



MaximPedich 4 февраля 2013 в 13:14

Разбираемся с hashCode() и equals()

Java

Из песочницы

Я недавно начал заниматься программированием, и в этой области для меня много новс. Данная статья рассчитана на начинающих java-программистов, и, надеюсь, поможет в освоении излюбленной темы для собеседований "hashCode и equals".

Хочу сразу предостеречь, что я не являюсь экспертом в данной теме и могу что-то не так Реклама понимать, поэтому, если вы нашли ошибку или неточность — свяжитесь со мной.

Что такое хеш-код?

Если очень просто, то хеш-код — это число. На самом деле просто, не так ли? Если более точно, то это битовая строка фиксированной длины, полученная из массива произвольной длины (википедия).

Пример №1

Выполним следующий код:

```
public class Main {
    public static void main(String[] args) {
        Object object = new Object();
        int hCode;
        hCode = object.hashCode();
        System.out.println(hCode);
    }
}
```

В результате выполнения программы в консоль выведется целое 10-ти значное число. Это число и есть наша битовая строка фиксированной длины. В java она представлена в виде числа примитивного типа int, который равен 4-м байтам, и может помещать числа от -2 147 483 648 до 2 147 483 647. На данном этапе важно понимать, что хеш-код это число, у которого есть свой предел, который для java ограничен примитивным целочисленным типом int.

Вторая часть объяснения гласит:

полученная из массива произвольной длины.

Под массивом произвольной длины мы будем понимать объект. В 1 примере в качестве массива произвольной длины у нас выступает объект типа Object.

В итоге, в терминах Java, хеш-код — это целочисленный результат работы метода, которому в качестве входного параметра передан объект.

Этот метод реализован таким образом, что для одного и того-же входного объекта, хеш-код всегда будет одинаковым. Следует понимать, что множество возможных хеш-кодов ограничено примитивным типом int, а множество объектов ограничено только нашей фантазией. Отсюда следует утверждение: "Множество объектов мощнее множества хеш-кодов". Из-за этого ограничения, вполне возможна ситуация, что хеш-коды разных объектов могут совпасть.

Здесь главное понять, что:

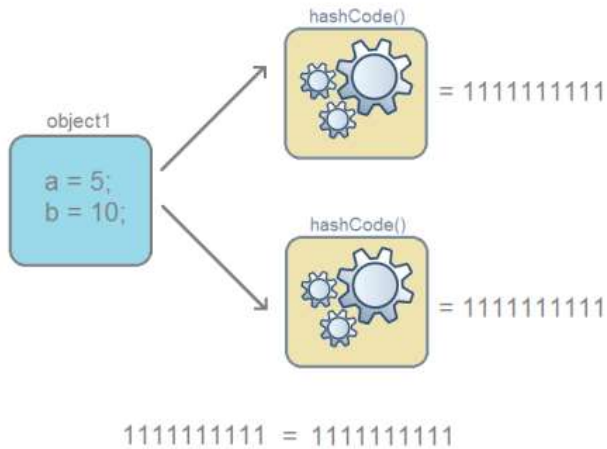
- Если хеш-коды разные, то и входные объекты гарантированно разные.
- Если хеш-коды равны, то входные объекты не всегда равны.

Ситуация, когда у *разных* объектов *одинаковые* хеш-коды называется — коллизией. Вероятность возникновения коллизии зависит от используемого алгоритма генерации хеш-кода.

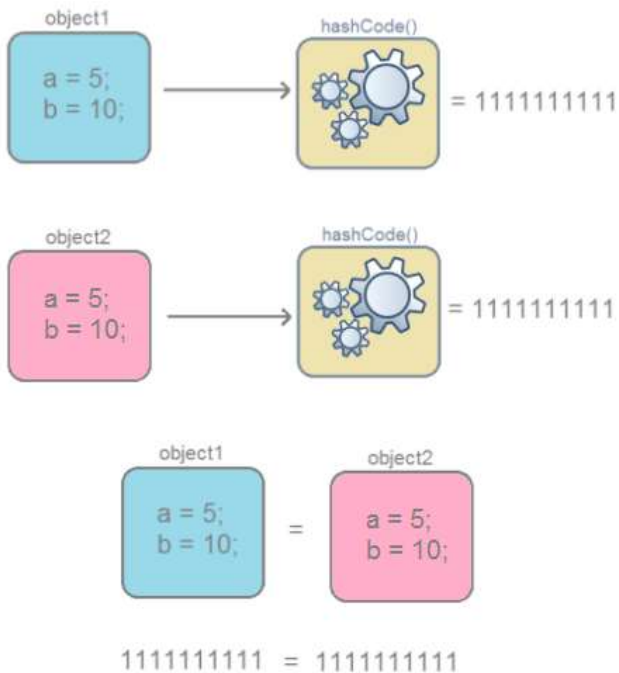
Подведём итог:

Сперва, что-бы избежать путаницы, определимся с терминологией. Одинаковые объекты — это объекты одного класса с одинаковым содержимым полей.

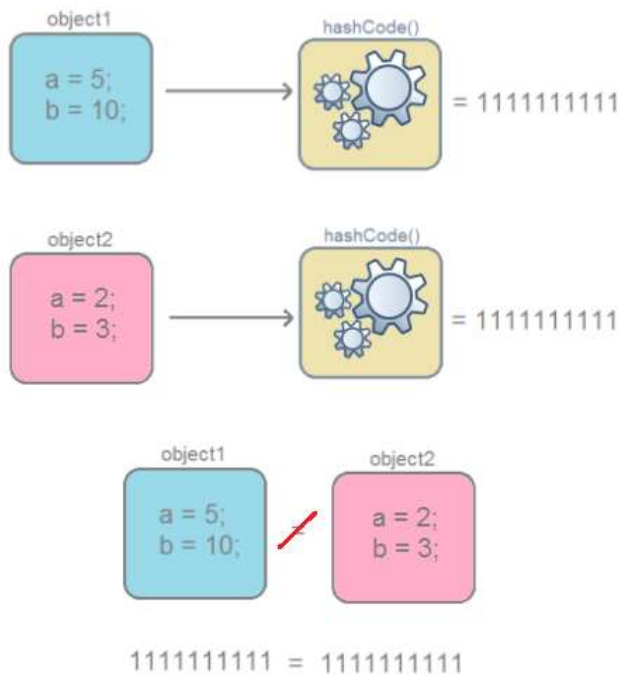
1. для одного и того-же объекта, хеш-код всегда будет одинаковым;



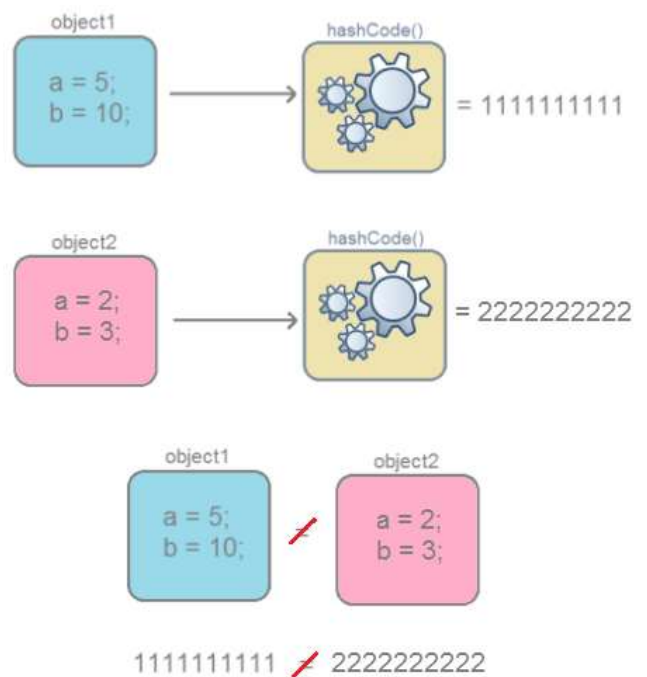
2. если объекты одинаковые, то и хеш-коды одинаковые (но не наоборот, см. правило 3).



3. если хеш-коды равны, то входные объекты не всегда равны (коллизия);



4. если хеш-коды разные, то и объекты гарантированно разные;



Понятие эквивалентности. Метод equals()

Начнем с того, что в java, каждый вызов оператора new порождает новый объект в памяти. Для иллюстрации создадим какой-нибудь класс, пускай он будет называться "BlackBox".

Пример №2

Выполним следующий код:

```
public class BlackBox {
    int varA;
    int varB;

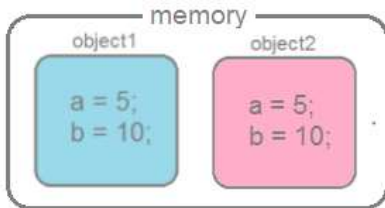
    BlackBox(int varA, int varB){
        this.varA = varA;
        this.varB = varB;
    }
}
```

```
}  
}
```

Создадим класс для демонстрации BlackBox.

```
public class DemoBlackBox {  
    public static void main(String[] args) {  
        BlackBox object1 = new BlackBox(5, 10);  
        BlackBox object2 = new BlackBox(5, 10);  
    }  
}
```

Во втором примере, в памяти создается два объекта.



Но, как вы уже обратили внимание, содержимое этих объектов одинаково, то есть эквивалентно. Для проверки эквивалентности в классе `Object` существует метод `equals()`, который сравнивает содержимое объектов и выводит значение типа `boolean true`, если содержимое эквивалентно, и `false` — если нет.

```
object1.equals(object2); // должно быть true, поскольку содержимое объектов эквивалентно
```

Эквивалентность и хеш-код тесно связаны между собой, поскольку хеш-код вычисляется на основании содержимого объекта (значения полей) и если у двух объектов одного и того же класса содержимое одинаковое, то и хеш-коды должны быть одинаковые (см. правило 2).

Иными словами:

```
object1.equals(object2) // должно быть true  
object1.hashCode() == object2.hashCode() // должно быть true
```

Я написал “должно быть”, потому что если вы выполните предыдущий пример, то на самом деле результатом выполнения всех операций будет `false`. Для пояснения причин, заглянем в исходные коды класса `Object`.

Класс Object

Как известно, все `java`-классы наследуются от класса `Object`. В этом классе уже определены методы `hashCode()` и `equals()`. Определяя свой класс, вы автоматически наследуете все методы класса `Object`. И в ситуации, когда в вашем классе не переопределены (`@overriding`) `hashCode()` и `equals()`, то используется их реализация из `Object`.

Рассмотрим исходный код метода `equals()` в классе `Object`.

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

При сравнение объектов, операция “==” вернет `true` лишь в одном случае — когда ссылки указывают на один и тот-же объект. В данном случае не учитывается содержимое полей.

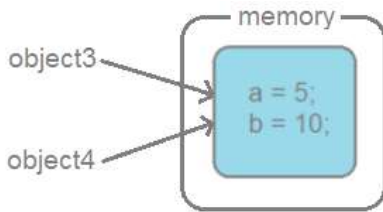
Выполнив приведённый ниже код, `equals` вернет `true`.

```
public class DemoBlackBox {  
    public static void main(String[] args) {  
        BlackBox object3 = new BlackBox(5, 10);  
    }  
}
```

```

        BlackBox object4 = object3; // Переменная object4 ссылается на
                                   // тот-же объект что и переменная object3
    object3.equals(object4) // true
    }
}

```



Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп



Войти

Регистрация

далее на 8 перед названием(), который тоже работает не так как предполагалось.

Заглянем в исходный код метода hashCode() в классе Object:

```
public native int hashCode();
```

Вот собственно и вся реализация. Ключевое слово native означает, что реализация данного метода выполнена на другом языке, например на C, C++ или ассемблере. Конкретный native int hashCode() реализован на C++, вот исходники — <http://hg.openjdk.java.net/jdk7/jdk7/hotspot/file/tip/src/share/vm/runtime/synchronizer.cpp> функция get_next_hash.

При вычислении хэш-кода для объектов класса Object по умолчанию используется Park-Miller RNG алгоритм. В основу работы данного алгоритма положен генератор случайных чисел. Это означает, что при каждом запуске программы у объекта будет разный хэш-код.

Получается, что используя реализацию метода hashCode() от класса Object, мы при каждом создании объекта класса new BlackBox(), будем получать разные хеш-коды. Мало того, перезапуская программу, мы будем получать абсолютно разные значения, поскольку это просто случайное число.

Но, как мы помним, должно выполняться правило: “если у двух объектов одного и того же класса содержимое одинаковое, то и хеш-коды должны быть одинаковые”. Поэтому, при создании пользовательского класса, принято переопределять методы hashCode() и equals() таким образом, что бы учитывались поля объекта.

Это можно сделать вручную либо воспользовавшись средствами генерации исходного кода в IDE. Например, в Eclipse это Source → Generate hashCode() and equals()...

В итоге, класс BlackBox приобретает вид:

```

public class BlackBox {
    int varA;
    int varB;

    BlackBox(int varA, int varB){
        this.varA = varA;
        this.varB = varB;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + varA;
        result = prime * result + varB;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
    }
}

```

```
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        BlackBox other = (BlackBox) obj;
        if (varA != other.varA)
            return false;
        if (varB != other.varB)
            return false;
        return true;
    }
}
```

Теперь методы hashCode() и equals() работают корректно и учитывают содержимое полей объекта:

```
object1.equals(object2); //true
object1.hashCode() == object2.hashCode(); //true
```

Кому интересно переопределение в ручную, можно почитать Effective Java — Joshua Bloch, chapter 3, item 8,9.

Итоги

Создавая пользовательский класс, нужно переопределять методы hashCode() и equals(), что бы они корректно работали и учитывали данные объекта. Кроме того, если оставить реализацию из Object, то при использовании java.util.HashMap возникнут проблемы, поскольку HashMap активно используют hashCode() и equals() в своей работе, но про это хорошо написано у @tarzan82 в посте Структуры данных в картинках.HashMap.

Ссылки:

- Исходный код класса java.lang.Object
- Откуда растут ноги у hashCode
- Как сгенерировать hashCode в Java
- Эффективное и безошибочное объявление методов hashCode() и equals()
- Сравнение объектов: теория
- Сравнение объектов: практика

Теги: java, hashcode, equals, собеседование

Хабы: Java

↑ +24 ↓ 677 536k 14 Поделиться

**17,0**

Карма

0,0

Рейтинг

Максим Педич @MaximPedich

Пользователь

ПОХОЖИЕ ПУБЛИКАЦИИ

5 августа 2020 в 16:31

Под капотом PVS-Studio для Java: разработка диагностик

↑ +16 👁 1,6k 📖 14 💬 11

16 января 2019 в 10:52

Задача от иностранной компании или как я провалил собеседование

↑ +9 👁 39,4k 📖 149 💬 63

17 февраля 2013 в 23:33

Заметки о реализации hashCode() в Java

↑ +5 👁 40k 📖 97 💬 26

КУРСЫ



Профессия Java-разработчик

8 сентября 2020 • 18 месяцев • 212 994 ₽ • SkillFactory



Разработчик Java

27 сентября 2020 • 5 месяцев • 60 000 ₽ • OTUS



Курс Java Junior Developer

9 сентября 2020 • 6 недель • 19 970 ₽ • Level UP



Java Professional

12 сентября 2020 • 2 месяца • 95 \$ • CyberBionic Systematics



Java Базовый

16 сентября 2020 • 6 недель • 95 \$ • CyberBionic Systematics

Больше курсов на Хабр Карьере



Яндексе.Директ

Обучение языку Python с нуля
Курсы от Яндекса!

praktikum.yandex.ru

Обучаем интернет-специалистов с нуля в Яндекс.Практикуме.
Вводный курс – бесплатно.

Реклама

Комментарии 14



barker 4 февраля 2013 в 13:58

↑ 0 ↓

Создавая пользовательский класс, нужно переопределять методы hashCode() и equals(), что бы они корректно работали и учитывали данные объекта. Кроме того, если оставить реализацию из Object, то при использовании java.util.HashMap возникнут проблемы




Никаких проблем не возникнет, конечно же. Всё будет корректно работать. А переопределять нужно, да, только при необходимости и правильно.



barker 4 февраля 2013 в 14:05


↑ +7 ↓

Вернее, возникнут, только если пытаться использовать в качестве ключа не исходный объект, а новый, который мы принимаем за тождественный исходному. Но в таком случае, очевидно, придётся уже по-любому equals переопределять, т.к. в других местах программы мы всё равно их сравнивать потенциально захотим, раз мы ввели отношение равенства. А значит и hashCode надо переопределить по-правильному.

 **Fashion** 4 февраля 2013 в 14:48  

 -8 

Проще использовать `HashCodeBuilder` и `EqualsBuilder` из библиотеки `commons-lang` чем городить что-то своё.

 **bay73** 4 февраля 2013 в 14:52  




 +9 

Создавая пользовательский класс, нужно переопределять методы `hashCode()` и `equals()`, что бы они корректно работали и учитывали данные объекта.

Это слишком категорично. Правильнее будет так: если необходимо, чтобы разные объекты с одинаковым содержимым рассматривались как равные, то надо переопределить `equals`. Практически это нужно далеко не всегда.

А вот категорическим правилом должно быть следующее:

«Переопределил `equals` — переопредели и `hashCode`»




 **Арх** 4 февраля 2013 в 15:16  

 0 

Почитав статью делается вывод, что просто всегда надо в Eclipse делать: `Source -> Generate hashCode() and equals()`;



«Пейте дети молоко — Будете здоровы»

Кстати можно было бы освятить тему `equals` и `hashCode` для объектов используемых в `orm`'ах, потому что там например в 90% случаев достаточно работать только с `id` объектов.

 **elw00d** 4 февраля 2013 в 15:42  

 +5 

Не следует реализовывать `equality members` для абсолютно всех пользовательских классов, поскольку наличие реализации `equals` и `hashCode` намекает программисту, читающему код, на то, что объекты класса используются в качестве ключа в хешмапах.

 **Throwable** 5 февраля 2013 в 14:22  

 +1 

Для генерации хешкодов и проверки на равенство удобно использовать методы класса `Objects` из библиотеки `Guava`:





code.google.com/p/guava-libraries/wiki/CommonObjectUtilitiesExplained

В `JDK7` уже вставлены соответствующие методы в класс `Objects`: docs.oracle.com/javase/7/docs/api/java/util/Objects.html

 **serf** 5 февраля 2013 в 21:55    

 0 

Добавлю, в `apache commons` также есть для этого утилиты.

 **Asen**  5 февраля 2013 в 22:26  

 0 

Часто бывает полезно разбираться в такого рода мелочах для более детально понимания работы целой системы(или подсистемы). +1)

Да, и кстати, коллизии — обычное дело для всех хэш-функций. Коллизии неизбежны, и нет ни одного алгоритма, который бы гарантировал полное отсутствие коллизий.

 **barker**  6 февраля 2013 в 17:46    

 0 

Очевидно, хеш по определению не может не иметь потенциальных коллизий, т.к. мощность множества хешей заведомо меньше мощности множества вариантов того что хешируется (иначе в хеше как таковом и смысла нет). А в данном случае хеш и не должен особо защищаться от коллизий, это для такой малой битности изначально невозможно. Для большинства реальных объектов коллизии очень вероятны. Например, для строк (объекты `java.lang.String`), даже коротких, вероятность коллизий их `hashCode` не просто есть, она огромна.

 **Asen** 6 февраля 2013 в 19:04    

 0 

Из этого следует, что не нужно применять алгоритм `hashCode()` повсеместно. В некоторых случаях это не только не принесет пользы, но и наделает вреда, с которым программисту придется потом разбираться. Прежде чем писать код, нужно все обдумать, как следует...

**barker**

6 февраля 2013 в 20:00



+1



Ну я так выше и написал, собственно. Ну в строках и всё таком прочем, что потенциально может являться ключом, переопределять всё же нужно. А нужно ли и как именно — тут от семантики объекта зависит, конечно. В чём-то похожем на строку, понятное дело, должен быть на содержимом построен хеш.

**Informatik**

4 октября 2015 в 23:35



0



А если переменные varA и varB будут типа double, тогда какую функцию hashCode() можно написать?

**withoutuniverse**

8 декабря 2015 в 14:44



0



Исходный код для класса java.lang.Double
результат получить и записать как «a & b», например

```
@Override
public int hashCode() {
    long v = doubleToLongBits(value);
    return (int) (v ^ (v >>> 32));
}
```

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

- Сутки
- Неделя
- Месяц

Многоликая Убунта в 2020 году

+39 26,6k 56 142

Black [O]lives Matter: раса, криминал и огонь на поражение в США. Часть 2

+69 17,2k 33 169

Ситуация с PandaDoc в Беларуси: четверо сотрудников задержано, на топ-менеджеров компании завели уголовные дела

+68 19,2k 10 190

Германия, или Туда и Обратно — 1

+116 31,7k 169 438

От ASCII к Unicode: как появились кириллические домены и адреса электронной почты

Мерапост

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Публикации	Устройство сайта	Реклама
Регистрация	Новости	Для авторов	Тарифы
	Хабы	Для компаний	Контент
	Компании	Документы	Семинары
	Пользователи	Соглашение	Мегапроекты
	Песочница	Конфиденциальность	Мерч

