



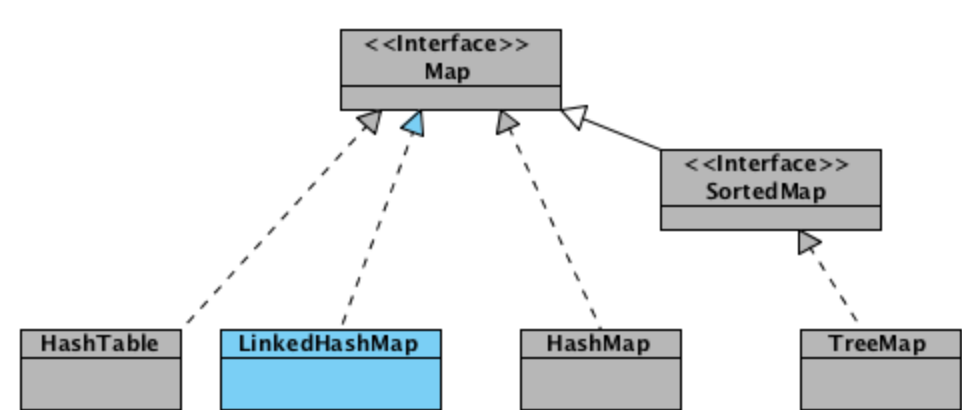
tarzan82 7 августа 2012 в 16:01

Структуры данных в картинках. LinkedHashMap

Java*

Привет Хабрачеловеки!

После затяжной паузы, я попробую продолжить визуализировать структуры данных в Java. В предыдущих статьях были замечены: [ArrayList](#), [LinkedList](#), [HashMap](#). Сегодня заглянем внутрь к LinkedHashMap.



Из названия можно догадаться что данная структура является симбиозом связанных списков и хэш-мапов. Действительно, `LinkedHashMap` расширяет класс `HashMap` и реализует интерфейс `Map`, но что же в нем такого от связанных списков? Давайте будем разбираться.

Создание объекта

```
Map<Integer, String> linkedHashMap = new LinkedHashMap<Integer, String>();
```

Footprint{Objects=3, References=26, Primitives=[int x 4, float, boolean]}
size: 160 bytes

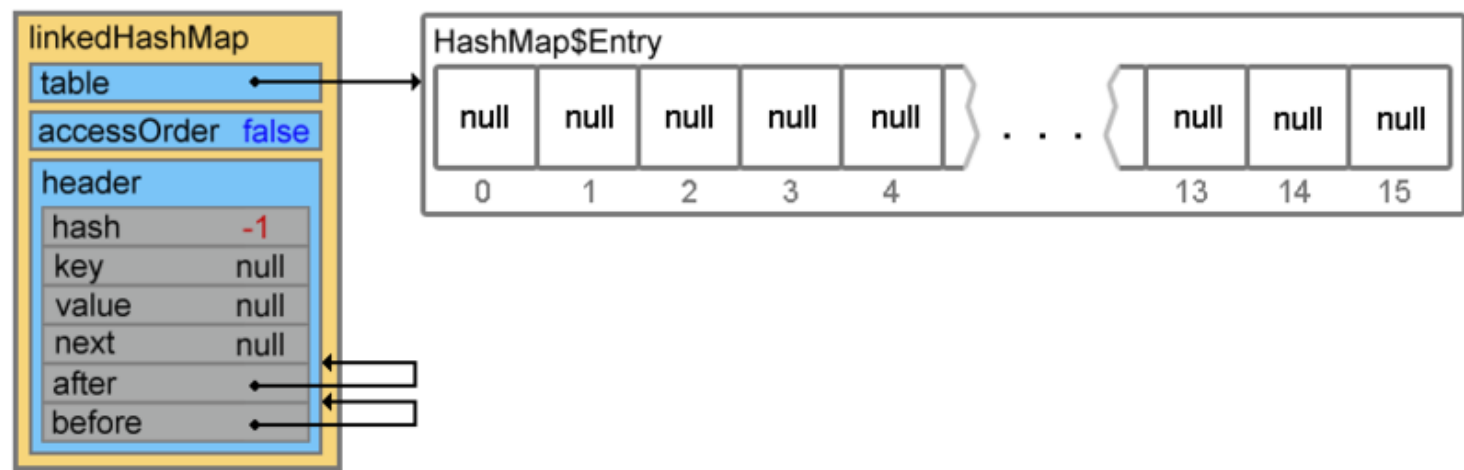
Только что созданный объект `linkedHashMap`, помимо свойств унаследованных от `HashMap` (такие как `table`, `loadFactor`, `threshold`, `size`, `entrySet` и т.п.), так же содержит два доп. свойства:

- **header** — «голова» двусвязного списка. При инициализации указывает сам на себя;
- **accessOrder** — указывает каким образом будет осуществляться доступ к элементам при использовании итератора. При значении **true** — по порядку последнего доступа (об этом в [конце](#) статьи). При значении **false** доступ осуществляется в том порядке, в каком элементы были вставлены.

Конструкторы класса `LinkedHashMap` достаточно скучные, вся их работа сводится к вызову конструктора родительского класса и установке значения свойству **accessOrder**. А вот инициализация свойства **header** происходит в переопределенном методе **init()** (*теперь становится понятно для чего в конструкторах класса **HashMap** присутствует вызов этой, ничегонеделющей функции*).

```
void init()
{
    header = new Entry<K,V>(-1, null, null, null);
    header.before = header.after = header;
}
```

Новый объект создан, свойства проинициализированы, можно переходить к добавлению элементов.



Добавление элементов

```
linkedHashMap.put(1, "obj1");
```

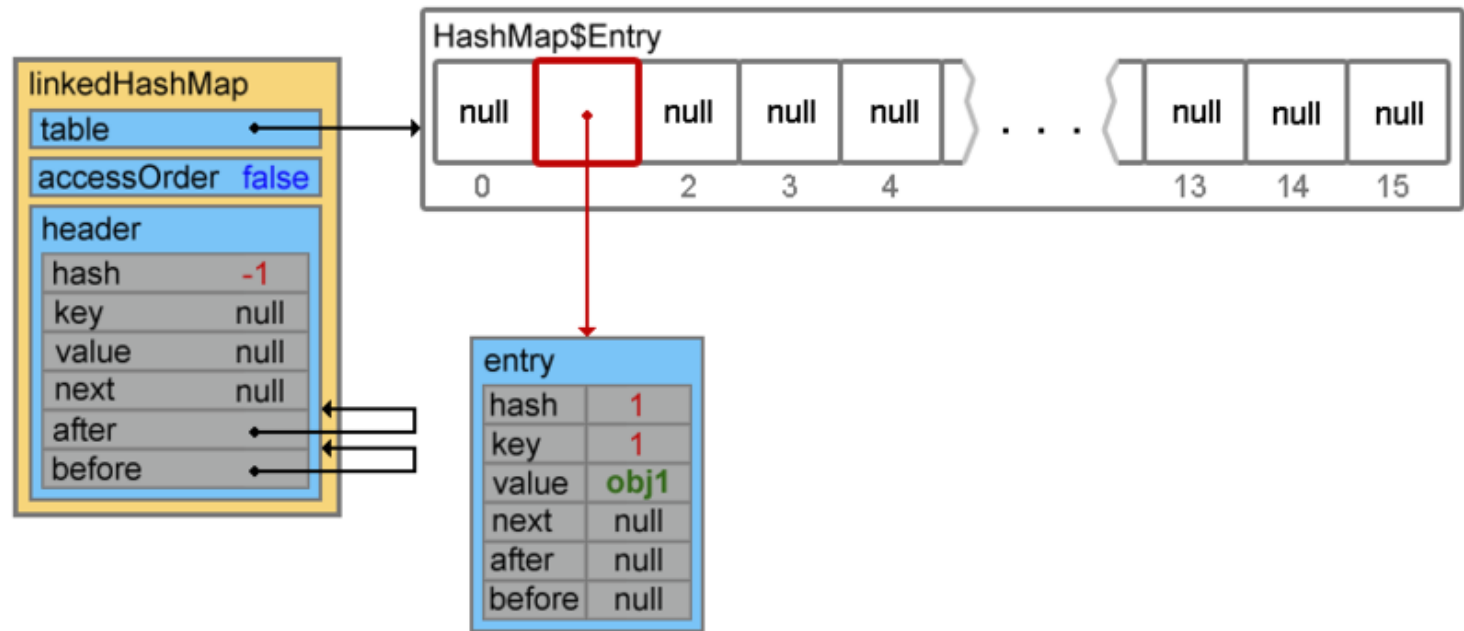
Footprint{Objects=7, References=32, Primitives=[char x 4, int x 9, float, boolean]}

size: 256 bytes

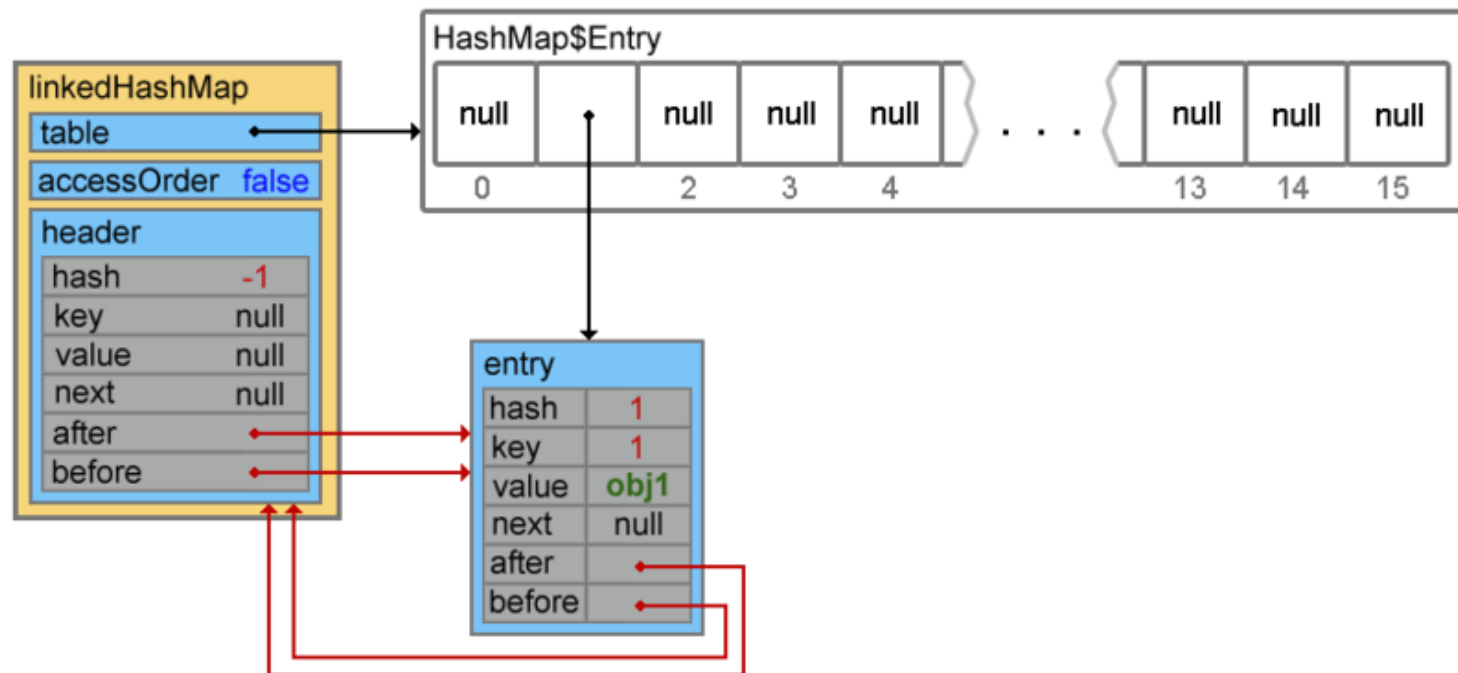
При добавлении элемента, первым вызывается метод **createEntry(hash, key, value, bucketIndex)** (по цепочке **put()** -> **addEntry()** -> **createEntry()**)

```
void createEntry(int hash, K key, V value, int bucketIndex)
{
    HashMap.Entry<K,V> old = table[bucketIndex];
    Entry<K,V> e = new Entry<K,V>(hash, key, value, old);
    table[bucketIndex] = e;
    e.addBefore(header);
    size++;
}
```

первые три строки добавляют элемент (при коллизиях добавление произойдет в начало цепочки, далее мы это увидим)



четвертая строка переопределяет ссылки двусвязного списка



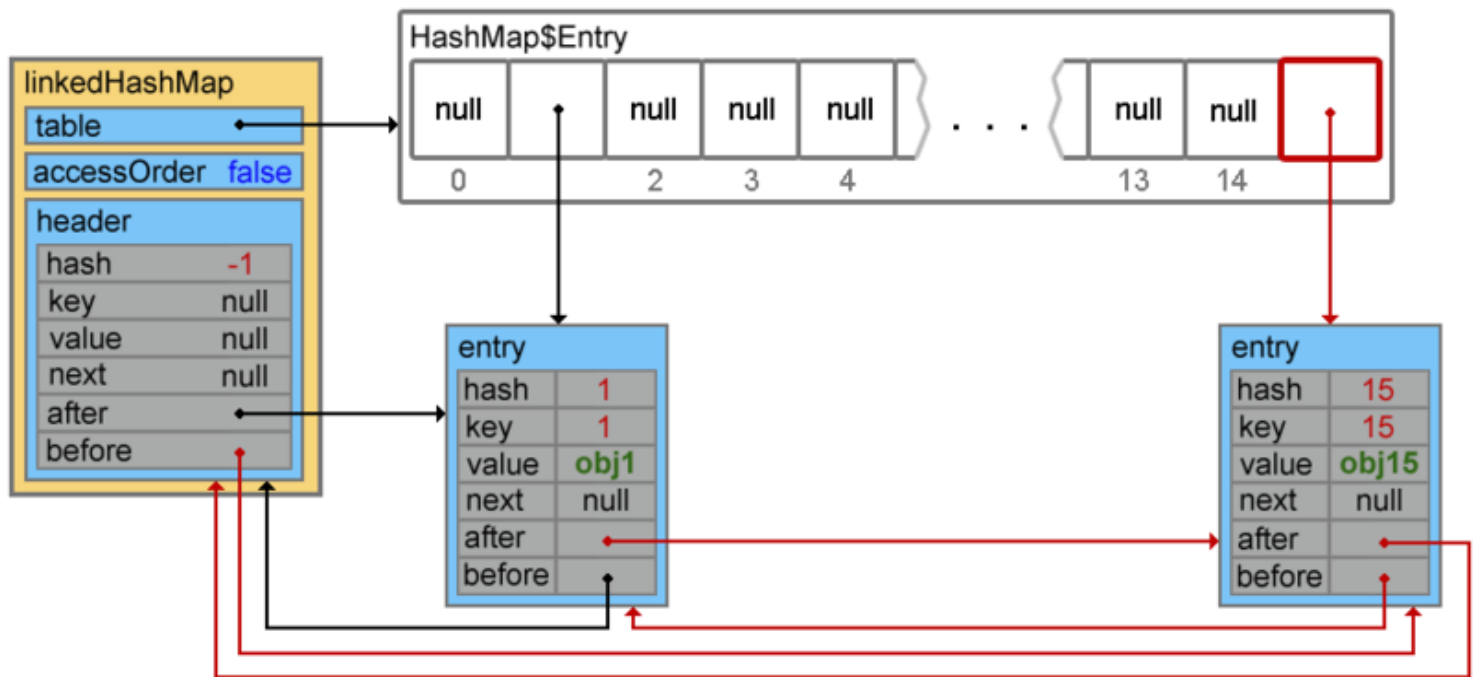
Всё что дальше происходит в методе **addEntry()** либо не представляет «функционального интереса»¹ либо повторяет функционал родительского класса.

Добавим еще парочку элементов

```
linkedHashMap.put(15, "obj15");
```

Footprint{Objects=11, References=38, Primitives=[float, boolean, char x 9, int x 14]}

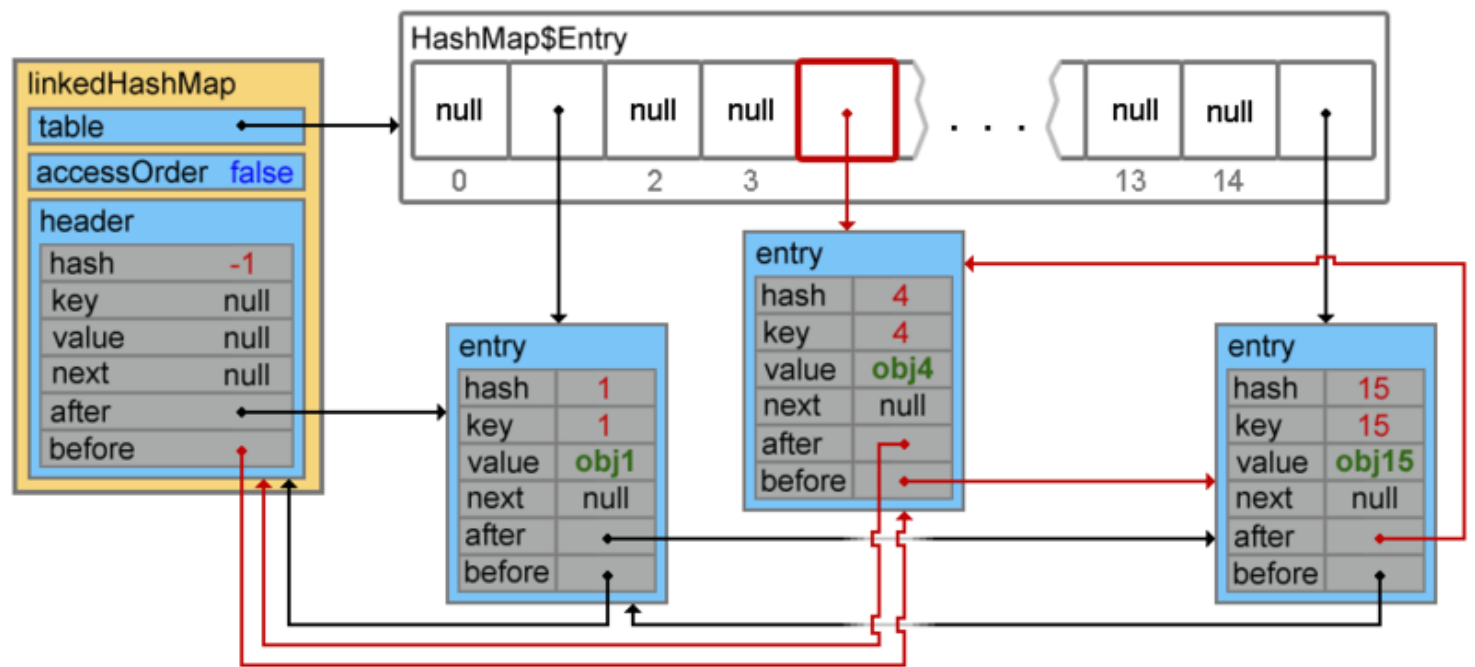
size: 352 bytes



```
linkedHashMap.put(4, "obj4");
```

Footprint{Objects=11, References=38, Primitives=[float, boolean, char x 9, int x 14]}

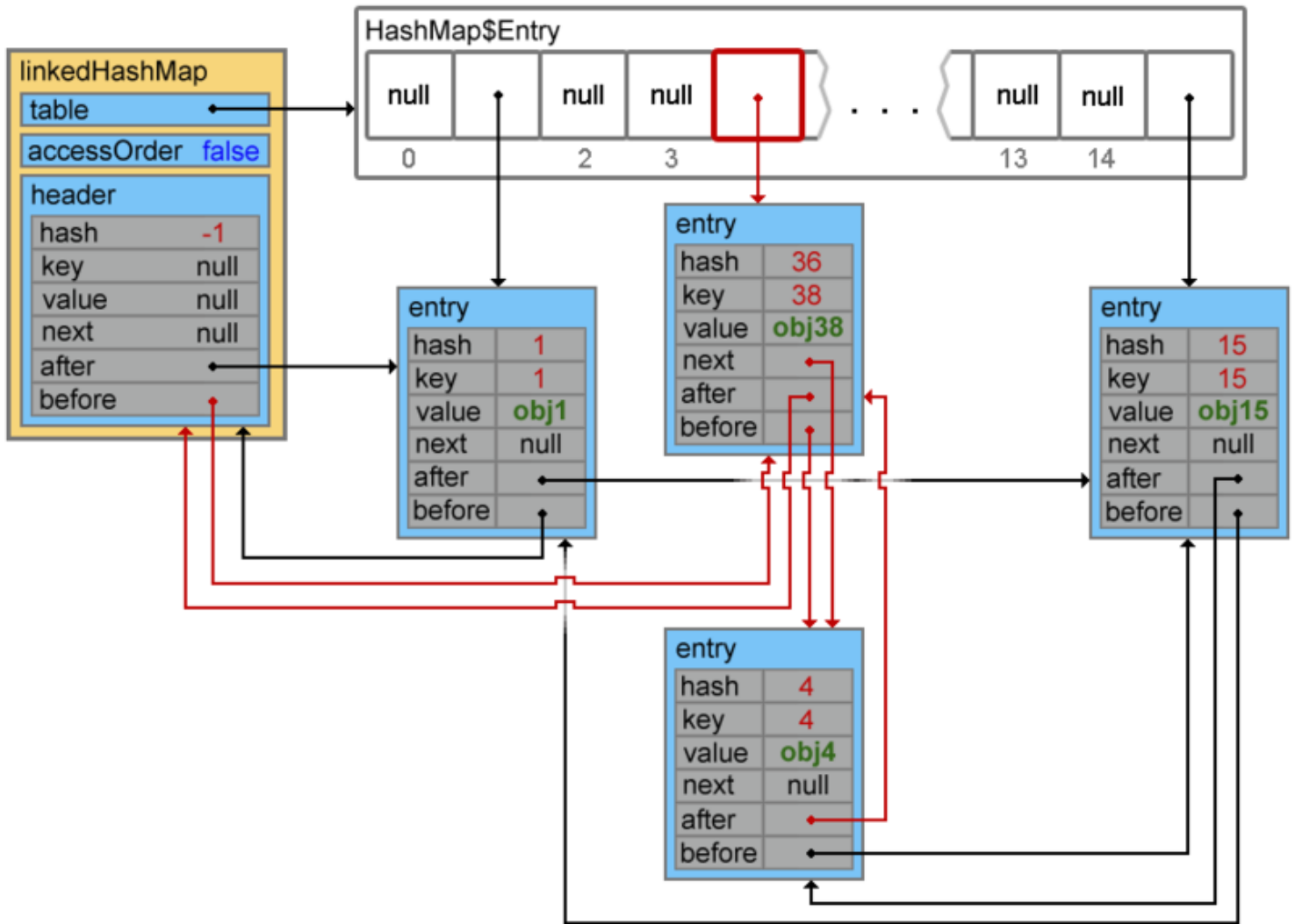
size: 448 bytes



При добавлении следующего элемента происходит коллизия, и элементы с ключами 4 и 38 образуют цепочку

```
linkedHashMap.put(38, "obj38");
```

Footprint{Objects=20, References=51, Primitives=[float, boolean, char x 18, int x 24]}
size: 560 bytes

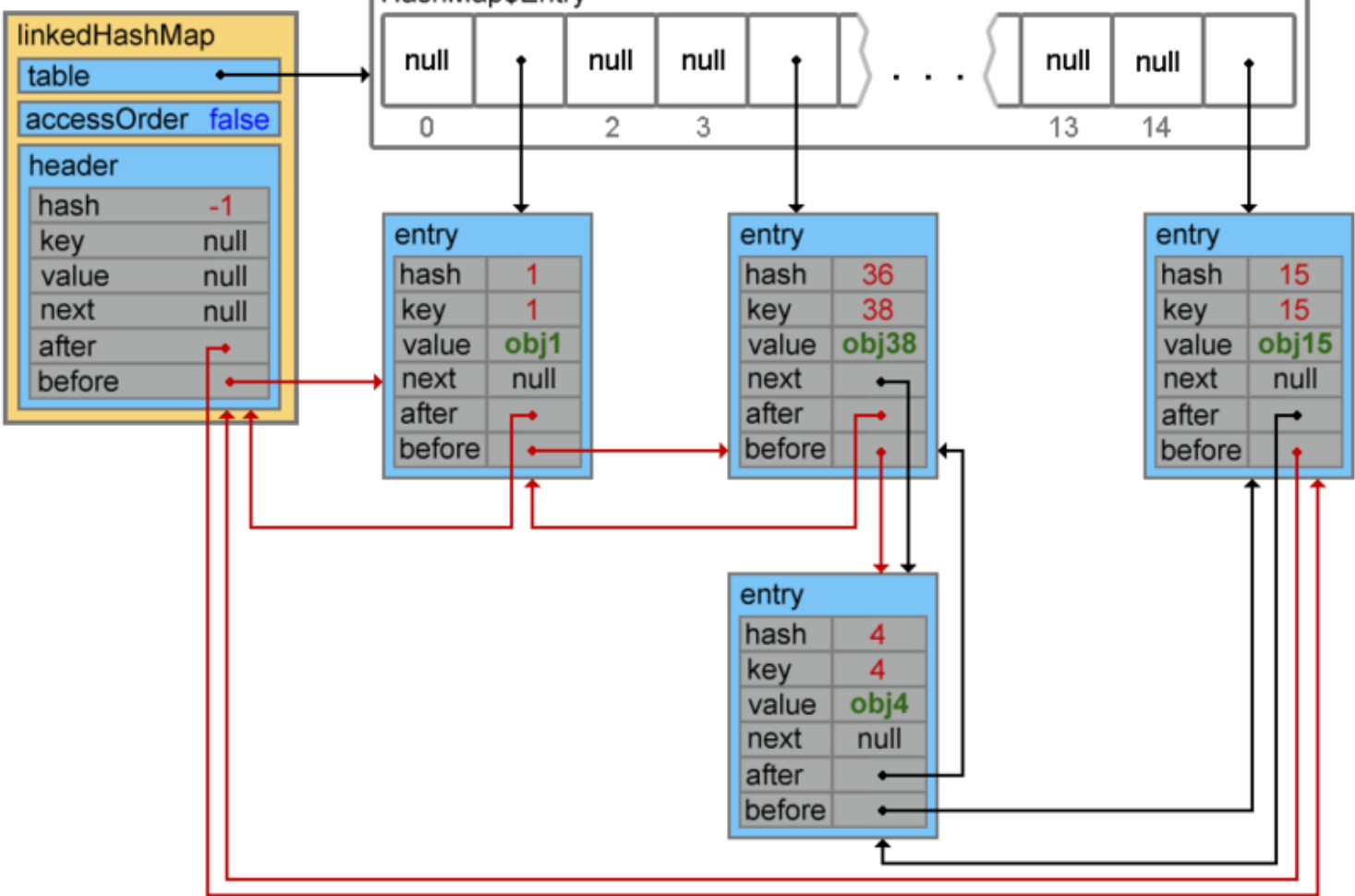


Обращаю ваше внимание, что в случае повторной вставки элемента (элемент с таким ключом уже существует) порядок доступа к элементам не изменится.

accessOrder == true

А теперь давайте рассмотрим пример когда свойство **accessOrder** имеет значение **true**. В такой ситуации поведение **LinkedHashMap** меняется и при вызовах методов **get()** и **put()** порядок элементов будет изменен — элемент к которому обращаемся будет помещен в конец.

```
Map<Integer, String> linkedHashMap = new LinkedHashMap<Integer, String>(15, 0.75f, true) {  
    put(1, "obj1");  
    put(15, "obj15");  
    put(4, "obj4");  
    put(38, "obj38");  
};  
// {1=obj1, 15=obj15, 4=obj4, 38=obj38}  
  
linkedHashMap.get(1); // or linkedHashMap.put(1, "Object1");  
// {15=obj15, 4=obj4, 38=obj38, 1=obj1}
```



Итераторы

Всё достаточно банально:

```
// 1.
Iterator<Entry<Integer, String>> itr1 = linkedHashMap.entrySet().iterator();
while (itr1.hasNext()) {
    Entry<Integer, String> entry = itr1.next();
    System.out.println(entry.getKey() + " = " + entry.getValue());
}

// 2.
Iterator<Integer> itr2 = linkedHashMap.keySet().iterator();
while(itr2.hasNext())
    System.out.println(itr2.next());

// 3.
Iterator<String> itr3 = linkedHashMap.values().iterator();
while (itr3.hasNext())
    System.out.println(itr3.next());
```

Ну и не забывайте про fail-fast. Коли уж начали перебор элементов — не изменяйте содержимое или заранее позаботьтесь о синхронизации.

Вместо итогов

Данная структура может слегка уступать по производительности родительскому **HashMap**, при этом время выполнения операций **add()**, **contains()**, **remove()** остается константой — $O(1)$. Понадобится чуть больше места в памяти для хранения элементов и их связей, но это совсем небольшая плата за дополнительные фишечки.

Вообще, из-за того что всю основную работу на себя берет родительский класс, серьезных отличий в реализации **HashMap** и **LinkedHashMap** не много. Можно упомянуть о парочке мелких:

- Методы **transfer()** и **containsValue()** устроены чуть проще из-за наличия двунаправленной связи между элементами;
- В классе **LinkedHashMap.Entry** реализованы методы **recordRemoval()** и **recordAccess()** (тот

Ссылки

Исходник LinkedHashMap
Исходники JDK OpenJDK & trade 6 Source Release — Build b23
Инструменты для замеров — memory-measurer и Guava (Google Core Libraries).

¹ — Вызов метода **removeEldestEntry(Map.Entry eldest)** всегда возвращает **false**.
Предполагается, что данный метод может быть переопределен для каких-либо нужд, например, для реализации кэширующих структур на основе **Map** (см. [ExpiringCache](#)). После того как **removeEldestEntry()** станет возвращать **true**, самый старый элемент будет удален при превышении макс. количества элементов.


Теги: java, LinkedHashMap, структуры данных

Хабы: Java

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электронпочта



113

0


Карма

Рейтинг

@tarzan82

Пользователь


Комментарии 14

- 

Monnoroch


07.08.2012 в 16:31



Блин, я вот даже и так знаю, как они устроены, а ваши картинки долго разбираю. Может это личное, но вы попробуйте подумать, как их рисовать позргономичнее.



+1

Ответить





- 

tarzan82


07.08.2012 в 17:26



Поясните пожалуйста, что конкретно вас не устроило в картинках?



0

Ответить





- 

Monnoroch


07.08.2012 в 17:51



Ну я же пояснил: долго разбирался. Я не дизайнер и не могу сказать, почему, просто сложные они.



0

Ответить





- 

cheremin

07.08.2012 в 18:07


Думаю, я могу высказать более конкретное пожелание: картинки непонятны потому, что на них вы пытаетесь изобразить все сразу. Мне кажется, было бы гораздо удобнее видеть схемы итеративно, по смысловым слоям. Т.е. на первой схеме только то, что нужно для первой схемы — например, не надо там рисовать поля, про которые пока не идет речь. На каждой новой схеме актуальные вещи (про которые сейчас говорим) насыщенным цветом, уже не актуальные блеклым, те, которые еще не актуальны — их вообще нет.


Это много работы, понятно. Но мне кажется результат будет того стоить.



+3

Ответить





- tarzan82

07.08.2012 в 21:02

Спасибо, постараюсь учесть ваши замечания

0

Ответить

RusMikle

31.08.2020 в 22:46

всё там понятно. Я сам подумал как ещё понятнее отобразить и не придумал. Просто надо немного сконцентрироваться.

0

Ответить

cheremin

07.08.2012 в 18:08

Да, и спасибо вам за работу. Продолжайте цикл, интересно

+2

Ответить

atomicus

07.08.2012 в 20:02

Благодарю, как всегда отлично и исчерпывающе. Не особо разделяю претензий к иллюстрациям, т.к. мне кажется, что рассматривать только картинки — это «скользкий» путь для обучения (кто-то называл это «информационным фаст-фудом»). И только вкупе с текстом картинки в прямом смысле «разжевывают» материал. Хотя доля истины есть, и поэтому пару мыслей (я тоже не дизайнер) все же попробую озвучить:

1. Шесть цветов для объекта — это как-то многовато (голубой, серый, черный, красный, зеленый, плюс темно-серая рамка). Можно попробовать рисовать объекты на белом фоне и постараться минимизировать число используемых цветов.

2. Попробовать сменить шрифт и убрать использование полужирного написания по умолчанию. Тогда измененные данные на каждой итерации можно будет выделять bold-шрифтом, чтобы сделать их заметнее. Сейчас масса инфографики вокруг, думаю вполне можно позаимствовать оттуда удачный шрифт.

3. Для более легкого восприятия глазом некоторые (не все) прямые углы можно сгладить. Например, углы на поворотах линий со стрелками можно сделать закругленными.

Не уверен, что данные изменения увеличат число людей, которым статья принесет реальную пользу, но то что плюсов и лайков у статей с красивыми картинками больше — это факт.

+1

Ответить

tarzan82

07.08.2012 в 21:19

Спасибо, приму к сведению ваши пункты

0

Ответить

atomAltera

07.08.2012 в 21:33

Спасибо за статью

0

Ответить

Walrus

08.08.2012 в 12:49

Первая картинка прям антиквариат. Где NavigableMap?

0

Ответить

Biox

13.08.2012 в 01:10

Можете объяснить, не совсем понятно почему произошла коллизия на 4-м элементе? Из-за того что превышен изначальный размер LinkedHashMap? Или из-за чего?

Это всего лишь мои догадки, т.к. лишь в разделе Access Order приведен пример вызова конструктора.

0

Ответить

tarzan82

13.08.2012 в 09:52

У элементов с ключами 4 и 38 хэш-коды равны 4 и 36 соответственно. Но, вычисление индекса по этим хэш-кодам дало одинаковый результат — 4. При другом размере table этой коллизии может не

быть, но могут быть другие.


Могу посоветовать предыдущую статью про [HashMap](#), в ней более подробно описаны функции получения хэш-кода и индексов.


 0

Ответить







BioX 27.08.2012 в 06:46 

Теперь все стало понятно: `index = h & (length — 1)`. Спасибо. Мне стоило начать со статьи про `HashMap` :)

 0

Ответить






Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

2 января в 12:00

LJV: Чему нас может научить визуализация структур данных в Java

 +85

 23K

 236

 11 +11

4 сентября 2021 в 16:46

Как снизить зависимость кода от структуры данных?

 +2

 8.9K


 59

 41 +41


18 июля 2021 в 16:55

Две открытые библиотеки для обучения байесовских сетей и идентификации структуры данных

 +6

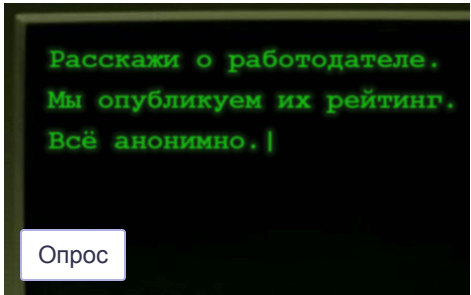
 2.3K

 48

 1 +1

МИНУТОЧКУ ВНИМАНИЯ

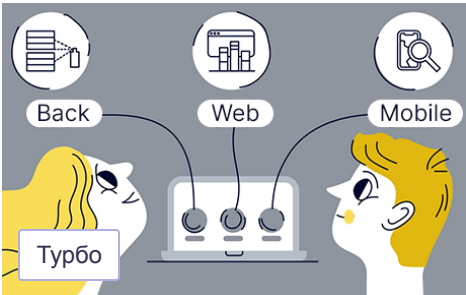
[Разместить](#)



Третье хабраисследование ru-IT-брендов



Помогите динозаврику добыть контент по Data Mining



Как проходит интервью QA-инженеров в Тинькофф

ВОПРОСЫ И ОТВЕТЫ

Как написать метод который принимает строку, преобразует ее и возвращает строку?

Java · Простой · 2 ответа

Ошибка прокси в автотестах Selenium, как убрать?

Java · Средний · 0 ответов

Стоит ли использовать Object как тип возвращаемого значение метода в java?

Java · Простой · 2 ответа

Spring Security. Как дать возможность юзеру изменять только свои объекты?

Java · Простой · 1 ответ

Делаю changelog на pgsql выходит ошибка, как решить?

PostgreSQL · Средний · 2 ответа

[Больше вопросов на Хабр Q&A](#)

вчера в 20:49

Что не так с ДЭГ Москвы на этот раз?

+264 27K 34 99 +99

вчера в 23:33

Смерть Mozilla — это смерть открытого Интернета

+88 23K 36 219 +219

вчера в 16:03

Бифуркация (фантастический рассказ)

+23 3.1K 12 15 +15

сегодня в 07:06

ЦБ хочет компенсировать страдания российских инвесторов за счет «недружественных» нерезидентов

+21 3.9K 5 3 +3

сегодня в 11:30

Linkkraft: offline-first браузер, который организует открытые вкладки и персональные заметки

+20 1K 10 22 +22

DAST ist fantastisch: отечественный динамический анализатор к взлету готов

Мегапост

ЧИТАЮТ СЕЙЧАС

Смерть Mozilla — это смерть открытого Интернета

23K 219 +219

Что не так с ДЭГ Москвы на этот раз?

27K 99 +99

Активность найма на IT-рынке в августе 2022

19K 15 +15

Крякнул софт? Суши сухари

28K 132 +132

Американские компании начали убирать кнопки Facebook** для авторизации со своих сайтов

5.9K 8 +8

Сага о SEO: большая статья о серверном рендеринге и не только

Турбо

РАБОТА

Java разработчик
425 вакансий

Ваш аккаунт

Войти
Регистрация

Разделы

Публикации
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные
программы
Стартапам
Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию