最近查找了一个BUG是关于equals问题，因为equals被重写了但是没有被关注，就是没想到会在这个问题上栽坑，所以就看了一下equals和hashCode的内容，总结一下避免以后不出现相同的问题！

> equals和hashCode方法java层面最初结构出现在Object类中

Object

```
/**
 * Returns a hash code value for the object. This method is
 * supported for the benefit of hash tables such as those provided by
 * {@link java.util.HashMap}.
 * <p>
 * The general contract of {@code hashCode} is:
 * <ul>
 * <li>Whenever it is invoked on the same object more than once during
 * an execution of a Java application, the {@code hashCode} method
 * must consistently return the same integer, provided no information
 * used in {@code equals} comparisons on the object is modified.
 * This integer need not remain consistent from one execution of an
 * application to another execution of the same application.
 * <li>If two objects are equal according to the {@code equals(Object)}
 * method, then calling the {@code hashCode} method on each of
 * the two objects must produce the same integer result.
 * <li>It is <em>not</em> required that if two objects are unequal
 * according to the {@link java.lang.Object#equals(java.lang.Object)}
 * method, then calling the {@code hashCode} method on each of the
 * two objects must produce distinct integer results. However, the
 * programmer should be aware that producing distinct integer results
 * for unequal objects may improve the performance of hash tables.
 * </ul>
 * <p>
 * As much as is reasonably practical, the hashCode method defined by
 * class {@code Object} does return distinct integers for distinct
 * objects. (This is typically implemented by converting the internal
 * address of the object into an integer, but this implementation
 * technique is not required by the
 * Java&trade; programming language.)
 *
 * @return a hash code value for this object.
 * @see java.lang.Object#equals(java.lang.Object)
 * @see java.lang.System#identityHashCode
```

```java
 */
public native int hashCode();

/**
 * Indicates whether some other object is "equal to" this one.
 * <p>
 * The {@code equals} method implements an equivalence relation
 * on non-null object references:
 * <ul>
 * <li>It is <i>reflexive</i>: for any non-null reference value
 * {@code x}, {@code x.equals(x)} should return
 * {@code true}.
 * <li>It is <i>symmetric</i>: for any non-null reference values
 * {@code x} and {@code y}, {@code x.equals(y)}
 * should return {@code true} if and only if
 * {@code y.equals(x)} returns {@code true}.
 * <li>It is <i>transitive</i>: for any non-null reference values
 * {@code x}, {@code y}, and {@code z}, if
 * {@code x.equals(y)} returns {@code true} and
 * {@code y.equals(z)} returns {@code true}, then
 * {@code x.equals(z)} should return {@code true}.
 * <li>It is <i>consistent</i>: for any non-null reference values
 * {@code x} and {@code y}, multiple invocations of
 * {@code x.equals(y)} consistently return {@code true}
 * or consistently return {@code false}, provided no
 * information used in {@code equals} comparisons on the
 * objects is modified.
 * <li>For any non-null reference value {@code x},
 * {@code x.equals(null)} should return {@code false}.
 * </ul>
 * <p>
 * The {@code equals} method for class {@code Object} implements
 * the most discriminating possible equivalence relation on objects;
 * that is, for any non-null reference values {@code x} and
 * {@code y}, this method returns {@code true} if and only
 * if {@code x} and {@code y} refer to the same object
 * ({@code x == y} has the value {@code true}).
 * <p>
 * Note that it is generally necessary to override the {@code hashCode}
 * method whenever this method is overridden, so as to maintain the
 * general contract for the {@code hashCode} method, which states
 * that equal objects must have equal hash codes.
 *
 * @param obj the reference object with which to compare.
 * @return {@code true} if this object is the same as the obj
 * argument; {@code false} otherwise.
 * @see #hashCode()
 * @see java.util.HashMap
 */
public boolean equals(Object obj) {
 return (this == obj);
}
```

上述注释可能有点长，但是主要内容没什么，先说一下equals:

equals

看上文Object#equals(Object obj)，如果方法内容被重写equals和==那就是一样的性质了。当然研究的内容肯定不是这么简单！

接下还是一段代码的展示equal和“==”的区别，选取原始类型boolean，int，原始类型包装类Boolean类,Integer，以及String类进行说明。

int和Integer

```



/**********************************************************************
***
 * @Title: EqualsDome
 * @Package com.base
 * @Description: 实验比较equals
 * @author shimingda
 * @date 2020/1/9
  * @version V1.0
  *********************************************************************
*****/
public class EqualsDome
{
 public static void main(String[] args)
 {
 int a1=130;
 int a2=130;
 Integer b1=130;
Integer b2=130;
Integer c1=new Integer(130);
Integer c2=new Integer(130);


System.out.println("a1==a2:"+(a1==a2));
System.out.println("a1==b1:"+(a1==b1));
System.out.println("b1==b2:"+(b1==b2));
System.out.println("bi==c1:"+(b1==c1));
System.out.println("c1==c2:"+(c1==c2));

System.out.println("b1.equals(b2):"+(b1.equals(b2)));
System.out.println("bi.equals(c1):"+(b1.equals(c1)));
System.out.println("c1.equals(c2):"+(c1.equals(c2)));

System.out.println("System.identityHashCode(a1) is : "+ System.identityHashCode(a1));
System.out.println("System.identityHashCode(a2) is : "+ System.identityHashCode(a2));
System.out.println("b1.hashCode() is : "+b1.hashCode()+" System.identityHashCode(b1) is : "+ System.identityHashCode(b1));
System.out.println("b2.hashCode() is : "+b2.hashCode()+" System.identityHashCode(b2) is : "+ System.identityHashCode(b2));
System.out.println("c1.hashCode() is : "+c1.hashCode()+" System.identityHashCode(c1) is : "+ System.identityHashCode(c1));
System.out.println("c2.hashCode() is : "+c2.hashCode()+" System.identityHashCode(c2) is : "+ System.identityHashCode(c2));
 }
}
--------------------------------------------------------------
结果:
a1==a2:true
a1==b1:true
b1==b2:false
bi==c1:false
c1==c2:false
b1.equals(b2):true
bi.equals(c1):true
c1.equals(c2):true
System.identityHashCode(a1) is : 1163157884
System.identityHashCode(a2) is : 1956725890
b1.hashCode() is : 130 System.identityHashCode(b1) is : 356573597
```

```
b2.hashCode() is : 130 System.identityHashCode(b2) is : 1735600054
c1.hashCode() is : 130 System.identityHashCode(c1) is : 21685669
c2.hashCode() is : 130 System.identityHashCode(c2) is : 2133927002
```

equals和hashCode方法

```

@Override
public int hashCode() {
 return Integer.hashCode(value);
}

public static int hashCode(int value) {
 return value;
}


public boolean equals(Object obj) {
 if (obj instanceof Integer) {
 return value == ((Integer)obj).intValue();
 }
 return false;
}
```

boolean和Boolean

```
package com.base;

import sun.applet.Main;

/*********************************************************************
***
 * @Title: EqualsDome
 * @Package com.base
 * @Description: 实验比较equals
 * @author shimingda
  * @date 2020/1/9
  * @version V1.0
  *******************************************************************
*****/
public class EqualsDome
{
 public static void main(String[] args)
{
boolean a1=true;
boolean a2=true;

Boolean b1=true;
Boolean b2=true;
Boolean c1=new Boolean(true);
Boolean c2=new Boolean(true);

System.out.println("a1==a2:"+(a1==a2));
System.out.println("a1==b1:"+(a1==b1));
System.out.println("b1==b2:"+(b1==b2));
System.out.println("bi==c1:"+(b1==c1));
System.out.println("c1==c2:"+(c1==c2));

System.out.println("b1.equals(b2):"+(b1.equals(b2)));
System.out.println("bi.equals(c1):"+(b1.equals(c1)));
System.out.println("c1.equals(c2):"+(c1.equals(c2)));

System.out.println("System.identityHashCode(a1) is : "+ System.identityHashCode(a1));
System.out.println("System.identityHashCode(a2) is : "+ System.identityHashCode(a2));
System.out.println("b1.hashCode() is : "+b1.hashCode()+" System.identityHashCode(b1) is : "+ System.identityHashCode(b1));
System.out.println("b2.hashCode() is : "+b2.hashCode()+" System.identityHashCode(b2) is : "+ System.identityHashCode(b2));
System.out.println("c1.hashCode() is : "+c1.hashCode()+" System.identityHashCode(c1) is : "+ System.identityHashCode(c1));
System.out.println("c2.hashCode() is : "+c2.hashCode()+" System.identityHashCode(c2) is : "+ System.identityHashCode(c2));
 }
}
----------------------------------------------
a1==a2:true
a1==b1:true
b1==b2:true
bi==c1:false
c1==c2:false
b1.equals(b2):true
bi.equals(c1):true
c1.equals(c2):true
```

```
System.identityHashCode(a1) is : 1163157884
System.identityHashCode(a2) is : 1163157884
b1.hashCode() is : 1231 System.identityHashCode(b1) is : 1163157884
b2.hashCode() is : 1231 System.identityHashCode(b2) is : 1163157884
c1.hashCode() is : 1231 System.identityHashCode(c1) is : 1956725890
c2.hashCode() is : 1231 System.identityHashCode(c2) is : 356573597

```

equals和hashCode方法

```

@Override
public int hashCode() {
return Boolean.hashCode(value);
}

public static int hashCode(boolean value) {
return value ? 1231 : 1237;
}


 public boolean equals(Object obj) {
 if (obj instanceof Boolean) {
 return value == ((Boolean)obj).booleanValue();
 }
 return false;
 }
```

String

```

/*****************************************************************************
 * @Title: EqualsDome
 * @Package com.base
 * @Description: 实验比较equals
 * @author shimingda
 * @date 2020/1/9
 * @version V1.0
****************************************************************************/
public class EqualsDome
{
 public static void main(String[] args)
 {
 String a1="a";
 String a2="a";
 String b1=new String("a");
 String b2=new String("a");


 System.out.println("a1==a2:"+(a1==a2));
 System.out.println("a1==b1:"+(a1==b1));
 System.out.println("b1==b2:"+(b1==b2));

 System.out.println("a1.equals(a2):"+(b1.equals(b2)));
 System.out.println("ai.equals(b1):"+(b1.equals(b1)));
 System.out.println("b1.equals(b2):"+(b1.equals(b2)));

 System.out.println("a1.hashCode() is : "+a1.hashCode()+" System.identityHashCode(a1) is : "+ System.identityHashCode(a1));
 System.out.println("a2.hashCode() is : "+a2.hashCode()+" System.identityHashCode(a2) is : "+ System.identityHashCode(a2));
 System.out.println("b1.hashCode() is : "+b1.hashCode()+" System.identityHashCode(b1) is : "+ System.identityHashCode(b1));
 System.out.println("b2.hashCode() is : "+b2.hashCode()+" System.identityHashCode(b2) is : "+ System.identityHashCode(b2));

 }
}
-----------------------------------------------------------------
a1==a2:true
a1==b1:false
b1==b2:false
a1.equals(a2):true
ai.equals(b1):true
b1.equals(b2):true
a1.hashCode() is : 97 System.identityHashCode(a1) is : 1163157884
a2.hashCode() is : 97 System.identityHashCode(a2) is : 1163157884
b1.hashCode() is : 97 System.identityHashCode(b1) is : 1956725890
b2.hashCode() is : 97 System.identityHashCode(b2) is : 356573597
```

equals和hashCode方法

```
/**
 * Returns a hash code for this string. The hash code for a
 * {@code String} object is computed as
```

```java
* <blockquote><pre>
* s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
* </pre></blockquote>
* using {@code int} arithmetic, where {@code s[i]} is the
* <i>i</i>th character of the string, {@code n} is the length of
* the string, and {@code ^} indicates exponentiation.
 * (The hash value of the empty string is zero.)
 *
 * @return a hash code value for this object.
 */
public int hashCode() {
 int h = hash;
 if (h == 0 && value.length > 0) {
 char val[] = value;

 for (int i = 0; i < value.length; i++) {
 h = 31 * h + val[i];
 }
 hash = h;
 }
 return h;
}
/**
 * Compares this string to the specified object. The result is {@code
 * true} if and only if the argument is not {@code null} and is a {@code
 * String} object that represents the same sequence of characters as this
 * object.
 *
 * @param anObject
 * The object to compare this {@code String} against
 *
 * @return {@code true} if the given object represents a {@code String}
 * equivalent to this string, {@code false} otherwise
 *
 * @see #compareTo(String)
 * @see #equalsIgnoreCase(String)
 */
public boolean equals(Object anObject) {
 if (this == anObject) {
```

```
return true;
}
if (anObject instanceof String) {
String anotherString = (String)anObject;
int n = value.length;
if (n == anotherString.value.length) {
char v1[] = value;
char v2[] = anotherString.value;
int i = 0;
while (n-- != 0) {
if (v1[i] != v2[i])
return false;
i++;
}
return true;
}
}
return false;
}
```

identityHashCode和hashCode关系请参考这个总结：HashCode与identityHashCode底层究竟发生了什么；

int大于127会有自动装箱问题：如有不理解请参考总结：自动装箱

根据以上实验可以看出

1.==是比较运算符，当是基本数据类型时，比较的是变量的值，当是其他类型的对象时，用它比较的是两个对象的引用地址值是否相等

2.equals是一个方法，如果对应的类没有重现Object类的equals()方法，则和==是一样的作用，如果重写要按照重写的方式进行比较。

以上equals和hashCode方法会同时被重写这是为什么?

> 如果不覆盖hashCode就会违反Object，hashCode的通用规定，从而导致该类无法结合所有散列的集合正常工作，例如，HashMap，HashSet等等集合。

通用规范约定（摘自Object规范[javaSE6]）

- 在应用程序的执行期间，只要对象的 equals方法的比较操作所用到的信息没有被修改那么对这同一个对象调用多次， hash Code方法都必须始终如一地返回同一个整数。在同个应用程序的多次执行过程中，每次执行所返回的整数可以不一致。

- ·如果两个对象根据 equals（object）方法比较是相等的，那么调用这两个对象中任意一个对象的 hash Code方法都必须产生同样的整数结果。
- ·如果两个对象根据 equals（object）方法比较是不相等的，那么调用这两个对象中任意一个等的对象产生截然不同的整数结果，有可能提高散列表（ hash table）的性能，给不相对象的 hash Code方法，则不一定要产生不同的整数结果。但是程序员应该知道，给不相等的对象产生截然不同的整数结果，有可能提高散列表的性能。

hashCode()

> 获取哈希码，也称为散列码，返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。

Object中hashCode（）方法是native 方法。本地方法内容请参考：本地方法

重写hashCode（）需要有好的设计，好的散列码通常倾向于“不相等的对象产生不同散列码”，以下方法是较好的设计形式，仅供参考。

```
@Override
 public int hashCode() {
 int result = 17;
 result = 31 * result + (param1== null ? 0 : param1.hashCode());
 result = 31 * result + (param2== null ? 0 : param2.hashCode());
 return result;
 }
```

为什么选择31

原因一 不容易产生结果冲突

参考String#hashcod（）的重写方式： * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]，31属于一个大小适中的质数，不容易产生计算冲突。

```
/**
 * Returns a hash code for this string. The hash code for a
 * {@code String} object is computed as
 * <blockquote><pre>
 * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
 * </pre></blockquote>
 * using {@code int} arithmetic, where {@code s[i]} is the
 * <i>i</i>th character of the string, {@code n} is the length of
 * the string, and {@code ^} indicates exponentiation.
  * (The hash value of the empty string is zero.)
  *
  * @return a hash code value for this object.
  */
```

```java
public int hashCode() {
 int h = hash;
 if (h == 0 && value.length > 0) {
 char val[] = value;

 for (int i = 0; i < value.length; i++) {
 h = 31 * h + val[i];
 }
 hash = h;
 }
 return h;
}
```

原因二 可被虚拟机优化

JVM里最有效的计算方式就是进行位运算了：

* 左移 << ：左边的最高位丢弃，右边补全0（把 << 左边的数据*2的移动次幂）。

* 右移 >> ：把>>左边的数据/2的移动次幂。

* 无符号右移 >>> ：无论最高位是0还是1，左边补齐0。