# Java理解概念

## Java的hashMap概念

在Java中，关于HashMap类的描述，以下错误的是

正确答案: B   你的答案: C (错误)

HashMap使用键/值得形式保存数据

HashMap 能够保证其中元素的顺序

HashMap允许将null用作键

HashMap允许将null用作值

* Hash家族嘛,睾龟的很,高逼格,必须一次性两个值存储,就是所谓的**键值对**.

但是呢,Hash家族内部分为了几个小家族,分别是**HashMap,Hashtable,TreeMap**.

这几个家族呢,对键值对能不能存储null这种不是很安全的"买卖"有不一样的行动.

其中的HashMap家族与Hashtable、TreeMap不同,**认为没有风险就没有利润**!于是乎,**准许自己的键值对都可以为null!**

Hashtable与TreeMap一看SUN国王居然默许了HashMap的冒险行为,使得HashMap家族的利润大大增加,这两个家族也不甘寂寞,于是乎也就**允许了自己的键值对可以为""**,但是不能触碰null的界限.

* **关于HashMap的一些说法：**

**a)** **HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。HashMap的底层结构是一个数组，数组中的每一项是一条链表。**

**b)** **HashMap的实例有俩个参数影响其性能： “初始容量” 和 装填因子。**

**c)** **HashMap实现不同步，线程不安全。  HashTable线程安全**

**d)** **HashMap中的key-value都是存储在Entry中的。**

**e)** **HashMap可以存null键和null值， 不保证元素的顺序恒久不变 ，它的底层使用的是数组和链表，通过hashCode()方法和equals方法保证键的唯一性**

**f)** **解决冲突主要有三种方法：定址法，拉链法，再散列法。HashMap是采用拉链法解决哈希冲突的。**

**注： 链表法是将相同hash值的对象组成一个链表放在hash值对应的槽位；**

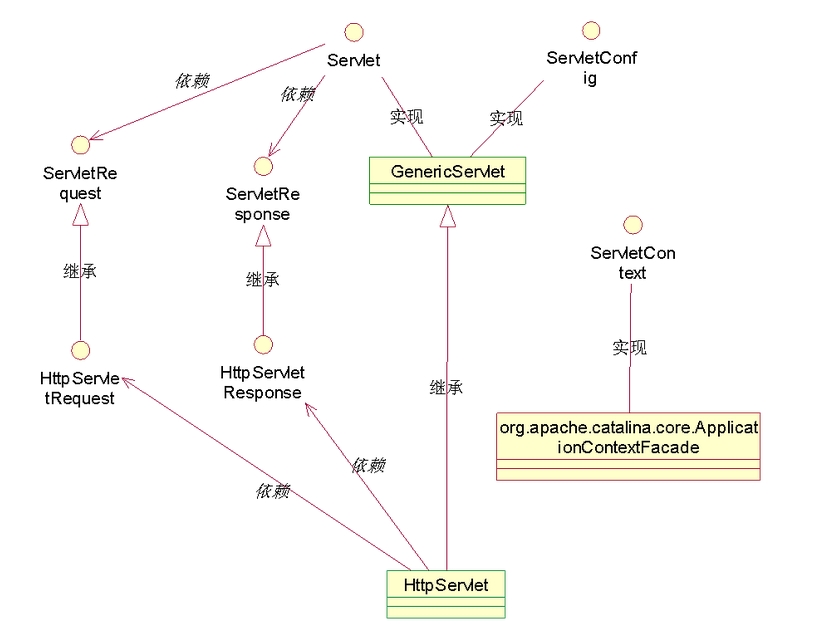
**用开放定址法解决冲突的做法是：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。 沿此序列逐个单元地查找，直到找到给定 的关键字，或者碰到一个开放的地址(即该地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。**

**拉链法解决冲突的做法是： 将所有关键字为同义词的结点链接在同一个单链表中 。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数 组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。在拉链法中，装填因子α可以大于1，但一般均取α≤1。拉链法适合未规定元素的大小。**

## 2.java跨平台的原因

**Java的跨平台特性是因为JVM的存在， 它可以执行.class字节码文件，而不是.java源代码。**   
**在Java平台上被编译为体系结构中立的字节码格式（后缀为class的文件）, 然后可以在实现这个Java平台的任何系统中运行**。这种途径适合于[异构](http://www.so.com/s?q=%E5%BC%82%E6%9E%84&ie=utf-8&src=internal_wenda_recommend_textn)的网络环境和[软件](http://www.so.com/s?q=%E8%BD%AF%E4%BB%B6&ie=utf-8&src=internal_wenda_recommend_textn)的分发。 **跨平台性**，Java语言是可移植的。**这种**[**可移植性**](http://www.so.com/s?q=%E5%8F%AF%E7%A7%BB%E6%A4%8D%E6%80%A7&ie=utf-8&src=internal_wenda_recommend_textn)**来源于体系结构中立性，另外，Java还严格规定了各个基本数据**[**类型**](http://www.so.com/s?q=%E7%B1%BB%E5%9E%8B&ie=utf-8&src=internal_wenda_recommend_textn)**的长度。Java系统本身也具有很强的可移植性，Java**[**编译器**](http://www.so.com/s?q=%E7%BC%96%E8%AF%91%E5%99%A8&ie=utf-8&src=internal_wenda_recommend_textn)**是用Java实现的，Java的**[**运行环境**](http://www.so.com/s?q=%E8%BF%90%E8%A1%8C%E7%8E%AF%E5%A2%83&ie=utf-8&src=internal_wenda_recommend_textn)**是用ANSI C实现的。**

## 3.servletConfig的接口默认从哪里实现？



## 4.内部类



## 5.java抽象类

A选项中，抽象类不能实例化，这句是正确的。后面，抽象类不能申明为对象，是错误的。java中，接口和抽象类，都可以声明为对象，只需要在实例化的时候，用一个该接口/抽象类的实现类实例化就行了。即：

interface / abstract class obj = new class();其中，new后面的class是前面接口/抽象类的实现类。

B选项中，接口是一种特殊的抽象类。其中，成员变量都是默认public static final修饰的，方法都是public abstract修饰的，并且除了default和static的以外，只有声明，没有方法体。

C选项中，抽象类的方法，可以是抽象abstract的，也可以是普通的。就算全部是普通方法，也可以用abstract修饰。

D选项中，导出类，及子类，抽象类的子类可以是抽象的，也可以是普通的。

## 6.以下是object类的方法的是

#### 1．clone方法

保护方法，实现对象的浅复制，只有实现了Cloneable接口才可以调用该方法，否则抛出CloneNotSupportedException异常。

#### 2．getClass方法

final方法，获得运行时类型。

#### 3．toString方法

该方法用得比较多，一般子类都有覆盖。

#### 4．finalize方法

该方法用于释放资源。因为无法确定该方法什么时候被调用，很少使用。

#### 5．equals方法

该方法是非常重要的一个方法。一般equals和==是不一样的，但是在Object中两者是一样的。子类一般都要重写这个方法。

#### 6．hashCode方法

该方法用于哈希查找，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hash- Code()==obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

#### 7．wait方法

wait方法就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

（1）其他线程调用了该对象的notify方法。

（2）其他线程调用了该对象的notifyAll方法。

（3）其他线程调用了interrupt中断该线程。

（4）时间间隔到了。

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常。

#### 8．notify方法

该方法唤醒在该对象上等待的某个线程。

#### 9．notifyAll方法

该方法唤醒在该对象上等待的所有线程。

## 7.静态和构造的执行顺序

子类A继承父类B, A a = new A(); 则父类B构造函数、父类B静态代码块、父类B非静态代码块、子类A构造函数、子类A静态代码块、子类A非静态代码块 执行的先后顺序是？

父类Ｂ静态代码块->子类Ａ静态代码块->父类Ｂ非静态代码块->父类Ｂ构造函数->子类Ａ非静态代码块->子类Ａ构造函数