## 基础知识：

### ◆abstract class和interface有什么区别?

第一：语法区别：

接口中可以定义，抽象方法，默认方法（default） 类方法（static），多实现

抽象类：可以定义抽象方法和普通类具有的方法和属性，抽象类不可以被实体化，单继承

第二：从含义上：

接口是对动作的抽象，提供一种标准，规范，实现类并不明确

抽象类是对类抽象。对一类事物的共同点抽象，可能已经明确实现类。

### .◆重载和重写的区别

重载：发生在同一个类中，方法名必须相同，参数类型不同、个数不同、顺序不同，方法返回值和访问修饰 符可以不同，发生在编译时绑定。

重写：发生在父子类中，方法名、参数列表必须相同，运行期绑定

### ◆String和StringBuffer、StringBuilder的区别

#### 可变性

String类中使用字符数组保存字符串，private final char value[]，所以string对象是不可变的StringBuilder与StringBuffer都继承自 AbstractStringBuilder类，在AbstractStringBuilder中也是使用字符数组保存字符串，char[] value，这两种对象都是可变的

#### 线程安全性

String中的对象是不可变的，也就可以理解为常量，线程安全。

StringBuffer对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。

StringBuilder并没有对方法进行加同步锁，所以是非线程安全的。

### ◆final, finally, finalize的区别

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。

内部类要访问局部变量，局部变量必须定义成final类型，例如，一段代码……

finally是异常处理语句结构的一部分，表示总是执行。

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。JVM不保证此方法总被调用

### ◆[Java中==和equals的区别，equals和hashCode的区别](https://blog.csdn.net/tiantiandjava/article/details/46988461)

1基本数据类型，也称原始数据类型。byte,short,char,int,long,float,double,boolean   
 他们之间的比较，应用双等号（==）,比较的是他们的值

2.复合数据类型(类)   
  当他们用（==）进行比较的时候，比较的是他们在内存中的存放地址，所以，除非是同一个new出来的对象，他们的比较后的结果为true，否则比较后结果为false。 JAVA当中所有的类都是继承于Object这个基类的，在Object中的基类中定义了一个equals的方法，这个方法的初始行为是比较对象的内存地 址，但在一些类库当中这个方法被覆盖掉了，如String,Integer,Date在这些类当中equals有其自身的实现，而不再是比较类在堆内存中的存放地址了。

3.理论上约定：equals相等，hashcode必相等；hashcode相等，equals可能不相等。

### ◆HashMap源码：

◆如何保证hash值不越界

ab[i = (n - 1) & hash] 最大值不会超过n。 与运算：相同为1

◆条件的运算结果可做后面条件的值。

### ◆HashMap的原理，内部数据结构？

　　HashMap是基于hashing的原理，底层使用哈希表（数组 + 链表）实现。里边最重要的两个方法put、get，使用put(key, value)存储对象到HashMap中，使用get(key)从HashMap中获取对象。   
　　存储对象时，我们将K/V传给put方法时，它调用hashCode计算hash从而得到bucket位置，进一步存储，HashMap会根据当前bucket的占用情况自动调整容量(超过Load Facotr则resize为原来的2倍)。获取对象时，我们将K传给get，它调用hashCode计算hash从而得到bucket位置，并进一步调用equals()方法确定键值对。如果发生碰撞的时候，Hashmap通过链表将产生碰撞冲突的元素组织起来，在Java 8中，如果一个bucket中碰撞冲突的元素超过某个限制(默认是8)，则使用红黑树来替换链表，从而提高速度。

### ◆HashMap 中 put 方法过程？

1.对key的hashCode做hash操作，然后再计算在bucket中的index（1.5 HashMap的哈希函数）；   
2.如果没碰撞直接放到bucket里；   
3.如果碰撞了，以链表的形式存在buckets后；   
4.如果节点已经存在就替换old value(保证key的唯一性)   
5.如果bucket满了(超过阈值，阈值=loadfactor\*current capacity，load factor默认0.75)，就要

resize。

### ◆HashMap 中get()方法的工作原理？

通过对key的hashCode()进行hashing，并计算下标( n-1 & hash)，从而获得buckets的位置。如果产生碰撞，则利用key.equals()方法去链表中查找对应的节点。

### ◆HashMap 怎样解决冲突？

HashMap中处理冲突的方法实际就是链地址法，内部数据结构是数组+单链表。

## JVM内存模式、

### ◆解释JVM内存模型：

#### 1.程序计数器：

1.如果线程正在执行的是一个java方法，那么计数器记录的当前代码的行号，确定执行哪一条字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能都会依赖这个计数器完成

2.如果执行的native方法，计数器当中的内容应当是空   
3.此内存区域在java的虚拟机规范当中是唯一一个没有规定OutOfMemoryError的区域

#### 2.JVM栈：

栈帧：一个栈帧随着一个方法的调用开始而创建，这个方法调用完成而销毁。局部变量表存放了编译器可知的各种基本数据类型（boolean、byte、char、short、int、float、long、double）、对象引用（reference类型，它不等同于对象本身，可能是指向一个代表对象的句柄或者其他与此对象相关的变量）和retureAddress类型。其中64位长度的long和double类型的数据会占2个局部变量空间，其余的数据类型只占1个。

Java栈也称作虚拟机栈（Java Vitual Machine Stack），JVM栈只对栈帧进行存储，压栈和出栈操作。Java栈是Java方法执行的内存模型。下面我们来看一个Java栈图。

对于所有的程序设计语言来说，栈这部分空间对程序员来说是不透明的。

栈内存的大小可以有两种设置，固定值和根据线程需要动态增长。  
在JVM栈这个数据区可能会发生抛出两种错误。   
1. StackOverflowError 出现在栈内存设置成固定值的时候，当程序执行需要的栈内存超过设定的固定值会抛出这个错误。   
2. OutOfMemoryError 出现在栈内存设置成动态增长的时候，当JVM尝试申请的内存大小超过了其可用内存时会抛出这个错误。

#### 3.本地方法栈

这块内存区域和虚拟机栈非常相似，他们的区别从名字就可以看出来：Java虚拟机栈是用来执行Java方法的，而本地方法栈是用来执行Native方法的。

#### 4.堆内存：

Java堆应该是虚拟机管辖范围内最大的一块内存区域。这块区域是被所有线程共享，在虚拟机一启动的时候就创建的，它的唯一目的就是存放对象实例，几乎所有对象的实例都需要到这里申请内存。

Java堆是垃圾回收期（GC）管理的主要区域，从内存回收的角度来看，现代的收集器都采用分代收集算法，所有这块区域又会细分为好几个区域：新生代、老年代；再细节的又会分为 Eden、From Survivor、To Survivor，

**新生代（Young Generation）**：对象被创建时，内存的分配首先发生在年轻代（大多数对象可以直接被创建在年老代），大部分的对象在创建后很快就不再使用，因此很快变得不可用，于是被年轻代的GC机制清理掉（IBM的研究表明，98%的对象都是很快消亡的），这个GC机制被称为Minor GC或叫Young GC。注意，Minor GC并不代表年轻代内存不足，它事实上只表示在Eden区上的GC。

年轻代上的内存分配是这样的，年轻代可以分为3个区域：Eden区（伊甸园，亚当和夏娃偷吃禁果生娃娃的地方，用来表示内存首次分配的区域，再贴切不过）和两个存活区（Survivor 0 、Survivor 1）

绝大多数刚创建的对象会被分配在Eden区，其中的大多数对象很快就会消亡。Eden区是连续的内存空间，因此在其上分配内存极快；

最初一次，当Eden区满的时候，执行Minor GC，将消亡的对象清理掉，并将剩余的对象复制到一个存活区Survivor0（此时，Survivor1是空白的，两个Survivor总有一个是空白的）；

下次Eden区满了，再执行一次Minor GC，将消亡的对象清理掉，将存活的对象复制到Survivor1中，然后清空Eden区；

 将Survivor0中消亡的对象清理掉，将其中可以晋级的对象晋级到Old区，将存活的对象也复制到Survivor1区，然后清空Survivor0区；

当两个存活区切换了几次（HotSpot虚拟机默认15次，用-XX:MaxTenuringThreshold控制，大于该值进入老年代，但这只是个最大值，并不代表一定是这个值）之后，仍然存活的对象（其实只有一小部分，比如，我们自己定义的对象），将被复制到老年代。

　从上面的过程可以看出，Eden区是连续的空间，且Survivor总有一个为空。经过一次GC和复制，一个Survivor中保存着当前还活着的对象，而Eden区和另一个Survivor区的内容都不再需要了，可以直接清空，到下一次GC时，两个Survivor的角色再互换。因此，这种方式分配内存和清理内存的效率都极高，这种垃圾回收的方式就是著名的**“停止-复制（Stop-and-copy）”清理法（将Eden区和一个Survivor中仍然存活的对象拷贝到另一个Survivor中）**

**年老代（Old Generation）**：对象如果在年轻代存活了足够长的时间而没有被清理掉（即在几次Young GC后存活了下来），则会被复制到年老代，年老代的空间一般比年轻代大，能存放更多的对象，在年老代上发生的GC次数也比年轻代少。当年老代内存不足时，将执行Major GC，也叫 Full GC。

#### 5.方法区(非堆内存)：

**【名词解析】  
        --->和java堆一样，方法区是一块所有线程共享的内存区域。  
        --->保存系统的类信息，比如，类的字段，方法，常量池等。  
        --->方法区的大小决定了系统可以保存多少个类，如果系统定义了太多的类，导致方法区溢出，虚拟机同样会抛出内存溢出的错误  
        --->jdk1.6和jdk1.7方法区可以理解为永久区。  
        --->jdk1.8已经将方法区取消，替代的是元数据区。  
        --->jdk1.8的元数据区可以使用参数-XX:MaxMetaspaceSzie设定大小，这是一块堆外的直接内存，与永久区不同，如果不指定大小，默认情况下，虚拟机会耗尽可用系统内存**

### ◆在JVM调优，

可以通过配置以下参数改变整个JVM堆的配置比例

1.Java heap的大小（新生代+老年代）

　　-Xms堆的最小值

　　-Xmx堆空间的最大值

2.新生代堆空间大小调整

　　-XX:NewSize新生代的最小值

　　-XX:MaxNewSize新生代的最大值

　　-XX:NewRatio设置新生代与老年代在堆空间的大小

　　-XX:SurvivorRatio新生代中Eden所占区域的大小

3.永久代大小调整

　　-XX:MaxPermSize

4.其他

　 -XX:MaxTenuringThreshold,设置将新生代对象转到老年代时需要经过多少次垃圾回收，但是仍然没有被回收

### ◆volatile的语义，它修饰的变量一定线程安全吗

**可见性**：当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，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。

**原子性**：在Java中，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，volatile:不具备。

### ◆Minor GC ，Full GC 触发条件

Minor GC触发条件：当Eden区满时，触发Minor GC。

Full GC触发条件：

（1）调用System.gc时，系统建议执行Full GC，但是不必然执行

（2）老年代空间不足

### ◆说一下强引用、软引用、弱引用、虚引用以及他们之间和gc的关系

强引用：new出的对象之类的引用，只要强引用还在，永远不会回收   
软引用：引用但非必须的对象，内存溢出异常之前，回收   
弱引用：非必须的对象，对象能生存到下一次垃圾收集发生之前。   
虚引用：对生存时间无影响，在垃圾回收时得到通知。

## 多线程

### ◆volatile和synchronized对比？

1）volatile本质是在告诉jvm当前变量在寄存器中的值是不确定的,需要从主存中读取,synchronized则是锁定当前变量,只有当前线程可以访问该变量,其他线程被阻塞住.

2）volatile仅能使用在变量级别,synchronized则可以使用在变量,方法.

3）volatile仅能实现变量的修改可见性,而synchronized则可以保证变量的修改可见性和原子性.

4）volatile不会造成线程的阻塞,而synchronized可能会造成线程的阻塞.

### ◆Lock和synchronized对比？

1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；

2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；

3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；

4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。

5）Lock可以提高多个线程进行读操作的效率。

6）在JDK1.5中，synchronized是性能低效的。因为这是一个重量级操作，它对性能最大的影响是阻塞式的实现，挂起线程和恢复线程的操作都需要转入内核态中完成，这些操作给系统的并发性带来了很大的压力。相比之下使用Java提供的Lock对象，性能更高一些。

### ◆什么是乐观锁和悲观锁？

（1）乐观锁：对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-设置这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

（2）悲观锁：对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，直接对操作资源上了锁

## Redis

### Redis支持的数据类型

Redis通过Key-Value的单值不同类型来区分, 以下是支持的类型:  
Strings  
Lists  
Sets 求交集、并集  
Sorted Set   
hashes

### – 什么是Redis？

redis是一个开源的、使用C语言编写的、支持网络交互的、可基于内存也可持久化的Key-Value数据库。

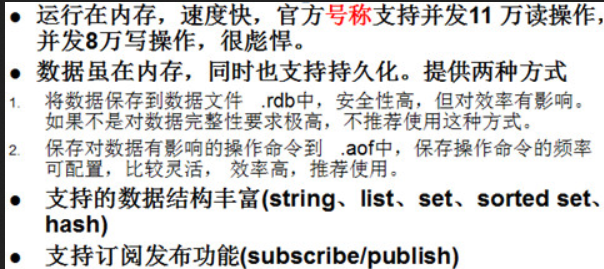
### –Redis的特性有哪些？

1、速度快   
Redis是用C语言实现的；   
Redis的所有数据存储在内存中。   
2、持久化   
Redis的所有数据存储在内存中，对数据的更新将异步地保存到磁盘上。   
3、支持多种数据结构   
Redis支持五种数据结构：String、List、Set、Hash、Zset   
4、支持多种编程语言   
Java、php、Python、Ruby、Lua、Node.js   
5、功能丰富   
除了支持五种数据结构之外，还支持事务、流水线、发布/订阅、消息队列等功能。   
6、源码简单   
约23000行C语言源代码。   
7、主从复制   
主服务器（master）执行添加、修改、删除，从服务器执行查询。   
8、高可用及分布式   
Redis-Sentinel（v2.8）支持高可用   
Redis-Cluster（v3.0）支持分布式

什么是高可用？   
一台服务器挂掉会有另一台服务器立马顶上，Redis-Sentinel监听所有的监听所有的Redis主从服务器，一旦发现主服务器挂掉了，他会从若干服务器中选择一台新的机器作为master，其他从服务器会成为这台新master的从服务器。

什么是分布式？   
在一台物理计算机上，把数据A放到数据库D1，把数据B放到数据库D2，把数据C放到数据库D3属于分布式，但由于数据库的访问连接池是有限的，所以这种分布式解决方案存在性能瓶颈；   
把数据A放到计算机C1的数据库D1，把数据B放到计算机C1的数据库D2，把数据C放到计算机C3的数据库D3也属于分布式，这种分布式解决方案扩展性好，如果系统增加了新业务X，X产生的数据Y可以放到计算机C4的数据库D4中，理论上来说这种方式可以无限扩展，从而保证系统的性能最大化。   
存在瓶颈且价格较高的高性能单台服务器->可无限扩展的分布式集群

### – 为什么要使用Redis？



– 如何使用Redis？

Redis 数据类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

String（字符串）

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

实例

redis 127.0.0.1:6379> SET name "runoob"

OK

redis 127.0.0.1:6379> GET name

"runoob"

在以上实例中我们使用了 Redis 的 **SET** 和 **GET** 命令。键为 name，对应的值为 **runoob**。

**注意：**一个键最大能存储512MB。

Hash（哈希）

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

实例

redis> HMSET myhash field1 "Hello" field2 "World"

"OK"

redis> HGET myhash field1

"Hello"

redis> HGET myhash field2

"World"

以上实例中 hash 数据类型存储了包含用户脚本信息的用户对象。 实例中我们使用了 Redis **HMSET, HGETALL** 命令，**user:1** 为键值。

每个 hash 可以存储 232 -1 键值对（40多亿）。

List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

实例

redis 127.0.0.1:6379> lpush runoob redis

(integer) 1

redis 127.0.0.1:6379> lpush runoob mongodb

(integer) 2

redis 127.0.0.1:6379> lpush runoob rabitmq

(integer) 3

redis 127.0.0.1:6379> lrange runoob 0 10

1) "rabitmq"

2) "mongodb"

3) "redis"

redis 127.0.0.1:6379>

列表最多可存储 232 - 1 元素 (4294967295, 每个列表可存储40多亿)。

Set（集合）

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

sadd 命令

添加一个string元素到,key对应的set集合中，成功返回1,如果元素已经在集合中返回0,key对应的set不存在返回错误。

sadd key member

实例

redis 127.0.0.1:6379> sadd runoob redis

(integer) 1

redis 127.0.0.1:6379> sadd runoob mongodb

(integer) 1

redis 127.0.0.1:6379> sadd runoob rabitmq

(integer) 1

redis 127.0.0.1:6379> sadd runoob rabitmq

(integer) 0

redis 127.0.0.1:6379> smembers runoob

1) "rabitmq"

2) "mongodb"

3) "redis"

**注意：**以上实例中 rabitmq 添加了两次，但根据集合内元素的唯一性，第二次插入的元素将被忽略。

集合中最大的成员数为 232 - 1(4294967295, 每个集合可存储40多亿个成员)。

zset(sorted set：有序集合)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

zadd 命令

添加元素到集合，元素在集合中存在则更新对应score

zadd key score member

实例

redis 127.0.0.1:6379> zadd runoob 0 redis

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 mongodb

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabitmq

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabitmq

(integer) 0

redis 127.0.0.1:6379> ZRANGEBYSCORE runoob 0 1000

1) "redis"

2) "mongodb"

3) "rabitmq"

### –Redis的实现原理是什么？

### –Redis的优缺点是什么？

**1. 使用redis有哪些好处？**

(1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

(2) 支持丰富数据类型，支持string，list，set，sorted set，hash

(3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

(4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

**2. redis相比memcached有哪些优势？**

(1) memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型

(2) redis的速度比memcached快很多

(3) redis可以持久化其数据

**3. redis常见性能问题和解决方案：**

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

## Spring

## 多线程：

### ◆详解synchronized与Lock的区别与使用

1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；

2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；

3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；

4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。

5）Lock可以提高多个线程进行读操作的效率。

## Servlet面试题：

### ◆说一说Servlet生命周期

Servlet生命周期包括三部分：

初始化：Web容器加载servlet，调用init()方法

处理请求：当请求到达时，运行其service()方法。service()自动派遣运行与请求相对应的doXXX（doGet或者doPost）方法。

销毁：服务结束，web容器会调用servlet的distroy()方法销毁servlet。

### ◆get提交和post提交有何区别

1. 从提交的数据量上来说，

get方式会将请求参数及参数值放在请求资源路径里面，携带的数据只能是文本，且大小不超过1024个字节；post方式会将请求参数及参数值放在实体内容里面，理论上没有限制，适合大量数据的提交。不仅可以提交文本还有二进制文件。

2从安全上来讲，

post方式相对安全因为请求参数及值存放在实体内容里面，而get方式会将请求参数及值显示在浏览器地址栏。但是要注意，post方式并没有将数据加密。

### ◆JSP与Servlet有什么区别

1.Servlet是服务器端的程序，动态生成html页面发送到客户端，但是这样程序里会有很多out.println(),java与html语言混在一起很乱，所以后来sun公司推出了JSP.其实JSP就是Servlet，每次运行的时候JSP都首先被编译成servlet文件，然后再被编译成class文件运行。有了jsp，在MVC项目中servlet不再负责动态生成页面，转而去负责控制程序逻辑的作用，控制jsp与javabean之间的流转。

### ◆doGet与doPost方法的两个参数是什么

HttpServletRequest：封装了与请求相关的信息

HttpServletResponse：封装了与响应相关的信息

### ◆request.getAttribute()和request.getParameter

（1）有setAttribute，没有setParameter方法

（2）getParameter获取到的值只能是字符串，不可以是对象，而getAttribute获取到的值是Object类型的。

（3）通过form表单或者url来向另一个页面或者servlet传递参数的时候需要用getParameter获取值；getAttribute只能获取setAttribute的值

（4）setAttribute是应用服务器把这个对象放到该页面所对应的一块内存当中，当你的页面服务器重定向到另一个页面的时候，应用服务器

会把这块内存拷贝到另一个页面对应的内存当中。通过getAttribute可以取得你存下的值，当然这种方法可以用来传对象。

用session也是一样的道理，这是说request和session的生命周期不一样而已。

### ◆JSP有哪些内置对象，作用是什么？

|  |  |
| --- | --- |
| JSP内置对象 | |
| 名称 | 作用 |
| request | 包含用户端请求的信息 |
| response | 包含服务器传回客户端的响应信息 |
| session | 与请求有关的会话期 |
| pageContext | 管理网页属性 |
| application | 服务器启动时创建，服务器关闭时停止，为多个应用程序保存信息 |
| out | 向客户端输出数据 |
| config | servlet的架构部件 |
| page | 指网页本身 |
| exception | 针对错误页面才可使用 |

### ◆四种会话跟踪技术作用域

（1）page：一个页面

（2）request：：一次请求

（3）session：一次会话

（4）application：服务器从启动到停止。

### ◆JSP中动态INCLUDE和静态INCLUDE有什么区别

include指令用于把另一个页面包含到当前页面中，在什么时候包含的？再转换成servlet的时候包含进去的。

动态INCLUDE用jsp:include动作实现 <jsp:include page="included.jsp" flush="true" />它总是会检查所含文件中的变化,适合用于包含动态页面,

并且可以带参数.

静态INCLUDE用include伪码实现,定不会检查所含文件的变化,适用于包含静态页面<%@ include file="included.htm" %>

### ◆forward和redirect的区别

转发与重定向

（1）从地址栏显示来说

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送

的内容从哪里来的,所以它的地址栏还是原来的地址.redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是

新的URL.

（2）从数据共享来说

forward:转发页面和转发到的页面可以共享request里面的数据.

redirect:不能共享数据.

（3）从运用地方来说

forward:一般用于用户登陆的时候,根据角色转发到相应的模块.

redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等.

（4）从效率来说

forward:高.

redirect:低.

<!--简述如何处理表单提交中文-->

处理表单提交的中文，分为两种情况，

1.处理post方式提交表单时的中文，步骤如下：

step1：确保表单所在的页面按照指定的字符集打开，在HTML中设置如下<metahttp-equiv=“content-type”content=“text/html;charset=utf-8”>另外，在HTML中，将表单form的提交方式设置为post。

step2：在服务器端按照上述设置的编码格式进行解码，代码如下：request.setCharacterEncoding(“utf-8”)；该行代码要在第一次使用request的时候进行设置。

2.处理get方式提交表单时的中文，步骤如下：step1：使用meta确保表单所在页面按指定字符集打开，在HTML中设置如下：<metahttp-equiv=“content-type”content=“text/html;charset=utf-8”>另外，在HTML中，将表单form的提交方式设置为get。step2：将从表单中获取的信息使用上述设置的字符集utf-8进行重新编码。例如：将从表单获取的username进行重新编码，代码如下：Stringusername=request.getParameter(“username”);username=newStringusername.getBytes(“iso-8859-1”),“utf-8”);

## SpringMVC

## Spring

### ◆解释什么是spring：

首先Spring是一个大的概念，Spring从最开始的一个Library到现在一个系列，其中最主要的包括Spring Framework, Spring Data, Spring Security, Spring Batch等等，以及快速框架Spring Boot，其中最重要的项目是Spring Framework，这个项目包括了IoC, AOP, MVC以及Testing。

第一个需要明白的是Spring的核心思想是什么？  
Spring整个系列的最最核心的概念当属IoC, AOP，什么是IoC和AOP就不展开了，简而言之，将对象创建过程的职责赋予容器，通过容器管理对象的生老病死， 将对象创建过程从编译时延期到运行时，即通过配置进行加载，这样一来就解决了不用编译后期选择具体实现，其实就是面向对象的核心理念，针对接口编程。IoC开始就是个factory加上依赖管理罢了，这样一来，一个系统的创建过程就从原先的new改为配置组装，内部通过注入解决了依赖关系，只要满足接口协议即插即用。通过IoC, AOP事实上形成了一个套路，通过这个套路完成了系统的整合。

所以Spring并没有说自己写一个ORM，而是用统一的套路完成了多个ORM的集成，这也是Spring越做越大的基础，慢慢就形成了Spring Way，其实这个才是Spring最有价值的地方。

第二当然就是一些实践，其实主流问的大概也就几个方向，用的最多的应该就是Spring MVC， Spring Data, Spring Security和Spring Boot这几块吧，因为毕竟这是实践性内容，很多时候都是show me the code，之前学习过程写过一个sample，基本全是标准Spring Way，你可以拿去参考一下，[GitHub - nonocast/todolist: A simple todolist system implemented with Spring's backend and TypeScript/React's frontend. Enjoy it.](https://link.zhihu.com/?target=https%3A//github.com/nonocast/todolist)

## SpringAOP

### ◆面试问题：怎么控制多个aop的执行循序

pring声明式事务是基于AOP实现的，那么，如果我们在同一个方法自定义多个AOP，我们如何指定他们的执行顺序呢？网上很多答案都是指定order，order越小越是最先执行

配置AOP执行顺序的三种方式：

1. 通过实现org.springframework.core.Ordered接口
   1. @Component
   2. @Aspect
   3. @Slf4j
   4. **public** **class** MessageQueueAopAspect1 **implements** Ordered{@Override
   5. **public** **int** getOrder() {
   6. // TODO Auto-generated method stub
   7. **return** 2;
   8. }
   10. }
2. 通过注解
   1. @Component
   2. @Aspect
   3. @Slf4j
   4. @Order(1)
   5. **public** **class** MessageQueueAopAspect1{
   7. ...
   8. }
3. 通过配置文件配置
   1. **<aop:config** expose-proxy="true"**>**
   2. **<aop:aspect** ref="aopBean" order="0"**>**
   3. **<aop:pointcut** id="testPointcut"  expression="@annotation(xxx.xxx.xxx.annotation.xxx)"**/>**
   4. **<aop:around** pointcut-ref="testPointcut" method="doAround" **/>**
   5. **</aop:aspect>**
   6. **</aop:config>**

### ◆说出Spring的通知类型有哪些？

1. 前置通知  
\* 在目标类的方法执行之前执行。  
\* 配置文件信息：<aop:before method="before" pointcut-ref="myPointcut3"/>  
\* 应用：可以对方法的参数来做校验  
  
2. 最终通知  
\* 在目标类的方法执行之后执行，如果程序出现了异常，最终通知也会执行。  
\* 在配置文件中编写具体的配置：<aop:after method="after" pointcut-ref="myPointcut3"/>  
\* 应用：例如像释放资源  
  
3. 后置通知  
\* 方法正常执行后的通知。   
\* 在配置文件中编写具体的配置：<aop:after-returning method="afterReturning" pointcut-ref="myPointcut2"/>  
\* 应用：可以修改方法的返回值  
  
4. 异常抛出通知  
\* 在抛出异常后通知  
\* 在配置文件中编写具体的配置：<aop:after-throwing method="afterThorwing" pointcut-ref="myPointcut3"/>   
\* 应用：包装异常的信息  
  
5. 环绕通知  
\* 方法的执行前后执行。  
\* 在配置文件中编写具体的配置：<aop:around method="around" pointcut-ref="myPointcut2"/>  
\* 要注意：目标的方法默认不执行，需要使用ProceedingJoinPoint对来让目标对象的方法执行

### ◆谈谈目标对象实现接口与目标对象不实现接口有什么区别？

（1）如果目标对象实现了接口,默认采用JDK的动态代理机制实现AOP

（2）如果目标对象实现了接口,可以强制spring采用CGLIB实现代理

（3） 如果目标对象没有实现接口,必须采用CGLIB实现代理,spring会自动的在CGLIB和JDK动态 代理之间切换

### ◆请描述JDK动态代理和CGLI代理的区别？

**JDK动态代理：**代理对象和目标对象实现了相同的接口，目标对象作为代理对象的一个属性，具体接口实现中，可以在调用目标对象相应方法前后加上其他业务处理逻辑。

代理模式在实际使用时需要指定具体的目标对象，如果为每个类都添加一个代理类的话，会导致类很多，同时如果不知道具体类的话，怎样实现代理模式呢？这就引出动态代理。

 JDK动态代理只能针对**实现了接口的类**生成代理。

**CGLIB（CODE GENERLIZE LIBRARY）代理：**是针对类实现代理，主要是对指定的类生成一个**子类**，覆盖其中的所有方法，所以该类或方法不能声明称final的。

如果目标对象没有实现接口，则默认会采用CGLIB代理；

如果目标对象实现了接口，可以强制使用CGLIB实现代理（添加CGLIB库，并在spring配置中加入<aop:aspectj-autoproxy proxy-target-class="true"/>）。

### ◆简述ProxyFactoryBean的作用是什么？

### ◆叙述Spring中的自动代理的原理？

### ◆ 写出创建代理对象需指定的三要素是什么？

### ◆写出代理的两种方式分别是什么？

### ◆请简述：什么是AOP？

### ◆AOP里面重要的几个名词概念解释：

切面（Aspect）： 一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是J2EE应用中一个关于横切关注点的很好的例子。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @Aspect 注解（@AspectJ风格）来实现。   
  
连接点（Joinpoint）： 在程序执行过程中某个特定的点，比如某方法调用的时候或者处理异常的时候。 在Spring AOP中，一个连接点 总是 代表一个方法的执行。 通过声明一个org.aspectj.lang.JoinPoint类型的参数可以使通知（Advice）的主体部分获得连接点信息。   
  
通知（Advice）： 在切面的某个特定的连接点（Joinpoint）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。 通知的类型将在后面部分进行讨论。许多AOP框架，包括Spring，都是以拦截器做通知模型， 并维护一个以连接点为中心的拦截器链。   
  
切入点（Pointcut）： 匹配连接点（Joinpoint）的断言。通知和一个切入点表达式关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。 切入点表达式如何和连接点匹配是AOP的核心：Spring缺省使用AspectJ切入点语法。   
  
引入（Introduction）： （也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。 Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。 例如，你可以使用一个引入来使bean实现 IsModified 接口，以便简化缓存机制。   
  
目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（advised） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。   
  
AOP代理（AOP Proxy）： AOP框架创建的对象，用来实现切面契约（aspect contract）（包括通知方法执行等功能）。 在Spring中，AOP代理可以是JDK动态代理或者CGLIB代理。 注意：Spring 2.0最新引入的基于模式（schema-based）风格和@AspectJ注解风格的切面声明，对于使用这些风格的用户来说，代理的创建是透明的。   
  
织入（Weaving）： 把切面（aspect）连接到其它的应用程序类型或者对象上，并创建一个被通知（advised）的对象。 这些可以在编译时（例如使用AspectJ编译器），类加载时和运行时完成。 Spring和其他纯Java AOP框架一样，在运行时完成织入。

### ◆简述AOP核心？

### ◆请叙述AOP事务的含义？

1. 下面关于spring描述错误的是：（ ）  
A Spring支持可插入的事务管理器，使事务划分更轻松，同时无需处理底层的问题。  
B Spring事务管理的通用抽象层还包括JTA策略和一个JDBC DataSource。  
C 与JTA或EJB CMT一样，Spring的事务支持依赖于Java EE环境。  
D Spring事务语义通过AOP应用于 POJO，通过XML或Java SE 5注释进行配置。

2. 下面选项哪个不是Spring中接口注入的方式？（ ）  
A 接口注入　　  
B 构造子注入　　  
C 设值注入　　  
D 指针注入

3. 下列关于Spring特性中IoC描述错误的是：（ ）  
A IoC就是指程序之间的关系由程序代码直接操控。  
B 所谓“控制反转”,是指控制权由应用代码转到外部容器，控制权的转移，  
C IoC将控制创建的职责搬进了框架中；并把它从应用代码脱离开来  
D 当使用Spring的IoC容器时只需指出组件需要的对象，在运行时Spring的IoC容器会根据XML配置数据提供给它。

简答题：  
1. 请简述Spring的工作机制？

2. 请回答你为什么用Spring的工作机制？

3. 请简述Spring是什么？

4. 简述spring的组成？

5.简述Spring容器提供了哪些功能？

6. 在Spring中，bean的注入有几种方式，各是什么？

7. 请简述：Spring bean的作用域？

8. 请叙述设值注入的优点？

9. 请叙述构造注入的优点？

10. 说出bean工厂创建bean的三种方式？  
a  
11. 请写出bean的生命周期的方法？

12. 请简述你对IOC的理解？

13. 请回答：IoC最大的好处是什么？

14. 简述IoC的类型？

15. Spring中依赖注入与传统编程之间的差别是什么？

## Spring事务

### ◆并发事务导致的问题

第一类

丢失更新：撤销一个事务时，把其他事务已提交的更新数据覆盖。

|  |
| --- |
| 小明去银行柜台存钱，他的账户里原来的余额为100元，现在打算存入100元。在他存钱的过程中，银行年费扣了5元，余额只剩95元。突然他又想着这100元要用来请女朋友看电影吃饭，不打算存了。在他撤回存钱操作后，余额依然为他存钱之前的100元。所以那5块钱到底扣了谁的？ |

脏读：一个事务读取到另一个事务未提交的更新数据。

|  |
| --- |
| 小明的银行卡余额里有100元。现在他打算用手机点一个外卖饮料，需要付款10元。但是这个时候，他的女朋友看中了一件衣服95元，她正在使用小明的银行卡付款。于是小明在付款的时候，程序后台读取到他的余额只有5块钱了，根本不够10元，所以系统拒绝了他的交易，告诉余额不足。但是小明的女朋友最后因为密码错误，无法进行交易。小明非常郁闷，明明银行卡里还有100元，怎么会余额不足呢？（他女朋友更郁闷。。。） |

幻读也叫虚读：一个事务执行两次查询，第二次结果集包含第一次中没有或某些行已经被删除的数据，造成两次结果不一致，只是另一个事务在这两次查询中间插入或删除了数据造成的。

|  |
| --- |
| 公司财务A在进行员工薪资核算业务，需要对小明的工资进行计算并录入系统，必须查询两次明细信息，然后将后一次的明细信息计算总数出来。财务在第一次明细查询时，基本工资2000元，全勤奖1000元，提成2000元，共计5000元。在进行第二次计算时，财务B突然接到通知，需要把下个月的节日福利也在这个月的工资中发放，每人100元。于是财务B在每个人的工资明细中又加了一条节日福利100元。而此时财务A获得第二次查询小明的工资明细后，发现工资明细变成了4条数据，总数是5100元。两次计算结果相差100元，财务A奇怪这多出来的一条明细100元是哪里来的呢？（都怪财务B不告诉A。。。） |

不可重复读：一个事务两次读取同一行的数据，结果得到不同状态的结果，中间正好另一个事务更新了该数据，两次结果相异，不可被信任。

|  |
| --- |
| 小明在手机上购买起购价为1W元理财产品。系统首先要判断他的余额够不够购买理财产品，如果足够再获取当前的余额，进行申请。系统第一次读取到小明的余额还剩1W元，刚好足够购买产品。但是这个时候刚好他女朋友又看中了一个包包5000元，这次密码终于不会再错误的女朋友毫不犹豫刷了小明的银行卡买下了这个包包。但是这个系统刚好在进行第二次确认，发现小明的余额上只有5000元，根本不够购买。于是系统很生气，拒绝了小明的购买行为，告诉他，你真是个骗子！！！ |

第二类：

丢失更新：是不可重复读的特殊情况。如果两个事物都读取同一行，然后两个都进行写操作，并提交，第一个事物所做的改变就会丢失。

|  |
| --- |
| 小明和女朋友一起去逛街。女朋友看中了一支口红，（对，女朋友就是用来表现买买买的）小明大方的掏出了自己的银行卡，告诉女朋友：亲爱的，随便刷，随便买，我坐着等你。然后小明就坐在商城座椅上玩手机，等着女朋友。这个时候，程序员的聊天群里有人推荐了一本书，小明一看，哎呀，真是本好书，还是限量发行呢，我一定更要买到。于是小明赶紧找到购买渠道，进行付款操作。而同时，小明的女朋友也在不亦乐乎的买买买，他们同时进行了一笔交易操作，但是这个时候银行系统出了问题，当他们都付款成功后，却发现，银行只扣了小明的买书钱，却没有扣去女朋友此时交易的钱。哈哈哈，小明真是太开心了！ |

## Mybatis



MyBatis应用程序根据XML配置文件创建SqlSessionFactory，SqlSessionFactory在根据配置，配置来源于两个地方，一处是配置文件，一处是Java代码的注解，获取一个SqlSession。SqlSession包含了执行sql所需要的所有方法，可以通过SqlSession实例直接运行映射的sql语句，完成对数据的增删改查和事务提交等，用完之后关闭SqlSession。

### 1、#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

### 2、当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

第2种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

### 3、 模糊查询like语句该怎么写?

第1种：在Java代码中添加sql通配符。

第2种：在sql语句中拼接通配符，会引起sql注入

### 数据库面试题：