# Java基础

## 1-常见名词解释？

OOA: Object-Oriented Analysis-面向对象分析

OOD: Object-Oriented Design-面向对象设计

OOP: Object-Oriented Programming-面向对象编程

OOT: Object-Oriented Testing-面向对象测试

ORM: Object-Relational Mapping-对象关系映射

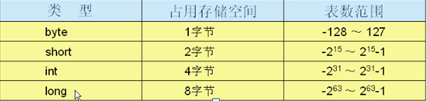
SOA: Service-Oriented Architecture-面向服务的体系结构

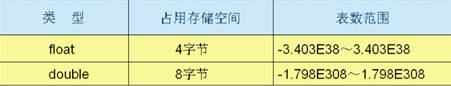
## 2-Java支持的数据类型有哪些？

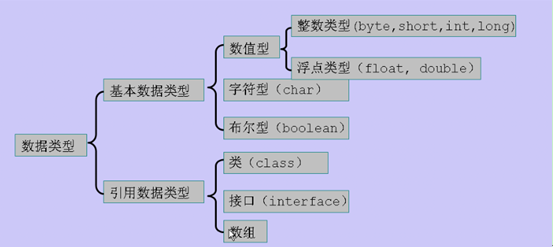
1.基本数据类型：

byte(1) short(2) int(4) long(8)

float(4) double(8) char(2) Boolean(true/false)







2.自动装箱与拆箱/常用方法：

Java提供8中基本类型的封装类型，分别为Byte、Short、Integer、Long、Float、Double、Character、Boolean

数据级别由低到高为：（byte-short-char）-int-long-float-double

3.类型转换问题：

转换原则：

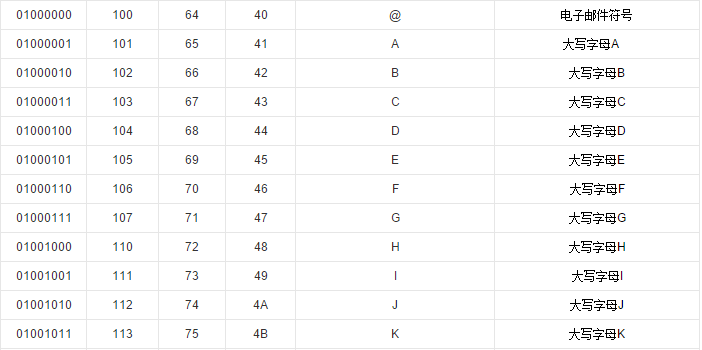
低级到高级的自动类型转换；

高级到低级的强制类型转换（会导致溢出或丢失精度）；

byte，short，char之间不会互相转换，并且三者在计算时首先转换为int类型

小数常量默认为double类型， 整数常量默认为int类型；

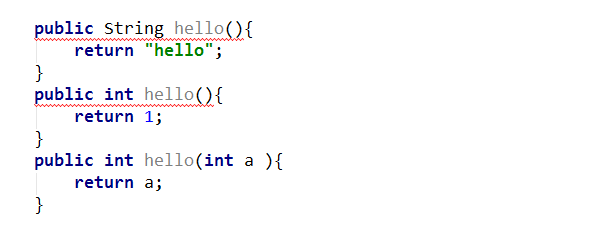
char字符型与整形和浮点型做运算时，先获得字符型的ASCII码值然后进行计算



## 3-Java中方法重载、方法覆盖？

1.方法重载：

方法重载发生在同一个类里面，是指有两个或者两个以上的方法，它们的方法名相同但是参数类型和个数不同，返回值可相同也可不同的方法，重载是一个类中多态性的表现



2.方法覆盖：

方法覆盖发生在子类与父类之间，是指子类定义的方法与父类具有相同的返回值类型、方法名和参数列表，这时父类方法将被覆盖；

使用super可引用父类方法；

子类方法的访问修饰符不能小于父类；（父类引用子类对象）；

子类的异常要小于父类的异常；

final private 修饰的方法不能被子类覆盖；

方法覆盖与实现接口的方法原理很相似，覆盖父类方法，要求比较严格

## 4-Java基类是什么？有哪些方法？

基类：Object

常用方法：

toString(); //将对象转换成string

equals(); //Object比较引用地址值，子类如String类比较对象值

native hashcode(); // hashcode值，每个对象值唯一

native clone();

native notify()

notifyAll()

wait();

…

## 5-****String,StringBuffer, StringBuilder 的区别是什么？****

1、String是字符串常量，StringBuffer和StringBuilder都是字符串变量。后两者的字符内容可变，而前者创建后内容不可变。

2、String不可变是因为在JDK中String类被声明为一个final类。

3、StringBuffer是线程安全的，而StringBuilder是非线程安全的。

## 6-面向对象的特点？

1.抽象性：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面

2.继承：继承是从已有类得到继承信息创建新类的过程

3.封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口

4.多态性：分为编译时多肽和运行时多肽，编译器多肽主要体现在方法重载，运行时多肽主要体现在方法重写

## 7-访问修饰符public,private,protected,以及不写（默认）时的区别？

| **修饰符** | **当前类** | **同 包** | **子 类** | **其他包** |
| --- | --- | --- | --- | --- |
| public | **√** | **√** | **√** | **√** |
| protected | **√** | **√** | **√** | × |
| default | **√** | **√** | × | × |
| private | **√** | × | × | × |

类的成员不写访问修饰时默认为default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java中，外部类的修饰符只能是public或默认，类的成员（包括内部类）的修饰符可以是以上四种。

## 8-String 是最基本的数据类型吗？

答、不是。Java中的基本数据类型只有8个：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

## 9-float f=3.4;是否正确？ 答:不正确。3.4是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成float f =3.4F。

## 10-short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？ 答：对于short s1 = 1; s1 = s1 + 1;由于1是int类型，因此s1+1运算结果也是int 型，需要强制转换类型才能赋值给short型。而short s1 = 1; s1 += 1;可以正确编译，因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

## 11-Java有没有goto？ 答：goto 是Java中的保留字，在目前版本的Java中没有使用。（根据James Gosling（Java之父）编写的《The Java Programming Language》一书的附录中给出了一个Java关键字列表，其中有goto和const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉C语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

## **12-int和Integer有什么区别？** 答：Java是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入了基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java为每一个基本数据类型都引入了对应的包装类型（wrapper class），int的包装类就是Integer，从Java 5开始引入了自动装箱/拆箱机制，使得二者可以相互转换。 Java 为每个原始类型提供了包装类型： - 原始类型: boolean，char，byte，short，int，long，float，double - 包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class AutoUnboxingTest {   public static void main(String[] args) {     Integer a = new Integer(3);     Integer b = 3; //将3自动装箱成Integer类型       int c = 3;      System.out.println(a == b)//false 两个引用没有引用同一对象System.out.println(a == c);//true a自动拆箱成int类型再和c比较    }  } |

最近还遇到一个面试题，也是和自动装箱和拆箱有点关系的，代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class Test03 {   public static void main(String[] args) {     Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;    System.out.println(f1 == f2);     System.out.println(f3 == f4);    }  } |

如果不明就里很容易认为两个输出要么都是true要么都是false。首先需要注意的是f1、f2、f3、f4四个变量都是Integer对象引用，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个Integer对象赋一个int值的时候，会调用Integer类的静态方法valueOf，如果看看valueOf的源代码就知道发生了什么

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static Integer valueOf(int i) {      if (i >= IntegerCache.low && i <= IntegerCache.high)          return IntegerCache.cache[i + (-IntegerCache.low)];      return new Integer(i);  } |

IntegerCache是Integer的内部类，其代码如下所示：

|  |  |
| --- | --- |
|  | /\*\*       \* Cache to support the object identity semantics of autoboxing for values between       \* -128 and 127 (inclusive) as required by JLS.       \*       \* The cache is initialized on first usage.  The size of the cache       \* may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.       \* During VM initialization, java.lang.Integer.IntegerCache.high property       \* may be set and saved in the private system properties in the       \* sun.misc.VM class.       \*/        private static class IntegerCache {          static final int low = -128;          static final int high;          static final Integer cache[];            static {              // high value may be configured by property              int h = 127;              String integerCacheHighPropValue =                  sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");              if (integerCacheHighPropValue != null) {                  try {                      int i = parseInt(integerCacheHighPropValue);                      i = Math.max(i, 127);                      // Maximum array size is Integer.MAX\_VALUE                      h = Math.min(i, Integer.MAX\_VALUE - (-low) -1);                  } catch( NumberFormatException nfe) {                      // If the property cannot be parsed into an int, ignore it.                  }              }              high = h;                cache = new Integer[(high - low) + 1];              int j = low;              for(int k = 0; k < cache.length; k++)                  cache[k] = new Integer(j++);                // range [-128, 127] must be interned (JLS7 5.1.7)              assert IntegerCache.high >= 127;          }            private IntegerCache() {}      } |

简单的说，如果整型字面量的值在-128到127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象，所以上面的面试题中f1==f2的结果是true，而f3==f4的结果是false。

提醒：越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

**8、&和&&的区别？**  
答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

***补充：****如果你熟悉JavaScript，那你可能更能感受到短路运算的强大，想成为JavaScript的高手就先从玩转短路运算开始吧。*

# Java多线程

## 什么叫多线程？如何实现？

## 异步和多线程的区别？

# Java集合

## 1. java中有哪些集合类？为什么要有那么多种？

# Java IO流

## 1.流有哪些？字节流与字符流有什么不一样？它们间是怎么转换的呢？

2.

# Java JVM原理

## JVM的原理？（深入理解java虚拟机）

# Java GC垃圾回收机制

## GC原理？

# Java设计模式

## 常见的设计模式和应用？

# 开源框架

## Spring

## Spring boot

## Spring mvc

## Dropwizard

## Netty

## Dubbo

## Hibernate/MyBaties/Ibaties

# 消息机制

## Rabbitmq

## Kafka

## RabbitMQ对比Kafka

业界对于消息的传递有多种方案和产品，本文就比较有代表性的两个MQ(rabbitMQ,kafka)进行阐述和做简单的对比，

在应用场景方面，

RabbitMQ,遵循AMQP协议，由内在高并发的erlanng语言开发，用在实时的对可靠性要求比较高的消息传递上。

kafka是Linkedin于2010年12月份开源的消息发布订阅系统,它主要用于处理活跃的流式数据,大数据量的数据处理上。

1)在架构模型方面，

RabbitMQ遵循AMQP协议，RabbitMQ的broker由Exchange,Binding,queue组成，其中exchange和binding组成了消息的路由键；客户端Producer通过连接channel和server进行通信，Consumer从queue获取消息进行消费（长连接，queue有消息会推送到consumer端，consumer循环从输入流读取数据）。rabbitMQ以broker为中心；有消息的确认机制。

kafka遵从一般的MQ结构，producer，broker，consumer，以consumer为中心，消息的消费信息保存的客户端consumer上，consumer根据消费的点，从broker上批量pull数据；无消息确认机制。

2)在吞吐量，

kafka具有高的吞吐量，内部采用消息的批量处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高。

rabbitMQ在吞吐量方面稍逊于kafka，他们的出发点不一样，rabbitMQ支持对消息的可靠的传递，支持事务，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。

3)在可用性方面，

rabbitMQ支持miror的queue，主queue失效，miror queue接管。

kafka的broker支持主备模式。

4)在集群负载均衡方面，

kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上。

rabbitMQ的负载均衡需要单独的loadbalancer(HAProxy等)进行支持。

# 缓存机制

## 1.Memcached 和 Redis区别？

Memcached原理：

Redis原理：

# 数据库

## 关系型数据库：MySQL/SQLServer/Oracle?

1.数据库的索引原理，b+树原理，trie树引申，二叉查找树原理？

## NoSql数据库：Redis/Mongodb/HBase?

Redis:

Redis官网和社区：

1. 官网：<http://redis.io/>
2. Redis命令：<http://doc.redisfans.com/>

1.原理：

1. Redis是一个基于key-value的高速缓存系统，类似于memcached，但是支持更复杂的数据结构List、Set、Sorted Set，并且有持久化的功能。

2. 应用场景：

1.key value store.是一个以key-value形式存储的数据库，定位直指MySQL，用来作为唯一的存储系统。

2.memory cache.是一个把数据存储在内存中的高速缓存，用来在应用和数据库间提供缓冲，替代memcachd。

3.data structrue server.把它支持对复杂数据结构的高速操作作为卖点，提供某些特殊业务场景的计算和展现需求。比如排行榜应用，Top 10之类的。

目前更多的人还是把它定位为一个memcached的升级版，提供更多的数据结构操作，仍然是一个cache。

MongoDB:

Mongodb官网和社区：

1. 官网：<https://www.mongodb.com/>
2. 中文社区：<http://www.mongoing.com/>

1.原理：

1. 基于分布式的文件存储数据库
2. 面向集合（collection-oriented）文档的数据库,数据被分组存在Collection中，每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的Document文档。集合的概念类似关系型数据库（RDBMS）里的表（table），不同的是它不需要定义任何模式（schema)
3. 模式自由（schema-free)，意味着对于存储在mongodb数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。
4. 存储在集合Collection中的文档，被存储为Key-Value对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为BSON（Binary JSON）。

2.优点：

1.文件存储格式为Bson，使用易于掌握和理解的Json风格语法。相对Json来说，Bson拥有更好的性能，主要表现为更快的遍历速度、操作更简易、增加了额外的  
数据类型。  
 2.模式自由，支持嵌入子文档和数组，无需事先创建数据结构，属于逆规范化的数据模型，有利于提高查询速度。  
 3.动态查询，支持丰富的查询表达式，使用Json形式的标记，可轻易查询文档中内嵌的对象和数组及子文档。  
 4.完整的索引支持，包括文档内嵌对象和数据，同时还提供了全文索引方式，MongoDB的查询优化器会分析查询表达式，并生成一个高效的查询计划。  
 5.使用高效的二进制数据存储，适合存储大型对象（如高清图片、视频等）。  
 6.支持多种复制模式，提供冗余及自动故障转移。支持Master-Slave、Replica Pairs/Replica Sets、有限Master-Master模式。  
 7.支持服务端脚本和Map/Reduce，可以实现海量数据计算，即实现云计算功能。  
 8.性能高、速度快。在多数场合，其查询速度对于MySQL要快的多，对于CPU占用非常小。部署很简单，几乎是零配置。  
 9.自动处理碎片，支持自动分片功能实现水平扩展的数据库集群，可以动态添加或移除节点。  
 10.内置GridFS，支持海量存储。  
 11.可通过网络访问，采用高效的MongoDB网络协议，在性能方面要优于http或Rest协议。  
 12.第三方支持丰富，MongoDB社区活跃，越来越多的公司和网站在生产环境中使用MongoDB进行技术架构优化，同时由10gen公司官方提供强大技术支持。

3.缺点：

1. mongodb不支持真正意义上的事务
2. mongodb占用空间过大。
3. 空间的预分配：为避免形成过多的硬盘碎片，mongodb每次空间不足时都会申请生成一大块的硬盘空间，而且申请的量从64M、128M、256M那样的指数递增，直到2G（32位系统限制，64位没有限制）为单个文件的最大体积。随着数据量的增加，你可以在其数据目录里看到这些整块生成容量不断递增的文件
4. 字段名所占用的空间：为了保持每个记录内的结构信息用于查询，mongodb需要把每个字段的key-value都以BSON的形式存储，如果 value域相对于key域并不大，比如存放数值型的数据，则数据的overhead是最大的。一种减少空间占用的方法是把字段名尽量取短一些，这样占用空间就小了，但这就要求在易读性与空间占用上作为权衡了。我曾建议作者把字段名作个index，每个字段名用一个字节表示，这样就不用担心字段名取多长了。但作者的担忧也不无道理，这种索引方式需要每次查询得到结果后把索引值跟原值作一个替换，再发送到客户端，这个替换也是挺耗费时间的。现在的实现算是拿空间来换取时间吧
5. 删除记录不释放空间：这很容易理解，为避免记录删除后的数据的大规模挪动，原记录空间不删除，只标记“已删除”即可，以后还可以重复利用
6. 可以定期运行db.repairDatabase()来整理记录，但这个过程会比较缓慢

3.MongoDB没有如MySQL那样成熟的维护工具，这对于开发和IT运营都是个值得注意的地方

4.使用场景：

MongoDB的主要目标是在键/值存储方式（提供了高性能和高度伸缩性）以及传统的RDBMS系统（丰富的功能）架起一座桥梁，集两者的优势于一身。  
（1）网站数据：MongoDB非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。  
（2）缓存：由于性能很高，MongoDB也适合作为信息基础设施的缓存层。在系统重启之后，由MongoDB搭建的持久化缓存层可以避免下层的数据源过载。  
（3）大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。  
（4）高伸缩性的场景：MongoDB非常适合由数十或数百台服务器组成的数据库。MongoDB的路线图中已经包含对MapReduce  
引擎的内置支持。  
（5）用于对象及JSON数据的存储：MongoDB的Bson数据格式非常适合文档化格式的存储及查询。2.

Mongodb对比 Redis:





Mongodb对比HBase:

1.Mongodb bson文档型数据库，整个数据都存在磁盘中，hbase是列式数据库，集群部署时每个familycolumn保存在单独的hdfs文件中。

2.Mongodb 主键是“\_id”,主键上面可以不建索引,记录插入的顺序和存放的顺序一样，hbase的主键就是row key，可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在hbase内部，row key保存为字节数组。存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。

字典序对int排序的结果是1,10,100,11,12,13,14,15,16,17,18,19,2,20,21,…,9,91,92,93,94,95,96,97,98,99。要保持整形的自然序，行键必须用0作左填充。

3.Mongodb支持二级索引，而hbase本身不支持二级索引

4.Mongodb支持集合查找，正则查找，范围查找，支持skip和limit等等，是最像mysql的nosql数据库，而hbase只支持三种查找：通过单个row key访问，通过row key的range，全表扫描

5.mongodb的update是update-in-place，也就是原地更新，除非原地容纳不下更新后的数据记录。而hbase的修改和添加都是同一个命令：put，如果put传入的row key已经存在就更新原记录,实际上hbase内部也不是更新，它只是将这一份数据已不同的版本保存下来而已，hbase默认的保存版本的历史数量是3。

6.mongodb的delete会将该行的数据标示为已删除，因为mongodb在删除记录时并不是真把记录从内存或文件中remove，而是将该删除记录数据置空（写0或特殊数字加以标识）同时将该记录所在地址放到一个list列表“释放列表”中，这样做的好就是就是如果有用户要执行插入记录操作时，mongodb会首先从该“释放列表”中获取size合适的“已删除记录”地址返回，这种方法会提升性能（避免了malloc内存操作），同时mongodb也使用了bucket size数组来定义多个大小size不同的列表，用于将要删除的记录根据其size大小放到合适的“释放列表”中。Hbase的delete是先新建一个tombstonemarkers，然后读的时候会和tombstonemarkers做merge，在 发生major compaction时delete的数据记录才会真真删除。

7.mongodb和hbase都支持mapreduce，不过mongodb的mapreduce支持不够强大，如果没有使用mongodb分片，mapreduce实际上不是并行执行的

8.mongodb支持shard分片，hbase根据row key自动负载均衡，这里shard key和row key的选取尽量用非递增的字段，尽量用分布均衡的字段，因为分片都是根据范围来选择对应的存取server的，如果用递增字段很容易热点server的产生，由于是根据key的范围来自动分片的，如果key分布不均衡就会导致有些key根本就没法切分，从而产生负载不均衡。

9.mongodb的读效率比写高，hbase默认适合写多读少的情况，可以通过hfile.block.cache.size配置，该配置storefile的读缓存占用Heap的大小百分比，0.2表示20%。该值直接影响数据读的性能。如果写比读少很多，开到0.4-0.5也没问题。如果读写较均衡，0.3左右。如果写比读多，果断默认0.2吧。设置这个值的时候，你同时要参考hbase.regionserver.global.memstore.upperLimit，该值是memstore占heap的最大百分比，两个参数一个影响读，一个影响写。如果两值加起来超过80-90%，会有OOM的风险，谨慎设置。

10.hbase采用的LSM思想(Log-Structured Merge-Tree)，就是将对数据的更改hold在内存中，达到指定的threadhold后将该批更改merge后批量写入到磁盘，这样将单个写变成了批量写，大大提高了写入速度，不过这样的话读的时候就费劲了，需要merge disk上的数据和memory中的修改数据，这显然降低了读的性能。mongodb采用的是mapfile+Journal思想，如果记录不在内存，先加载到内存，然后在内存中更改后记录日志，然后隔一段时间批量的写入data文件，这样对内存的要求较高，至少需要容纳下热点数据和索引。

# 网络协议

## HTTP ?

## TCP ?

## UDP ?

## MQTT(物联网协议) ?