### 1、 Statement

1、Statement 是 Java 执行数据库操作的一个重要接口，用于在已经建立数据库连接的基础上，向数据库发送要执行的SQL语句。Statement对象，用于执行不带参数的简单SQL语句。 子接口有 CallableStatement和 PreparedStatement

A: Statement 对象用于执行不带参数的简单 SQL 语句。

B: PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句。

C: CallableStatement 对象用于执行对数据库已存在的存储过程的调用。

### 2、HashMap、HashTable和ConcurrentHashMap的区别？

HashMap是线程不安全的，ConcurrentHashMap和HashTable都是线程安全的。

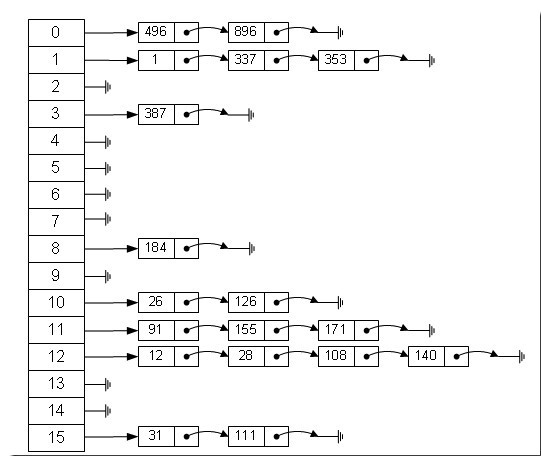
1、HashMap是非线程安全的，HashTable是线程安全的。

2、HashMap的键和值都允许有null值存在，而HashTable则不行。

3、因为线程安全的问题，HashMap效率比HashTable的要高。

**HashMap的内部存储结构**

Java中数据存储方式最底层的两种结构，一种是数组，另一种就是链表，数组的特点：连续空间，寻址迅速，但是在删除或者添加元素的时候需要有较大幅度的移动，所以查询速度快，增删较慢。而链表正好相反，由于空间不连续，寻址困难，增删元素只需修改指针，所以查询慢、增删快。哈希表具有较快（常量级）的查询速度，及相对较快的增删速度，所以很适合在海量数据的环境中使用



哈希表是由数组+链表组成的，一个长度为16的数组中，每个元素存储的是一个链表的头结点。那么这些元素是按照什么样的规则存储到数组中呢。一般情况是通过hash(key)%len获得，也就是元素的key的哈希值对数组长度取模得到。

**HashMap：（父类AbstractMap）**

数据结构：hash表。Entry类型的数组代表每个桶，真正存储元素的是一个Entry的链表，每个桶存一个Entry类型的引用（指针），负载因子为0.75，初始化length为16

put：通过hash算法，找到key所对应的桶，从链表头插入

get：通过hash算法，找到key所对应的桶，然后从第一个元素顺着链表一个一个比较，如果找到返回，否则返回null

\*HashMap允许null的key或者value

\*线程不安全

**HashTable：(父类Dictionary)**

数据结构：hash表。Entry类型的数组代表每个桶，真正存储元素的是一个Entry的链表，每个桶存一个Entry类型的引用（指针），负载因子为0.75，初始化length为16

put：通过hash算法，找到key所对应的桶，从链表头插入，这个过程中是通过synchronized关键字来对put方法加锁，达到线程安全

get：通过hash算法，找到key所对应的桶，然后从第一个元素顺着链表一个一个比较，如果找到返回，否则返回null

\***HashMap不允许null的key或者value**

\*线程安全

**ConcurrentHashMap：(较前两种多了段，分段加锁的概念)**

数据结构：首先是段数组，每个段对应一个hash表。Entry类型的数组代表每个桶，真正存储元素的是一个Entry的链表，每个桶存一个Entry类型的引用（指针）

**通过锁分离达到线程安全**，把整个hash表分成几段，每段一个锁，每段其实就是一个小的hash表，这样就会出现只要两个线程没有争用一个段的数据，就可以同时访问修改hash表，并发进行，从而效率更高效。但是也会有对整个表锁的操作，比如size()等，这个时候锁的顺序就会很重要了，以免发生死锁

put：通过hash算法，找到段，然后找到key所对应的桶，从链表头插入，这个过程中是通过分段加锁，达到线程安全

get：通过hash算法，找到段，找到key所对应的桶，然后从第一个元素顺着链表一个一个比较，如果找到返回，否则返回null

\*线程安全

ConcurrentHashMap基于concurrentLevel划分出了多个Segment来对key-value进行存储，从而避免每次锁定整个数组，在默认的情况下，**允许16个线程并发无阻塞的操作集合对象，尽可能地减少并发时的阻塞现象。**

### 3 、ArrayList、LinkedList、Vector的区别

List接口直接继承Collection接口，Collection接口继承Iterable接口

List特性：

1、可以存放同一种类型的元素。

2、内部维护元素之间的顺序，是有序集合。

3、元素是可以重复的

区别如下：

（1）ArrayList内部存储的数据结构是数组存储。数组的特点：元素可以快速访问。每个元素之间是紧邻的不能有间隔，缺点：数组空间不够元素存储需要扩容的时候会开辟一个新的数组把旧的数组元素拷贝过去，比较消性能。从ArrayList中间位置插入和删除元素，都需要循环移动元素的位置，因此数组特性决定了数组的特点：适合随机查找和遍历，不适合经常需要插入和删除操作。

（2）Vector内部实现和ArrayList一样都是数组存储，最大的不同就是它支持线程的同步，所以访问比ArrayList慢，但是数据安全，所以对元素的操作没有并发操作的时候用ArrayList比较快。

（3）LinkedList内部存储用的数据结构是链表。链表的特点：适合动态的插入和删除。访问遍历比较慢。另外不支持get，remove，insertList方法。可以当做堆栈、队列以及双向队列使用。LinkedList是线程不安全的。所以需要同步的时候需要自己手动同步，比较费事，可以使用提供的集合工具类实例化的时候同步：具体使用List<String> springokList=Collections.synchronizedCollection(new 需要同步的类)。

ArrayLsit和Vector的主要区别：

（1）同步性

（2）数据增长策略

ArrayList与Vector都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时，就需要增加ArrayList与Vector的存储空间，每次要增加存储空间时，不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要取得一定的平衡。Vector默认增长为原来两倍，而ArrayList的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的1.5倍）。

### 4、反射

Java通过反射机制，可以在程序运行时加载，和使用编译期完全未知的类，并且可以生成相关类对象的实例，从而可以生成相关类对象的实例，从而可以调用其方法或者改变某个属性值。

1、反射机制的概念

在Java中的反射机制是指在运行状态中，对于任意一个类都能够知道这个类所有的属性和方法，并且对于任意一个对象，都能够调用它的任意一个方法，这种动态获取信息以及动态调用对象方法的功能成为Java语言的反射机制。

2、反射应用的场合

对象的两种类型：编译时类型和运行时类型。

编译时的类型由声明对象时使用的类型来决定，运行时的类型由实际赋值给对象的类型来决定。程序在运行时还可能接收到外部传入的对象，该对象的编译时类型为Object,但是程序有需要调用该对象的运行时类型的方法，为了解决这些问题，程序需要在运行时发现对象和类的真实信息，然而，如果编译时根本无法预知该对象和类属于哪些类，程序只能依靠运行时信息来发现该对象和类的真实信息，此时就必须使用到反射了。

3、Java反射API

反射API用来生成JVM中的类、接口或者对象的信息

-Class类：反射的核心类，可以获取类的属性，方法等信息

-Field类：Java.lang.reflect包中的类，表示类的成员变量，可以用来获取和设置类之中的属性值。

-Method类：Java.lang.reflect包中的类，表示类的方法，它可以用来获取类中的方法信息或者执行方法。

-Constructor类：Java.lang.reflect包中的类，表示类的构造方法。

4、使用反射的步骤：

@获取想要操作的类的Class对象。

@调用Class类中的方法。

@使用反射API来操作这些信息。

### 5、1.8 1.9 2.0新特性

**JDK1.8新特性（2014）**

**1、接口允许默认方法**

Java8允许我们给接口添加一个非抽象的方法实现，只需使用default关键字即可，这个特征又叫做扩展方法。实现接口的类默认都拥有在接口中定义的缺省方法，当子类中没有覆盖缺省方法时，则对子类的该方法的调用将调用接口中的实现。

**2、Lambda表达式**

Java8中没必要使用传统的匿名对象，java8提供了更简洁的语法，通俗来讲，lambda就是一种没有名字的函数，使用这种表达式的好处是可以使得我们的代码更加简洁，但是也会造成程序刻度性大大降低，在一些情况下，也会造成代码效率差。

3、方法与构造函数引用

Java8允许你使用：：关键字来传递方法或者构造函数引用。

**4、函数式接口**

函数式接口(Functional Interface)就是一个有且仅有一个抽象方法，但是可以有多个非抽象方法的接口。函数式接口可以被隐式转换为lambda表达式。

5、**优化了HashMap以及ConcurrentHashMap**，将HashMap原来数组+链表的结构优化成了数组+链表+红黑树的结构，**减少了hash碰撞造成的链表长度过长，时间复杂度过高的问题**，ConcurrentHashMap则改进了原先的分段锁的方式，采用(瞬态)Transient volatile HashEntry<K,V>[] table来保存数据。

6、接口的默认方法无法在lambda表达式内部被访问

7、**Java8中支持多重注解，**允许我们把同一类型的注解多次使用，只需给该注解标注一下@Repeatable即可

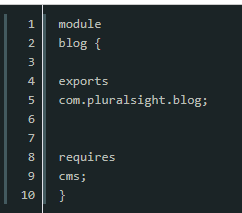
8、**PermGen空间被移除了，取而代之的是Metaspace**，JVM选项-XX：PermSize与-XX：MaxPermSize分别被-XX：MetaSpaceSize与-XX：MaxMetaspaceSize取代。

**JDK1.9新特性(2017)**

1、**java平台级模块系统**

Java9定义功能是一套全新的模块系统，当代码库越来越大，要面对两个基础问题，很难真正对代码进行封装，系统并没有对不同部分之间的依赖关系有个明确概念。

模块化的JAR文件都包含一个额外的模块描述器，在这个模块描述器中，对其他模块的依赖是通过”requires”来表示的，另外，“exports”语句控制着哪些包是可以被其他模块访问到的。



通过封装JDK的内部类，平台更安全，持续改进也更容易，当启动一个模块化应用时，JVM会验证是否所有的模块都能使用，这基于“requires”语句，模块允许你更好地强制结构化封装并明确依赖 。

2、 **jlink工具**

当使用具有显式依赖关系的模块和模块化的JDK时，可以使用JAVA9提供的jlink工具创建针对应用程序进行优化的最小运行时映像而不需要使用完全加载JDK安装版本。

**3、使用JShell提供交互式编程环境**

许多语言已经具有交互式编程环境，你可以从控制台启动jshell,并直接启动输入和执行java代码，它可以即时反馈

4、私有的接口方法

Java8为我们带来了接口的默认方法，使得接口现在也可以包含行为，而不仅仅是方法签名，但默认方法不能是私有的，使用java9，你可以向接口添加私有辅助方法来解决此问题。

5、**新的HTTP调用方法**

Java9中有**新的方式（新的 HttpClient API）来处理HTTP调用**，这个迟到的特性用于代替老旧的‘HttpURLConnection’API，**并提供WebSocket和HTTP/2.0的支持**，HttpClient 还提供了新的 API 来处理 HTTP/2.0 的特性，比如流和服务端推送。

6、**改进的javadoc**

Javadoc现在支持在API文档中进行搜索，另外，javadoc的输出现在兼容H5标准，此外，每个javadoc页面都包含有关jdk模块类或接口来源的信息。

**JDK1.9新特性(2017)**

1、**局部变量类型推断**

局部变量类型推断将引入“var”关键字，也就是你可以随意定义变量而不必指定变量类型。

2、**GC改进和内存管理**

Java9中有两个JEP专门用于改进当前的垃圾收集元素。

第一个垃圾收集器接口（JEP304），引入一个纯净的垃圾收集器接口，帮助改进不同垃圾收集器的源代码隔离。

第二个JEP（307）,是针对G1的并行完全GC，其重点在于通过完全GC并行来改善G1最坏情况的等待时间。

3、**线程本地握手**

**JDK10将引入一种在线程上执行回调的新方法**，因此这将会很方便能停止单个线程而不是停止全部线程或者一个都不停。

4、**备用内存设备上的堆分配**

允许HotSpotVM在备用内存设备上分配Java对象堆内存，该内存设备将由用户指定。

### 6、Java注解

未来的开发模式都是基于注解的，JPA是基于注解的，Spring2.5、Hibernate3.x以上都是基于注解的,现在的Struts2有一部分是基于注解的，注解是一种趋势，注解是JDK1.5之后才有的新特性。

JDK1.5之后内部提供了三个注解：（一个注解就是一个类）

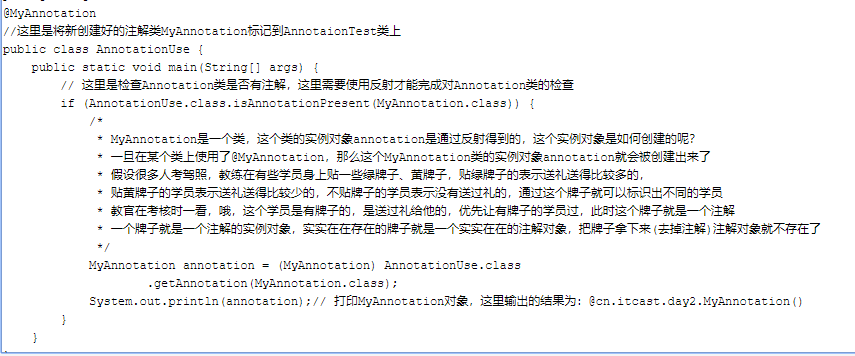
@Deprecated 意思是“**废弃的，过时的**”

@Override 意思是“**重写、覆盖**”

@SuppressWarnings 意思是“**压缩警告**”

注解相当于一种标记，在程序中加入注解就等于为程序打上某种标记，没有加，就等于没有任何标记，以后，javac编译器、开发工具和其他程序可以通过反射来了解你的类及各种元素上有无任何标记，看你的程序有什么标记，就去干相应的事，标记可以加载包、类、属性、方法、方法的参数以及局部变量上。

自定义注解MyAnnotation，用反射测试annotationUse定义上是否有@MyAnnotation



注解的作用？

1、可以理解为插件，是代码级别的插件。

2、java注解是附加在代码中的一些元信息，用于在编译、运行时经行解析使用时起到说明、配置的功能。

3、注解的解析完全依赖于反射。。

Java的反射机制就是通过类的对象，获取该对象的类的相关信息，类的相关信息包括用了什么注解，类的名称，各成员变量的名称等。

**2什么时候用到注解？我们一般用不到自定义注解**，我们平时别人开发好的框架如Hibernate、Spring、Struts等时，只需要按照框架的规则在代码里定义注解即可，而不会在代码里调用和操作注解。只有在自己动手写类似于Hibernate框架的时候，会用到自定义注解。**这也是为什么我们一般用不到java反射**，因为只有在使用自定义注解的时候，才会用到java反射，**而我们平时连自定义注解都用不到**。

### 7、动态代理

AOP的原理就是java动态代理机制，

Java动态代理机制中，有一个重要的接口InvocationHandler和一个重要的类 Proxy，是必须用到的

每一个动态代理类都必须显现InvocationHandler接口，并且每个代理类的实例都关联了一个Handler，当我们通过代理对象调用一个方法的时候，这个方法的调用就会被转发为由InvocationHandler接口的invoke方法来进行调用

**Proxy这个类的作用就是用来动态创建一个代理对象的类。**

作用：

1、主要用来做方法的增强，让你在不修改源码的情况下，增强一些方法，在方法的执行前后做任何你想做的事情，因为在InvocationHandler的invoke方法中，你可以直接获取正在调用方法对象，具体应用，比如可以添加调用日志、做事务控制等

2、远程调用，

比如现在有Java接口，这个接口的实现部署在其它服务器上，在编写客户端代码的时候，没办法直接调用接口方法，因为接口是不能直接生成对象的，这个时候就可以考虑代理模式（动态代理）了。通过Proxy.newProxyInstance代理一个该接口对应的InvocationHandler对象，然后在InvocationHandler的invoke方法内封装通讯细节就可以了，比如RMII

### 8、 SVN GIT优缺点

**1．SVN优缺点（集中式版本控制）**

优点：

1、 管理方便，逻辑明确，符合一般人思维习惯。

2、 易于管理，集中式服务器更能保证安全性。

3、 代码一致性非常高。

4、 适合开发人数不多的项目开发。

缺点：

1、 服务器压力太大，数据库容量暴增。

2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。

3、 不适合开源开发（开发人数非常非常多）

**2、Git优缺点（分布式版本控制）**

优点：

1、适合分布式开发，强调个体

2、公共服务器压力和数据量都不会太大。

3、灵活速度快

4、任意两个开发者之间可以很容易解决冲突

缺点：

学习周期长

不符合常规思想

代码保密性差

### 9、 多态

分为两种：方法覆盖和方法重载

1、方法重载：指定我们可以定义一些名称相同的方法，通过不同的输入参数来区分这些方法，然后调用的时候，VM就会根据不同的参数样式，来选择合适的方法执行。

2、方法覆盖;Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类的完全相同的方法给覆盖了，父类引用指向子类应用。

### 10、String、StringBuilder、StringBuffer

1、运行速度

StringBuilder > StringBuffer > String，原因，String为字符串常量，而StringBuilder和StringBuffer均为字符串变量，即String对象一旦创建之后对象是不可更改的，但后两者对象是变量，是可以更改的。Java中对String对象进行的操作实际上是一个不断创建新的对象并且将旧的对象回收的一个过程，所以执行速度很慢。

2、线程安全

StringBuilder是线程不安全的，而StringBuffer是线程安全的，StringBuffer中很多方法可以带有synchronized关键字，所以可以保证线程是安全的，但StringBuilder的方法则没有该关键字，所以不能保证线程安全，有可能会出现一些错误的操作

3、　String：适用于少量的字符串操作的情况

StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况

StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况