weakReference

Java Collections
4 уровень, 5 лекция

ОТКРЫТА

— И снова здравствуйте!

Сейчас я расскажу тебе еще про одну замечательную штуку — WeakReference – слабые ссылки.

Выглядит она почти так же, как и SoftReference:

Пример

```
1  //создание объекта Cat
2  Cat cat = new Cat();
3
4  //создание слабой ссылки на объект Cat
5  WeakReference<Cat> catRef = new WeakReference<Cat>(cat);
6
7  //теперь на объект ссылается только слабая ссылка catRef.
8  cat = null;
9
10 //теперь на объект ссылается еще и обычная переменная cat
11 cat = catRef.get();
12
13 //очищаем слабую ссылку
14 catRef.clear();
```

У слабой ссылки есть другая особенность.

Если на объект не осталось обычных ссылок и мягких ссылок, а только слабые ссылки, то этот объект является живым, но он будет уничтожен при ближайшей сборке мусора.

— А можно еще раз, в чем отличия между этими ссылками?

— Объект, который удерживает от смерти только SoftReference может пережить сколько угодно сборок мусора и скорее всего, будет уничтожен при нехватке программе памяти.

Объект, который удерживает от смерти только WeakReference не переживает ближайшей сборки мусора. Но пока она не произошла, его можно получить, вызвав метод get() у WeakReference и вызвать его методы или сделать что-нибудь еще.

— А если на объект ссылаются и SoftReference и WeakReference?

— Тут все просто. Если на объект есть хотя бы одна обычная ссылка – он считается живым. Такие ссылки, кстати, называются StrongReference.

Если на объект нет обычных ссылок, но есть SoftReference, то он – SoftReference.

Если на объект нет обычных ссылок и SoftReference, но есть WeakReference, то он – WeakReference.

Подумай сам, SoftReference защищает объект от удаления и гарантирует, что объект будет удален только при нехватке памяти. WeakReference удерживает объект до ближайшей сборки мусора. SoftReference удерживает от удаления сильнее.

— Ага. Вроде понятно

— Звучит серьезно!

— А то! WeakHashMap – это HashMap, у которого ключи – это слабые ссылки – WeakReference.

Т.е. ты добавляешь в такой HashMap объекты и работаешь с ними. Все как обычно.

Пока на объекты, которые ты хранишь в WeakHashMap в качестве ключей есть обычные (сильные или мягкие) ссылки, эти объекты будут живы.

Но, представь, что во всем приложении больше не осталось ссылок на эти объекты. Все что удерживает их от смерти – это несколько WeakReference внутри WeakHashMap. Тогда после ближайшей очистки мусора такие объекты исчезнут из WeakHashMap. Сами. Как будто их там и не было.

— Не уверен, что понял.

— Ты хранишь в WeakHashMap пары объектов – ключ и значение. Но WeakHashMap ссылается на ключи не прямо, а через WeakReference. Поэтому, когда объекты, используемые в качестве ключей, станут слабодостижимыми, они уничтожатся при ближайшей сборке мусора. А значит, из WeakHashMap автоматически удалятся и их значения.

В WeakHashMap очень удобно хранить дополнительную информацию к каким-то объектам.

Во-первых, ее очень легко получить, если использовать сам объект в качестве ключа.

Во-вторых, если объект будет уничтожен, из HashMap исчезнет и он, и все привязанные к нему данные.

Пример:

Пример

```
1  //создаем объект для хранения дополнительной информации о пользователе
2  WeakHashMap<User, StatisticInfo> userStatistics = new WeakHashMap<User, StatisticInfo>();
3
4  //кладем информацию о пользователе в userStatistics
5  User user = session.getUser();
6  userStatistics.put(user, new StatisticInfo (…));
7
8  //получаем информацию о пользователе из userStatistics
9  User user = session.getUser();
10 StatisticInfo statistics = userStatistics.get(user);
11
12 //удаление любой информации о пользователе из userStatistics
13 User user = session.getUser();
14 userStatistics.remove(user);
```

1. Внутри WeakHashMap ключи хранятся в виде WeakReference.
2. Как только объект user будет уничтожен сборщиком мусора, в WeakHashMap неявно вызовется метод remove(user) и любая «привязанная» к объекту user информация, будет удалена из WeakHashMap автоматически.

— Выглядит как мощный инструмент. А где можно его использовать?

— По обстоятельствам. Ну, допустим, у тебя в программе есть нить, которая отслеживает работу некоторых объектов-заданий и пишет информацию о них в лог. Тогда эта нить может хранить отслеживаемые объекты в таком WeakHashMap. Как только объекты станут не нужны, сборщик мусора удалит их, автоматически удалятся и ссылки на них из WeakHashMap.

— Звучит интересно. Сразу чувствуется, ну не писал я еще серьезных программ на Java, чтобы задействовать такие мощные механизмы. Но я буду расти в эту сторону, спасибо большое, Элли, за такой интересный урок.

+85

Комментарии (30) популярные новые старые

JavaCoder

Введите текст комментария

Андрей Коротков Уровень 38, Киров — 10 марта, 14:22

Второй пример немного неточен. Код в нем указан как бы в одном месте. Но данный код не будет работать в одном методе. Подразумевается, что строки 5-6 используются в одном месте, строки 9-10 где-то в другом месте (другой метод или даже другая программа), тоже и строки 13-14. Целесообразно разделить пример на 3 отдельных фрейма или вообще на 3 метода.

Ответить +2

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 22 января, 16:10

Получается, я с тем же успехом могу пользоваться обычной мапой, просто должен буду ещё вручную удалять объекты, которые уже не нужны, да?

Ответить 0

Stepan Уровень 27, Москва, Россия — 30 января, 20:19

Судя по тексту - нет. На сколько я понял самостоятельно удаленный объект вполне себе может пережить сборку мусора, если он ранее пережил множество очисток памяти, т.е. он далеко не первого поколения и потому считается долгожителем, а значит на него сборщик мусора обрати внимание не сразу. Тогда как "WeakReference не переживает ближайшей сборки мусора".

Ответить +1

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 30 января, 21:04

Хочешь сказать, что старый, но недостижимый ветеран сбора мусора будет ещё какое-то время держаться в памяти по непонятной причине? А смысл?

Ответить 0

Stepan Уровень 27, Москва, Россия — 31 января, 10:35

Цитата: "Сборщик Мусора очень — сложная и эффективная составляющая Java. Многие его части работают эвристически – на основе алгоритмов-догадок. Поэтому он часто «не слушается» пользователя."
Смысл вижу только один - не занимать время ЦП на тотальные заботы о мусоре. Я считаю, что сборщик мусора просматривает долгоживущие объекты несколько реже. Вообще надо найти время и разобраться со всеми премудростями работы сборщика мусора.

Ответить +2

Богдан Зінченко Frontend Developer в **iSolutions** — 3 января 2021, 23:44

Спасибо, Элли! Хоть кто-то вспомнил, что квест называется "Java Collections".

Ответить +17

Евгений Ведущий инженер в **ПАО Сбербанк** EXPERT — 27 июня 2020, 15:37

Вроде бы всё понятно.
Мапа, которую лучше всего использовать для метаданных (данных об объекте или о других данных). В качестве ключа используем целевой объект, и как только объект будет удалён сборщиком мусора, нам не надо париться с удалением информации о нём из мапы - она также автоматически будет удалена сборщиком мусора.

Ответить +15

Евгений Буш Программист в **Компания Nordside** EXPERT — 26 марта 2020, 12:42

А вот если между 8 и 9 строками вставить простую мапу, то память о васе не умрет даже после его null и прога зависнет в бесконечном цикле

```
1  Map<User,StatisticInfo>map = new HashMap<>();
2  map.put(vasya,new StatisticInfo("Its a vasya."));
3  map.put(vasilisa,new StatisticInfo("Its a vasilisa."));
```

ну если ссылку на мапу занулить, то вполне умрёт. А почему программа зависнет в бесконечном цикле? Где цикл то?

Ответить 0

Евгений Буш Программист в **Компания Nordside** EXPERT 7 апреля 2020, 10:39

цикл в 19 строке. мапу убивать нельзя, там могут быть другие полезные ссылки.

Ответить 0

Илья Backend Developer в **СберТех** 7 февраля 2021, 14:05

тут максимум 14 строк

Ответить +1

Галкин Юрий Уровень 41, Москва 12 января, 16:37

Строк явно не больше 18.

Ответить 0

Vorlock Уровень 31, Днепр, Украина 10 января 2020, 23:27

Спасибо, конечно, Элли!
но можно было бы и попонятней разъяснить)))

Ответить +16

Sergey Уровень 41, Санкт-Петербург 22 декабря 2019, 11:42

WeakHashMap

Рекомендую посмотреть с 23 минуты, очень доходчиво объяснено!

Ответить +42

Павел Уровень 36, Минск, Беларусь 5 января 2020, 09:55

спасибо! заценил

Ответить +1

Ihor Уровень 41, Киев, Украина 11 июня 2020, 12:18

годится!!!! понимание приходит понемногу)

Ответить +1

Aleksandr Liadov Уровень 40, Самара, Россия 11 ноября 2020, 21:53

Да, прикольно, ещё с примерами "где это пригодится", отлично зашло видео с 23ей минуты)

Ответить +1

Pig Man Главная свинья в **Свинарнике** 7 февраля 2021, 20:27

А для словаря, который самостоятельно удаляет запись по ключу, который стал недосягаем, нужно какое-то еще более доходчивое объяснение?

Ответить 0

Greatsky future developer в **future developer** 13 января, 15:43

Плюсую норм но я бы рекомендовал с первой минуты - по теме с 23 конечно - то в начале тоже ест что послушать

Ответить +1

Дмитрий Уровень 37, Нижний Новгород 14 ноября 2019, 01:14

Я вот как понял(надеюсь кому-то поможет): у нас есть объект, который мы хотим поместить в хэшмап, но так же, мы не хотим держать в голове и при неуверенности проверять, что все объекты-ключи в нашей хэшмапе существуют(!=null). Тогда мы берем WeakHashMap, которая реализована так, что, если на объект-ключ указывают только слабые ссылки, то он будет удален из WeakHashMap. Заметьте, что ссылок strong больше нет - это обязательное условие, ну, и soft, кстати говоря, тоже, иначе объект не будет удален сборщиком и будет дальше существовать в куче(heap), и как следствие останется в WeakHashMap.

Ответить +3

Владимир Ушкин Уровень 41, Армавир, Россия 13 ноября 2019, 20:55

Что за session ? Не пойму.

Ответить +2

Дмитрий Уровень 37, Нижний Новгород 14 ноября 2019, 00:54

ну, видимо сессия(интернет термин), из нее получаем какого то пользователя

Ответить 0

почему в примере не создали объект типа WeakReference и по идее его надо было добавить в Map?

Ответить

+1

CEO Уровень 34

31 марта 2019, 11:07

На сколько я понял:

```
1  // локальный переменная, и так умрет
2  User user = session.getUser();
3
4  // кладем в мапу объект и на него будет
5  // ссылаться только слабая ссылка
6  userStatistics.put(user, new StatisticInfo (…));
```

Т.е. единственная ссылка, которая имеется - из мапы, которая по определению является слабой

Ответить

+6

Yakobs Zingelgofer Уровень 41, Россия

2 апреля 2019, 15:48

но переменная ссылочного типа user никуда не делась, и как и раньше хранит ссылку на объект. Т.е. я вполне могу сделать вот так:

```
1  User user = session.getUser();
2  userStatistics.put(user, new StatisticInfo (…));
3  System.out.println(user.toString());
```

user как лежал во фрейме метода (в стэке), так и лежит, как указывал на объект в хипе так и указывает.
Для понятности: когда вызывается метод .put(), то он СОЗДАЕТ свою собственную локальную переменную, которая лежит в стэке во фрейме-put. А вот уже в эту переменную приходит ссылка на объект, который хранится в хипе. Локальные переменные из фрейма во фрейм не ходят.

Ответить

+1

CEO Уровень 34

6 апреля 2019, 20:21

я же написал, что переменная user - локальная переменная, как только мы выйдем из метода, она будет не доступна.

Ответить

0

Показать еще комментарии