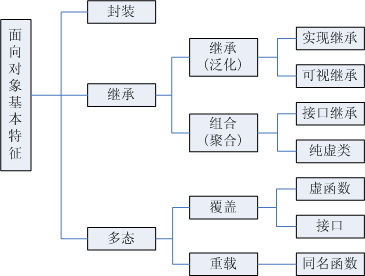
**继承、多态、封装**



**封装**

封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

**继承**

面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。 继承现有类 + 扩展

继承概念的实现方式有三类：实现继承、接口继承和可视继承。

**多态**

多态性（polymorphisn）是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

覆写（Override）的两个函数的函数特征相同，重载（Overload）的两个函数的函数名虽然相同，但函数特征不同。

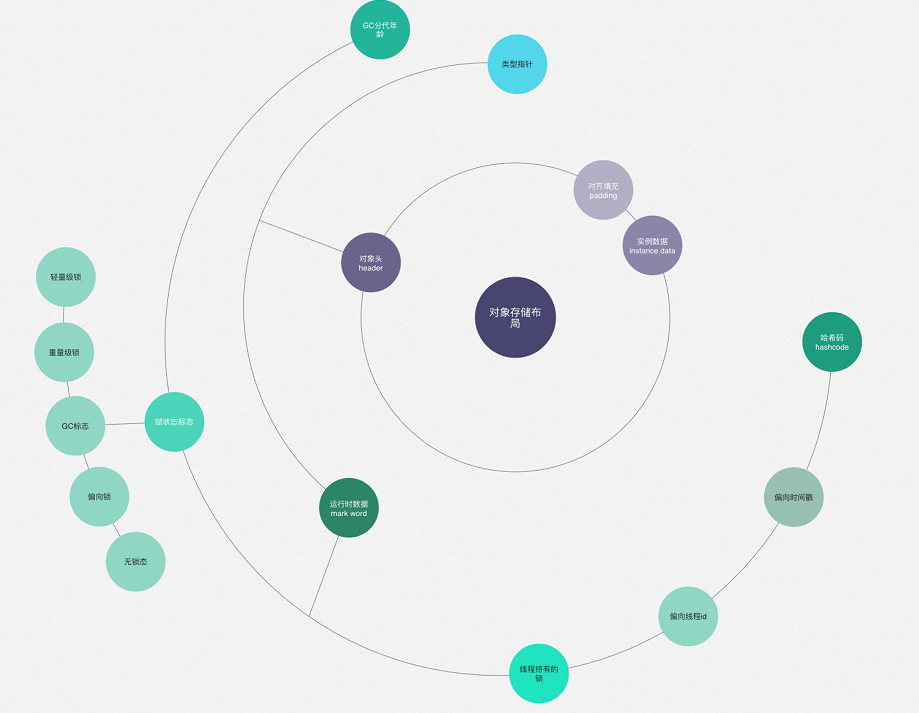
**单一职责、开闭、里氏替换**

**单一职责原则：**   
定义**：**就一个类而言，应该仅有一个引起它变化的原因。   
**软件设计真正要做的许多内容，就是发现职责并把这些职责相互分离。**要去判断是否应该分离出类来，也不难，那就是如果你能够想到多于一个的动机去改变一个类，那么这个类就具有多于一个的职责，就应该考虑**类的职责分离**。

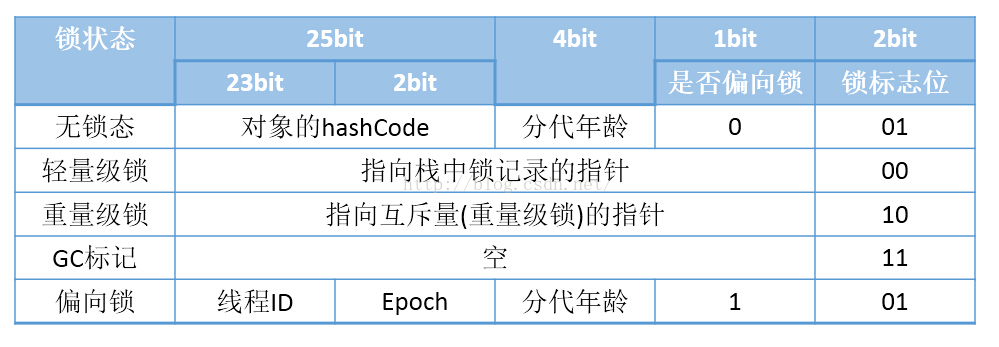
**开放封闭原则：**   
定义**：**是说软件实体（类、模块、函数等等）应该可以扩展，但是不可以修改。**开闭原则是面向对象设计的核心所在。**即对扩展是开放的，对于修改是封闭的。

**里氏代换原则（CSP）：**   
子类型必须能够替换掉它们的父类型。只有当子类可以替换掉父类，软件单位的功能不受到影响时，父类才能真正被复用，而子类也能够**在父类的基础上增加行为。**

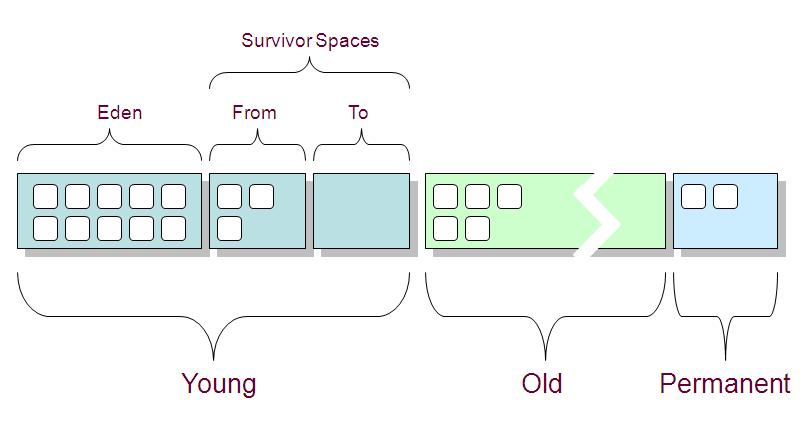
**Java对象在JVM中的头信息**



**锁状态详情：**



**GC示意图：**



**Object类有哪些方法，各个方法的含义与作用**

registerNatives()

为了加深对java本地方法的理解，在网上找到了该方法的C源码部分，如下：

static JNINativeMethod methods[] = {

{"hashCode", "()I", (void \*)&JVM\_IHashCode},

{"wait", "(J)V", (void \*)&JVM\_MonitorWait},

{"notify", "()V", (void \*)&JVM\_MonitorNotify},

{"notifyAll", "()V", (void \*)&JVM\_MonitorNotifyAll},

{"clone", "()Ljava/lang/Object;", (void \*)&JVM\_Clone},

};

JNIEXPORT void JNICALL

Java\_java\_lang\_Object\_registerNatives(JNIEnv \*env, jclass cls)

{

(\*env)->RegisterNatives(env, cls,

methods, sizeof(methods)/sizeof(methods[0]));

}

对几个本地方法进行注册(也就是初始化java方法映射到C的方法)。

clone() 创建并返回此对象的一个副本。

equals(Object) 判断其他某个对象是否与此对象“相等”。

finalize() 垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法

getClass() 返回此 Object 的运行时类

hashCode() 获取该对象的哈希码值，默认类名+hashcode

notify() 唤醒此对象监视器上等待的单个线程

notifyAll() 唤醒此对象监视器上等待的所有线程

toString() 返回该对象的字符串内容信息

wait() 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待

wait(long) 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待

wait(long, int) 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待

**List （ArrayList，LinkedList）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **add** | **delete** | **insert** | **index取值** | **iterator取值** |
| **ArrayList/Vector/Stack** | **好** | **差** | **差** | **极优** | **极优** |
| **LinkedList** | **好** | **好** | **好** | **差** | **极优** |

1．对ArrayList和LinkedList而言，在列表末尾增加一个元素所花的开销都是固定的。在ArrayList的中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动，开销较大；而在LinkedList的中间插入或删除一个元素的开销是固定的。

2．LinkedList不支持高效的随机元素访问。

3．ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间

可以这样说：当操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能；当你的操作是在一列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用LinkedList了。

案例地址：https://github.com/ben201708/learn\_java/blob/master/src/com/company/week01/testList.java

**Map (HashMap，LinkedHashMap，TreeMap)**

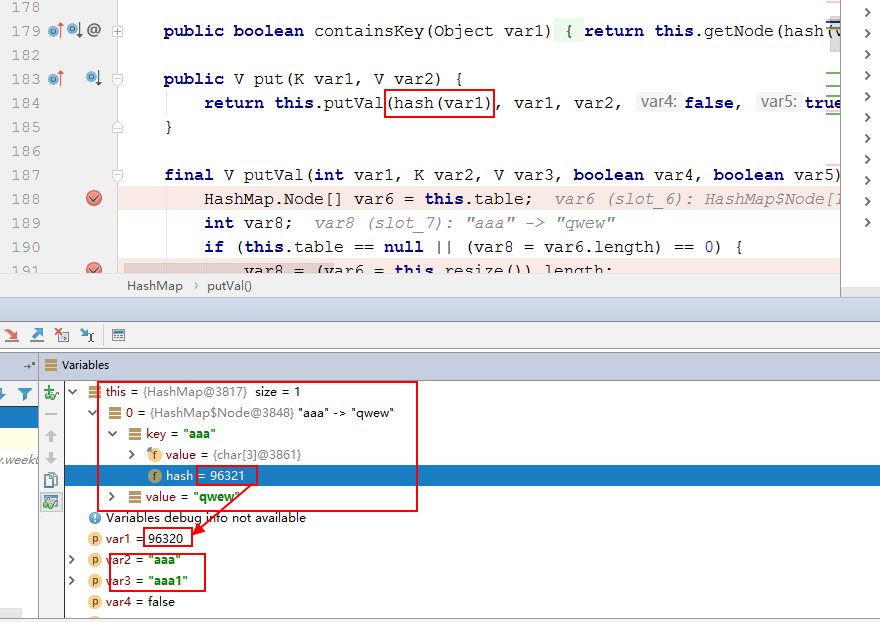
Map主要用于存储健值对，根据键得到值，因此不允许键重复(重复了覆盖了),但允许值重复。

Hashmap 是一个最常用的Map,它根据键的HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。HashMap最多只允许一条记录的键为Null;允许多条记录的值为 Null;HashMap不支持线程的同步，即任一时刻可以有多个线程同时写HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections的synchronizedMap方法使HashMap具有同步的能力，或者使用ConcurrentHashMap。

LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比HashMap慢，不过有种情况例外，当HashMap容量很大，实际数据较少时，遍历起来可能会比LinkedHashMap慢，因为LinkedHashMap的遍历速度只和实际数据有关，和容量无关，而HashMap的遍历速度和他的容量有关。

TreeMap实现SortMap接口，能够把它保存的记录根据键排序,默认是按键值的升序排序，也可以指定排序的比较器，当用Iterator 遍历TreeMap时，得到的记录是排过序的。

**Hash冲突：**



**JDK8 的HashMap，链表到了8就需要变成颗红黑树了**

