**阅读JDK文档及源代码并简要回答以下问题。**

注：JDKAPI文档（<http://docs.oracle.com/javase/8/docs/api/index.html>或从<http://cf.pku.cn/tds/java>下载chm格式的）

JDK的源代码（一般在 C:\Program Files\Java\jdk1.8.0\src.zip）中

-----java.lang.Object类-----

1. 其equals与==有没有差别？

==判断是否是同一个对象

equals判断对象是否相等。不同对象可能相等。

1. hashCode及clone前面有个什么修饰语？

Native，Native方法的实现由非Java语言实现。

1. 其toString()返回什么？

public String toString() {

return getClass().getName() + "@" + Integer.toHexString(hashCode());

}

类名@hashCode的十六进制表示

1. 其构造函数及finalize()做了什么？

构造函数和finalize()是空方法，分别在创建对象和垃圾回收时调用

-----java.lang.Class类-----

1. 为什么说Class类不能被继承？

Class类之前有final修饰符

1. 为什么说我们不能 new Class()?

Class的构造方法之前有private修饰符，无法从外部调用

1. forName()中执行了什么流程？

public static Class<?> forName(String className)

throws ClassNotFoundException {

Class<?> caller = Reflection.getCallerClass();

return forName0(className, true, ClassLoader.getClassLoader(caller), caller);

}

private static native Class<?> forName0(String name, boolean initialize,

ClassLoader loader,

Class<?> caller)

throws ClassNotFoundException;

根据给定的类名参数className，查找与className相对应的Class实例，然后加载、连接该实例对象，之后返回这个Class实例

1. toString()返回什么？

public String toString() {

return (isInterface() ? "interface " : (isPrimitive() ? "" : "class "))

+ getName();

}

如果是接口，返回 interface + 名称

如果是基本类型，直接返回类名

否则返回Class + 名称

1. newInstance()中执行了什么？

public T newInstance()

throws InstantiationException, IllegalAccessException

{

if (System.getSecurityManager() != null) {

checkMemberAccess(Member.PUBLIC, Reflection.getCallerClass(), false);

}

// NOTE: the following code may not be strictly correct under

// the current Java memory model.

// Constructor lookup

if (cachedConstructor == null) {

if (this == Class.class) {

throw new IllegalAccessException(

"Can not call newInstance() on the Class for java.lang.Class"

);

}

try {

Class<?>[] empty = {};

final Constructor<T> c = getConstructor0(empty, Member.DECLARED);

// Disable accessibility checks on the constructor

// since we have to do the security check here anyway

// (the stack depth is wrong for the Constructor's

// security check to work)

java.security.AccessController.doPrivileged(

new java.security.PrivilegedAction<Void>() {

public Void run() {

c.setAccessible(true);

return null;

}

});

cachedConstructor = c;

} catch (NoSuchMethodException e) {

throw (InstantiationException)

new InstantiationException(getName()).initCause(e);

}

}

Constructor<T> tmpConstructor = cachedConstructor;

// Security check (same as in java.lang.reflect.Constructor)

int modifiers = tmpConstructor.getModifiers();

if (!Reflection.quickCheckMemberAccess(this, modifiers)) {

Class<?> caller = Reflection.getCallerClass();

if (newInstanceCallerCache != caller) {

Reflection.ensureMemberAccess(caller, this, null, modifiers);

newInstanceCallerCache = caller;

}

}

// Run constructor

try {

return tmpConstructor.newInstance((Object[])null);

} catch (InvocationTargetException e) {

Unsafe.getUnsafe().throwException(e.getTargetException());

// Not reached

return null;

}

}

进行安全检查，以无参方式创造一个当前类的实例

-----java.lang.String类-----

1. 为什么说String是immutable的？为什么不要多次循环中使用+=？ 其value是加了哪个修饰语？其replace、append、trim等方法返回的值是什么？
   1. String类前有final修饰符，value字段是private final，且没有提供修改value的方法，所以String是immutable的
   2. 每一次+=都是新建一个String对象，多次循环使用会增大开销
   3. Value前有final修饰
   4. Replace方法生成一个新的字符数组buf，将value中的oldchar替换成newchar，并返回由buf生成的新的String对象。如果没有要替换的字符，则返回原来的对象。
   5. Append方法：String类中没有找到这一方法，StringBuilder中有这一方法，直接对字符串进行操作，在一个StringBuilder对象后加上一个字符串，并返回StringBuilder对象
   6. Trim方法检测对象开头和结尾的空格，返回中间的子串（一个新的对象），子串的头尾没有空格
2. hashCode使用了什么公式，试比较一下它与身份证验证码、ISBN（国际标准书号）、银行卡号的验证码的生成算法有何不同？
   1. for (int i = 0; i < value.length; i++) { h = 31 \* h + val[i];} 将每个字符乘以31的(x-1)次方并相加，x是字符在字符串中从右向左数的位置
   2. 身份证验证码：将身份证号码前17位数分别乘以7 9 10 5 8 4 2 1 6 3 7 9 10 5 8 4 2，结果相加，除以11求余，余数为0 1 2 3 4 5 6 7 8 9 10分别对应1 0 X 9 8 7 6 5 4 3 2
   3. ISBN(13位)：用1乘ISBN的前12位中的奇数位，用3乘以偶数位，成绩之和以10为模，用10减去此模，即可得到校验位的值
   4. ISBN(10位)：前9位数分别乘以10, 9, 8, 7, 6, 5, 4, 3, 2，求和除以11取余，用11减去余数，如果是10则为X，是11则为0
   5. 银行卡号验证码：奇数位相加，偶数位乘以2，除以9取余相加，其和除以10取余，用10减去余数，是10则为0

Hashcode没有取余操作，且对象不相等时，其Hashcode相同的概率非常小

1. equals方法是判断内容相等还是引用相等？compareTo是比较什么？
   1. 判断内容相等
   2. 如果两个字符串相等则返回0。否则比较字符大小，如果相等则比较下一个字符。如果一直相等直到某个字符串比较完毕，则比较字符串长度

public int compareTo(String anotherString) {

int len1 = value.length;

int len2 = anotherString.value.length;

int lim = Math.min(len1, len2);

char v1[] = value;

char v2[] = anotherString.value;

int k = 0;

while (k < lim) {

char c1 = v1[k];

char c2 = v2[k];

if (c1 != c2) {

return c1 - c2;

}

k++;

}

return len1 - len2;

}

1. intern()方法的注释表明什么样的字符串一定是放到一个池子中的？

内容相等的字符串调用intern()方法相等

字符串池包含了所有literal strings和字符串常量表达式。

通过new操作符创建的字符串对象不指向字符串池中的任何对象，但是可以通过使用字符串的intern()方法来指向字符串池中与之内容相同的对象（如果池中已经有相同的字符串，则直接返回池中的字符串，否则先将字符串添加到池中，再返回）。

如果不用new创建字符串，先在池中寻找是否已经存在这个字符串，如果没有，则建立新的字符串对象，然后要创建的字符串变量指向这个地址，如果已经存在，则直接指向池中的这一对象地址。

1. 其indexOf使用了什么样的算法流程？

public int indexOf(int ch, int fromIndex) {

final int max = value.length;

if (fromIndex < 0) {

fromIndex = 0;

} else if (fromIndex >= max) {

// Note: fromIndex might be near -1>>>1.

return -1;

}

if (ch < Character.MIN\_SUPPLEMENTARY\_CODE\_POINT) {

// handle most cases here (ch is a BMP code point or a

// negative value (invalid code point))

final char[] value = this.value;

for (int i = fromIndex; i < max; i++) {

if (value[i] == ch) {

return i;

}

}

return -1;

} else {

return indexOfSupplementary(ch, fromIndex);

}

}

检测查找起始位置是否合法，如果小于零则将其设为零，如果超出字符串范围则返回-1

检测要查找的字符是否属于增补字符，如果不是，逐位比较字符串和待查找字符，相等则返回当前位置，如果没找到，返回-1；

如果是增补字符，调indexOfSupplementary方法，增补字符编码为两个代码单元，第一个单元来自于高代理范围（U+D800 至 U+DBFF），第二个单元来自于低代理范围（U+DC00 至 U+DFFF），对字符串逐位比较（需要比较两个代码单元的值），相等则返回当前位置，没找到则返回-1。

1. substring函数会抛出什么样的异常，这个异常是不是一定要捕获？

public String substring(int beginIndex) {

if (beginIndex < 0) {

throw new StringIndexOutOfBoundsException(beginIndex);

}

int subLen = value.length - beginIndex;

if (subLen < 0) {

throw new StringIndexOutOfBoundsException(subLen);

}

return (beginIndex == 0) ? this : new String(value, beginIndex, subLen);

}

StringIndexOutOfBoundsException，不一定要捕获

1. concat函数使用了哪个类的什么方法，这个方法中最终又是调用了什么类的什么方法？

public String concat(String str) {

int otherLen = str.length();

if (otherLen == 0) {

return this;

}

int len = value.length;

char buf[] = Arrays.copyOf(value, len + otherLen);

str.getChars(buf, len);

return new String(buf, true);

}

Arrays的copyOf方法，System的arraycopy方法

1. replaceAll中借用了什么类来实现？valueOf及format呢？

public String replaceAll(String regex, String replacement) {

return Pattern.compile(regex).matcher(this).replaceAll(replacement);

}

Pattern类

public static String valueOf(Object obj) {

return (obj == null) ? "null" : obj.toString();

}

Object类中的toString方法

public static String format(String format, Object... args) {

return new Formatter().format(format, args).toString();

}

Formatter类

-----java.lang.StringBuilder 及 AbstractStringBuilder类-----

1. StringBuilder的初始内存是多少字符？

public StringBuilder() {

super(16);

}

16

1. expandCapacity()方法是在原来的基础上扩展多少？

void expandCapacity(int minimumCapacity) {

int newCapacity = value.length \* 2 + 2;

if (newCapacity - minimumCapacity < 0)

newCapacity = minimumCapacity;

if (newCapacity < 0) {

if (minimumCapacity < 0) // overflow

throw new OutOfMemoryError();

newCapacity = Integer.MAX\_VALUE;

}

value = Arrays.copyOf(value, newCapacity);

}

扩展原来的长度+2

1. insert()、append()、delete()等方法的返回值是什么，这有什么好处？

返回this，即旧的StringBuffer对象。不同于String类每次操作都返回一个新的对象，StringBuffer对象是可变的，返回原来的对象节省开销。

-----java.lang.Integer类-----

1. value字段前面加了什么修饰词？为什么说Integer是immutable的？

private final int value; //在800+行，找了好久好久好久

Value字段前有final修饰

因为Integer类有final修饰，且value字段是private，没有对value进行修改的方法如何创建一个自己的不可变类：   
      .所有成员都是private   
      .不提供对成员的改变方法，例如：setXXXX   
      .确保所有的方法不会被重载。手段有两种：使用final Class(强不可变类)，或者将所有类方法加上final(弱不可变类)。   
      .如果某一个类成员不是原始变量(primitive)或者不可变类，必须通过在成员初始化(in)或者get方法(out)时通过深度clone方法，来确保类的不可变。

1. 其常数如MAX\_VALUE前面用了什么修饰词？

@Native public static final int MIN\_VALUE = 0x80000000;

@native, public, static, final

1. toString(int i)用了什么办法加快其计算的速度？

public static String toString(int i) {

if (i == Integer.MIN\_VALUE)

return "-2147483648";

int size = (i < 0) ? stringSize(-i) + 1 : stringSize(i);

char[] buf = new char[size];

getChars(i, size, buf);

return new String(buf, true);

}

利用getChars方法，再构造String

getChars方法：预先储存了char类型的'0'到‘9’，当i大于等于65536时，每次处理两位，当i较小时，采用快速算法。

1. parseInt函数会抛出什么样的异常，这个异常是不是一定要捕获？

NumberFormatException，不一定要捕获

1. IntegerCache是起什么作用的？valueOf(int i)表明什么范围内的int在boxing后会缓存？
2. IntegerCache会将IntegerCache.low到IntegerCache.high之间的整数对象预先存储起来，在这个范围内的内容相等的Integer对象是同一个，以节省内存，降低创建和销毁对象的开销。
3. 默认对-128~127范围内的Integer进行缓存，可以自行设定IntegerCache.high（缓存上界）的值
4. hashCode是怎样算的？

public int hashCode() {

return Integer.hashCode(value);

}

public static int hashCode(int value) {

return value;

}

直接返回Value（Integer对象的整数值）

-----java.lang.Math类-----

1. 举几个方法是native的方法。

如sin(), cons(), tan(), exp(), log(), sqrt(), pow()

（调用了StrictMath类中的同名方法，在StrictMath类中有native修饰符

-----java.util.Random类-----

1. 构造函数用什么进行了初始化？

public Random() {

this(seedUniquifier() ^ System.nanoTime());

}

用seedUniquifier() ^ System.nanoTime()获得种子

1. 随机数的公式是用了什么?

nextseed = (oldseed \* multiplier + addend) & mask; nextseed >>> (48 - bits)

-----java.math.BigInteger类-----

1. 判断一个大数是否可能为素数，有两个passXXX函数，分别是什么？

private boolean passesMillerRabin(int iterations, Random rnd)

private boolean passesLucasLehmer()

1. 内部是用什么来表示大整数的

final int[] mag;

Int数组

-----java.util.Arrays类-----

1. sort(int[])是使用什么排序方法？

public static void sort(int[] a) {

DualPivotQuicksort.sort(a, 0, a.length - 1, null, 0, 0);

}

使用DualPivot快速排序，改进经典的快排算法，提高效率

1. sort(Object[] a)是使用什么排序方法？

public static void sort(Object[] a) {

if (LegacyMergeSort.userRequested)

legacyMergeSort(a);

else

ComparableTimSort.sort(a, 0, a.length, null, 0, 0);

}

归并排序

-----java.util.ArrayList类-----

1. grow()的代码中可以看出每次容量增加多少倍？

private void grow(int minCapacity) {

// overflow-conscious code

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + (oldCapacity >> 1);

if (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

if (newCapacity - MAX\_ARRAY\_SIZE > 0)

newCapacity = hugeCapacity(minCapacity);

// minCapacity is usually close to size, so this is a win:

elementData = Arrays.copyOf(elementData, newCapacity);

}

若原容量的1.5倍小于minCapacity，则增加到minCapacity；否则增加0.5倍

-----java.util.Vector类-----

1. grow()的代码中可以看出每次容量增加多少倍？

private void grow(int minCapacity) {

// overflow-conscious code

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + ((capacityIncrement > 0) ?

capacityIncrement : oldCapacity);

if (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

if (newCapacity - MAX\_ARRAY\_SIZE > 0)

newCapacity = hugeCapacity(minCapacity);

elementData = Arrays.copyOf(elementData, newCapacity);

}

若原容量的2倍小于minCapacity，则增加到minCapacity；否则增加1倍

-----java.util.Stack类-----

1. Stack类继承了什么类？

class Stack<E> extends Vector<E>

Vector类

1. push()与pop()进行了什么操作？

public E push(E item) {

addElement(item);

return item;

}

Push()调用Vector的addElement()方法，实现入栈（将一个新元素加入栈，并返回这个元素）

public synchronized E pop() {

E obj;

int len = size();

obj = peek();

removeElementAt(len - 1);

return obj;

}

Pop()先取栈顶元素，再调用removeElementAt(len-1)实现出栈（将顶端元素从栈中删除，并返回这个元素）

-----java.util.Hashtable类-----

1. 其内部类class Entry含有哪4个字段？

private static class Entry<K,V> implements Map.Entry<K,V> {

final int hash;

final K key;

V value;

Entry<K,V> next;

含有hash, key, value, next四个字段

-----java.util.TreeMap类-----

1. containsKey()的代码中可以看出要找到对象的条件是什么？如果一个Key对象的hashCode是变化的（如随着内容而算出来的），这个Key对象还会找到吗？ public boolean containsKey(Object key) {

return getEntry(key) != null;

}

final Entry<K,V> getEntry(Object key) {

// Offload comparator-based version for sake of performance

if (comparator != null)

return getEntryUsingComparator(key);

if (key == null)

throw new NullPointerException();

@SuppressWarnings("unchecked")

Comparable<? super K> k = (Comparable<? super K>) key;

Entry<K,V> p = root;

while (p != null) {

int cmp = k.compareTo(p.key);

if (cmp < 0)

p = p.left;

else if (cmp > 0)

p = p.right;

else

return p;

}

return null;

}

找到对象的条件：getEntry(key)不是null

在getEntry方法中，如果comparator为null，使用compareTo对key和当前位置进行比较，根据比较情况决定向左或向右选取下一个比较值，找不到相等的则返回null；如果comparator不为null，调用getEntryUsintComparator方法，与上面类似，只不一定用comparator进行比较。

TreeMap是红黑树算法的实现，其内部是有序的，如果hashCode是变化的，就不能进行正确的比较，因此就无法找到key。

-----java.util.TreeSet类-----

1. TreeSet的底层是用什么对象来实现的？

private transient NavigableMap<E,Object> m;

用NavigableMap对象实现

-----java.util.Timer类-----

1. Timer的底层是如何实现重复执行的？

private void mainLoop() {

while (true) {

try {

TimerTask task;

boolean taskFired;

synchronized(queue) {

// Wait for queue to become non-empty

while (queue.isEmpty() && newTasksMayBeScheduled)

queue.wait();

if (queue.isEmpty())

break; // Queue is empty and will forever remain; die

// Queue nonempty; look at first evt and do the right thing

long currentTime, executionTime;

task = queue.getMin();

synchronized(task.lock) {

if (task.state == TimerTask.CANCELLED) {

queue.removeMin();

continue; // No action required, poll queue again

}

currentTime = System.currentTimeMillis();

executionTime = task.nextExecutionTime;

if (taskFired = (executionTime<=currentTime)) {

if (task.period == 0) { // Non-repeating, remove

queue.removeMin();

task.state = TimerTask.EXECUTED;

} else { // Repeating task, reschedule

queue.rescheduleMin(

task.period<0 ? currentTime - task.period

: executionTime + task.period);

}

}

}

if (!taskFired) // Task hasn't yet fired; wait

queue.wait(executionTime - currentTime);

}

if (taskFired) // Task fired; run it, holding no locks

task.run();

} catch(InterruptedException e) {

}

}

}

}

1. 先获得队列锁，然后去TaskQueue中取TimerTask，然后去判断此队列为空且新任务可安排标记是打开的。如果不满足，线程等待，将队列锁释放。
2. 如果队列为空，那么跳出死循环。
3. 取得队列中的下一个元素，并获得任务锁。
4. 检查任务状态，如果任务状态为取消，那么直接取消，并跳过此轮循环。
5. 得到任务的计划执行时间，并检查与当前时间的先后，如果当前时间已经到或者超过计划执行时间，那么置状态位为执行。
6. 释放任务锁。
7. 如果没有，线程等待执行时间和当前时间差。
8. 释放队列锁
9. 看任务是否可以执行标记，来确定是否执行任务
10. 反复从1开始

-----javax.swing.Timer类-----

1. Timer的底层是在哪里调用SwingUtilities.invokeLater的？

void post() {

if (notify.compareAndSet(false, true) || !coalesce) {

AccessController.doPrivileged(new PrivilegedAction<Void>() {

public Void run() {

SwingUtilities.invokeLater(doPostEvent);

return null;

}

}, getAccessControlContext());

}

}

在post()方法中调用