能够永远流通。所以银行家算法其核心是：保证银行家系统的资源数至少不小于一个客户的所需要的资源数

73、sleep（）与 wait（）区别

sleep（）不会释放锁，会自动唤醒

wait（）会释放锁，需要调用 notify（）唤醒

74、如何检车线程是否有锁

Thread 中有个 holdsLock(Object o)的方法，返回 true 则有锁

75、工作队列

ArrayBlockingQueue：基于数据的阻塞队列

LinkedBlockingQueue：基于链表的阻塞队列

SynchronoursQueue：同步队列

PriorityBlockingQueue：给予优先级阻塞队列

DelayBlockingQueue：延时队列

76、拒绝策略

AbortPolicy：默认，拒绝任务并抛出异常

CallerRunsPolicy：拒绝并在调用者的线程中直接执行线程的 run 方法

DiscardPolicy：拒绝任务，不做任何动作

DiscardOldestPolicy：丢弃队列中第一个任务，并把任务添加到队尾

77、JVM 的内存区域

- 方法区：线程共享的，存储类的信息、常量、静态变量等数据
- Java 堆：线程共享的，存储对象实例信息，垃圾回收的主要区域
- Java 虚拟机栈：线程私有的，每个方法执行时都会创建一个栈帧用于存放局局部变量表（基本数据类型和对象引用）、操作帧（中间操作的工作区指令）、帧数据（方法的所有符号、catch 块的信息）
- 本地方法栈：与 java 虚拟机栈一样，只是服务对象是 Native 方法
- 程序计数器：线程私有的，存放着线程下一执行指令

78、JVM 的内存模型（JMM）

主内存+工作内存

主内存中存储着所有变量

工作线程都有各自的工作内存，会从主内存中复制到各自的内存中，然后进行读写操作

79、JVM 的运行堆

分为 3 大区：新生代、年老代、永久代
新生代分为：1 个 Eden 区和两个 Survivor 区
堆大小调整：
-Xms　堆的最小值
-Xmx　堆的最大值
新生代调整：
-XX:newSize　新生代最小值
-XX:maxNewSize　新生代最大值
-XX:newRation　新生代与年老代的比例
-XX:survivorRation　新生代中 Eden 与 Survivor 的比例
其他
-XX:MaxTenuringThreshold　GC 年龄

80、Minor GC（新生代回收）

- 内存比例 Eden：S0：S1 = 8：1：1
- 新对象生成并在 Eden 区申请内存失败时触发
- Eden 中经过 GC 后存活下来的对象会进入其中一个 Survivor 中，另一个 Survivor 充当备份区；当 Survivor0 饱满之后，就会对 Eden 和 SurvivorA 进行 GC，并把存活的对象复制到 Survivor1，就这样循环，S0 与 S1 之间相互复制
- 每次 GC，对象年龄+1，当到达 15（默认）之后会进入年老代，或者 Survivor 区饱和，此时，该区中相同年龄段的对象大小总和超过 survivor 的一般，则大于或等于该年龄的对象进入年老代

81、Full GC

- 整个堆内存清理，包括年轻代、年老代、永久代
- 触发条件：（1）年老代被写满；（2）永久代被写满；（3）System.gc()显示调用；（4）上一次 GC 后堆的各区分配策略动态变化

82、GC 的判定方法

- 引用计数（jdk1.2 之前）：当对象有被引用是，计数+1，引用销毁时计数-1，当对象的引用计数为 0 时，可以被回收
- 根搜素算法：从一个 GC Root 节点开始，查找对应的节点，依次向下查找，当所有引用节点遍历完毕后，剩余的节点视为无用节点，可以被回收

83、对象的引用类型

- 强引用：只要引用存在，GC 就不回收；内存溢出时，才会进行第二次回收
- 软引用：内存溢出时，才会进行第二次回收
- 弱引用：生存到下一垃圾回收时
- 虚引用：无法找到对象，垃圾回收时回收

84、GC 回收算法

- 复制算法：新生代的内存被划分为一块较大的 Eden 空间和两块较小的

Survivor 空间，每次使用 Eden 和其中一块 Survivor。每次回收时，将 Eden 和 Survivor 中还存活着的对象一次性复制到另外一块 Survivor 空间上，最后清理掉 Eden 和刚才用过的 Survivor 空间（适用存活对象少，不造成内存碎片）

- 标记-清理算法：从根节点进行扫描，对存活的对象进行标记，标记完毕后，对整个空间没有标记的对象进行回收（造成内存碎片）
- 标记-整理算法：与标记-清理算法一样进行标记处理，但在清除时，对存活的对象向左端空闲空间移动，并更新相应的指针（解决标记-清除算法的内存碎片问题）

85、GC 回收器

- 串行垃圾回收器（Serial）（新生代：复制算法；年老代：标记-整理算法）
- 并行垃圾回收器（Parallel）（新生代：复制算法；年老代：标记-整理算法）
-
- 并发标记扫描垃圾回收器（Scavenge|CMS）（新生代：复制算法；年老代：标记-清除算法）
-
- G1 垃圾回收器（G1 GarbageCollector）（不分代，标记-整理算法）

86、时间复杂度含义

我们知道常数项对函数的增长速度影响并不大，所以当 T(n) = c，c 为一个常数的时候，我们说这个算法的时间复杂度为 O(1)；如果 T(n) 不等于一个常数项时，直接将常数项省略。

比如

第一个 Hello, World 的例子中 T(n) = 2，所以我们说那个函数(算法)的时间复杂度为 O(1)。

T(n) = n + 29，此时时间复杂度为 O(n)。

我们知道高次项对于函数的增长速度的影响是最大的。n^3 的增长速度是远超 n^2 的，同时 n^2 的增长速度是远超 n 的。 同时因为要求的精度不高，所以我们直接忽略低此项。

比如

T(n) = n^3 + n^2 + 29，此时时间复杂度为 O(n^3)。

因为函数的阶数对函数的增长速度的影响是最显著的，所以我们忽略与最高阶相乘的常数。

比如

T(n) = 3n^3，此时时间复杂度为 O(n^3)。

综合起来：如果一个算法的执行次数是 T(n)，那么只保留最高次项，同时忽略最高项的系数后得到函数 f(n)，此时算法的时间复杂度就是 O(f(n))。为了方便描述，下文称此为 大 O 推导法。

87、排序算法

- 冒泡排序

冒泡排序算法的算法过程如下：

①. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。

②. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

③. 针对所有的元素重复以上的步骤，除了最后一个。

④. 持续每次对越来越少的元素重复上面的步骤①~③，直到没有任何一对数字需要比较。

- 快速排序

快速排序使用分治策略来把一个序列（list）分为两个子序列（sub-lists）。步骤为：

①. 从数列中挑出一个元素，称为"基准"（pivot）。

②. 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。

③. 递归地（recursively）把小于基准值元素的子数列和大于基准值元素的子数列排序。

①. 挖坑法 用伪代码描述如下：

（1）low = L; high = R; 将基准数挖出形成第一个坑a[low]。

（2）high--，由后向前找比它小的数，找到后挖出此数填前一个坑a[low]中。

（3）low++，由前向后找比它大的数，找到后也挖出此数填到前一个坑a[high]中。

（4）再重复执行②，③二步，直到low==high，将基准数填入a[low]中。

- 插入排序

一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下：

①. 从第一个元素开始，该元素可以认为已经被排序

②. 取出下一个元素，在已经排序的元素序列中从后向前扫描

③. 如果该元素（已排序）大于新元素，将该元素移到下一位置

④. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置

⑤. 将新元素插入到该位置后

⑥. 重复步骤②~⑤

- 希尔排序

①. 选择一个增量序列t1，t2，…，tk，其中ti>tj，tk=1；（一般初次取数组半长，之后每次再减半，直到增量为1）

②. 按增量序列个数k，对序列进行k 趟排序；

③. 每趟排序，根据对应的增量ti，将待排序列分割成若干长度为m 的子序列，分别对各子表进行直接插入排序。仅增量因子为1 时，整个序列作为一个表来处理，表长度即为整个序列的长度

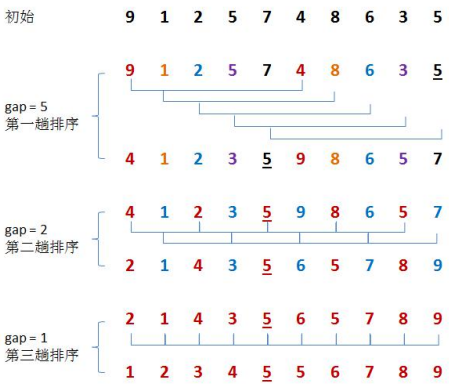


图-希尔排序示例图

在上面这幅图中：初始时，有一个大小为 10 的无序序列。

在第一趟排序中，我们不妨设 gap1 = N / 2 = 5，即相隔距离为 5 的元素组成一组，可以分为 5 组。

接下来，按照直接插入排序的方法对每个组进行排序。

在第二趟排序中，我们把上次的 gap 缩小 一半，即 gap2 = gap1 / 2 = 2 (取整数)。这样每相隔距离为 2 的元素组成一组，可以分为 2 组。

按照直接插入排序的方法对每个组进行排序。

在第三趟排序中，再次把 gap 缩小一半，即gap3 = gap2 / 2 = 1。 这样相隔距离为 1 的元素组成一组，即只有一组。 按照直接插入排序的方法对每个组进行排序。此时，排序已经结束。

需要注意一下的是，图中有两个相等数值的元素 5 和 5 。我们可以清楚的看到，在排序过程中，两个元素位置交换了。 所以，希尔排序是不稳定的算法。

- 选择排序

①. 从待排序序列中，找到关键字最小的元素；

②. 如果最小元素不是待排序序列的第一个元素，将其和第一个元素互换；

③. 从余下的 N – 1 个元素中，找出关键字最小的元素，重复①、②步，直到排序结束。

- 并归排序

采用递归法：①. 将序列每相邻两个数字进行归并操作，形成 floor(n/2)个序列，排序后每个序列包含两个元素；

②. 将上述序列再次归并，形成 floor(n/4)个序列，每个序列包含四个元素；

③. 重复步骤②，直到所有元素排序完毕

- 堆排序

将待排序序列构造成一个大顶堆，此时，整个序列的最大值就是堆顶的根节点。将其与末尾元素进行交换，此时末尾就为最大值。然后将剩余 n-1 个元素重新构造成一个堆，这样会得到 n 个元素的次小值。如此反复执行，便能得到一个有序序列了