## 一、ConcurrentLinkedDeque介绍

ConcurrentLinkedDeque 是双向链表结构的无界并发队列。从JDK 7开始加入到J.U.C的行列中。使用CAS实现并发安全，与 ConcurrentLinkedQueue 的区别是该阻塞队列同时支持FIFO和FILO两种操作方式，即可以从队列的头和尾同时操作(插入/删除)。适合“多生产，多消费”的场景。内存一致性遵循对 ConcurrentLinkedDeque 的插入操作先行发生于(happen-before)访问或移除操作。相较于 ConcurrentLinkedQueue，ConcurrentLinkedDeque 由于是双端队列，所以在操作和概念上会更加复杂。

注意：size方法不是一个准确的操作

一、ConcurrentLinkedDeque介绍

ConcurrentLinkedDeque 是双向链表结构的无界并发队列。从JDK 7开始加入到J.U.C的行列中。使用CAS实现并发安全，与 ConcurrentLinkedQueue 的区别是该阻塞队列同时支持FIFO和FILO两种操作方式，即可以从队列的头和尾同时操作(插入/删除)。适合“多生产，多消费”的场景。内存一致性遵循对 ConcurrentLinkedDeque 的插入操作先行发生于(happen-before)访问或移除操作。相较于 ConcurrentLinkedQueue，ConcurrentLinkedDeque 由于是双端队列，所以在操作和概念上会更加复杂。

注意：size方法不是一个准确的操作

二、ConcurrentLinkedDeque的一些方法介绍

1、add(E e)：在此deque的尾部插入指定的元素，返回值为Boolean。

2、addFirst(E e)：在此deque前面插入指定的元素。

3、addLast(E e)：在此deque的末尾插入指定的元素。

4、clear()：从这个deque中删除所有的元素。

5、contains(Object o)：返回 true如果这个deque包含至少一个元素 e ，返回值为Boolean。

6、descendingIterator()：以相反的顺序返回此deque中的元素的迭代器，返回值为Iterator。

7、element()：检索但不删除由此deque表示的队列的头部（换句话说，该deque的第一个元素）。

8、getFirst()：检索，但不删除，这个deque的第一个元素。

9、getLast()：检索，但不删除，这个deque的最后一个元素。

10、isEmpty()：如果此集合不包含元素，则返回 true 。

11、iterator()：以正确的顺序返回此deque中的元素的迭代器，返回值为Iterator 。

12、offer(E e)：在此deque的尾部插入指定的元素，返回值为boolean。

13、offerFirst(E e)：在此deque前面插入指定的元素，返回值为boolean。

14、offerLast(E e)：在此deque的末尾插入指定的元素，返回值为boolean。

15、peek()：检索但不删除由此deque表示的队列的头（换句话说，该deque的第一个元素），如果此deque为空，则返回 null 。

16、peekFirst()：检索但不删除此deque的第一个元素，如果此deque为空，则返回 null 。

17、peekLast()：检索但不删除此deque的最后一个元素，如果此deque为空，则返回 null 。

18、poll()：检索并删除由此deque表示的队列的头部（换句话说，该deque的第一个元素），如果此deque为空，则返回 null 。

19、pollFirst()：检索并删除此deque的第一个元素，如果此deque为空，则返回 null 。

20、pollLast()：检索并删除此deque的最后一个元素，如果此deque为空，则返回 null 。

21、pop()：从这个deque表示的堆栈中弹出一个元素。

22、push(E e)：将元素推送到由此deque代表的堆栈（换句话说，在该deque的头部），如果可以立即执行，而不违反容量限制，则抛出 IllegalStateException如果当前没有可用空间）。

23、remove()：检索并删除由此deque表示的队列的头（换句话说，该deque的第一个元素）。

24、remove(Object o)：删除第一个元素 e ，使 o.equals(e) ，如果这样一个元素存在于这个deque，返回值为boolean。

25、removeFirst()：检索并删除此deque的第一个元素。

26、removeFirstOccurrence(Object o)：删除第一个元素 e ，使 o.equals(e) ，如果这样一个元素存在于这个deque，返回值为boolean。

27、removeLast()：检索并删除此deque的最后一个元素。

28、size()：返回此deque中的元素数。

三、Java代码示例

package chapter3.concurrentlinkeddeque;

import java.util.Iterator;

import java.util.concurrent.ConcurrentLinkedDeque;

import java.util.concurrent.ConcurrentLinkedQueue;

/\*\*

\* @author czd

\*/

public class ConcurrentLinkedDequeTest {

public static void main(String[] args) {

/\*\*

\* 构造方法

\* ConcurrentLinkedDeque():

\* 构造一个空的德克。

\* ConcurrentLinkedDeque(Collection<? extends E> c):

\* 构造最初包含给定集合的元素的deque，以集合的迭代器的遍历顺序添加。

\*/

/\*\*

\* 1、add(E e):在此deque的尾部插入指定的元素。

\*

\* 2、iterator()：以正确的顺序返回此deque中的元素的迭代器。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque = new ConcurrentLinkedDeque<>();

Boolean addBoolean = concurrentLinkedDeque.add(5);

System.out.println("是否插入deque尾部成功？" + addBoolean);

Iterator<Integer> iterator = concurrentLinkedDeque.iterator();

while (iterator.hasNext()){

System.out.println("iterator的结果：" + iterator.next());

}

/\*\*

\* 3、addFirst(E e):在此deque前面插入指定的元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque1 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque1.addFirst(1);

concurrentLinkedDeque1.addFirst(2);

concurrentLinkedDeque1.addFirst(3);

Iterator<Integer> iterator1 = concurrentLinkedDeque1.iterator();

while (iterator1.hasNext()){

System.out.println("iterator的结果：" + iterator1.next());

}

/\*\*

\* 4、addLast(E e):在此deque的末尾插入指定的元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque2 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque2.addLast(7);

concurrentLinkedDeque2.addLast(8);

concurrentLinkedDeque2.addLast(9);

System.out.println("=================");

Iterator<Integer> iterator2 = concurrentLinkedDeque2.iterator();

while (iterator2.hasNext()){

System.out.println("iterator的结果：" + iterator2.next());

}

/\*\*

\* 5、contains(Object o)：返回 true如果这个deque包含至少一个元素 e ，返回值为Boolean。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque3 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque3.add(7);

concurrentLinkedDeque3.add(8);

concurrentLinkedDeque3.add(9);

Boolean containsBoolean = concurrentLinkedDeque3.contains(8);

System.out.println("concurrentLinkedDeque3是否包含8: " + containsBoolean);

/\*\*

\* 6、getFirst()：检索，但不删除，这个deque的第一个元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque4 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque4.add(1);

concurrentLinkedDeque4.add(2);

concurrentLinkedDeque4.add(3);

Integer getFirstResult = concurrentLinkedDeque4.getFirst();

System.out.println("deque的第一个元素：" + getFirstResult);

/\*\*

\* 7、getLast():检索，但不删除，这个deque的最后一个元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque5 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque5.add(4);

concurrentLinkedDeque5.add(5);

concurrentLinkedDeque5.add(6);

Integer getLast = concurrentLinkedDeque5.getLast();

System.out.println("deque的最后一个元素：" + getLast);

/\*\*

\* 8、isEmpty()：如果此集合不包含元素，则返回 true 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque6 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque6.add(7);

concurrentLinkedDeque6.add(8);

concurrentLinkedDeque6.add(9);

Boolean isEmptyBoolean = concurrentLinkedDeque5.isEmpty();

System.out.println("deque是否为空：" + isEmptyBoolean);

/\*\*

\* 9、offer(E e):在此deque的尾部插入指定的元素,返回值为Boolean。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque7 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque7.offer(7);

concurrentLinkedDeque7.offer(8);

concurrentLinkedDeque7.offer(9);

System.out.println("=================");

Iterator<Integer> iterator3 = concurrentLinkedDeque7.iterator();

while (iterator3.hasNext()){

System.out.println("iterator的结果：" + iterator3.next());

}

/\*\*

\* 10、offerFirst(E e):在此deque前面插入指定的元素,返回值为Boolean。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque8 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque8.offerFirst(7);

concurrentLinkedDeque8.offerFirst(8);

concurrentLinkedDeque8.offerFirst(9);

System.out.println("=================");

Iterator<Integer> iterator4 = concurrentLinkedDeque8.iterator();

while (iterator4.hasNext()){

System.out.println("iterator的结果：" + iterator4.next());

}

/\*\*

\* 11、offerLast(E e):在此deque的末尾插入指定的元素,返回值为Boolean。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque9 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque9.offerLast(7);

concurrentLinkedDeque9.offerLast(8);

concurrentLinkedDeque9.offerLast(9);

System.out.println("=================");

Iterator<Integer> iterator5 = concurrentLinkedDeque9.iterator();

while (iterator5.hasNext()){

System.out.println("iterator的结果：" + iterator5.next());

}

/\*\*

\* 12、peek():检索但不删除由此deque表示的队列的头（换句话说，该deque的第一个元素），

\* 如果此deque为空，则返回 null 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque10 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque10.offer(7);

concurrentLinkedDeque10.offer(8);

concurrentLinkedDeque10.offer(9);

System.out.println("队列的头: " + concurrentLinkedDeque10.peek());

/\*\*

\* 13、peekFirst():检索但不删除此deque的第一个元素，如果此deque为空，则返回 null 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque11 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque11.offer(4);

concurrentLinkedDeque11.offer(5);

concurrentLinkedDeque11.offer(6);

System.out.println("队列的头: " + concurrentLinkedDeque11.peekFirst());

/\*\*

\* 14、peekLast():检索但不删除此deque的最后一个元素，如果此deque为空，则返回 null

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque12 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque12.offer(1);

concurrentLinkedDeque12.offer(2);

concurrentLinkedDeque12.offer(3);

System.out.println("队列的最后一个元素: " + concurrentLinkedDeque12.peekLast());

/\*\*

\* 15、 poll():检索并删除由此deque表示的队列的头部（换句话说，该deque的第一个元素），

\* 如果此deque为空，则返回 null 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque13 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque13.addFirst(1);

concurrentLinkedDeque13.addFirst(2);

concurrentLinkedDeque13.addFirst(3);

Integer addFirstResult = concurrentLinkedDeque13.poll();

System.out.println("addFirstResult: " + addFirstResult);

System.out.println("concurrentLinkedDeque13是否还包含3？" + concurrentLinkedDeque13.contains(3));

/\*\*

\* 16、pollFirst():检索并删除此deque的第一个元素，如果此deque为空，则返回 null 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque14 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque14.addFirst(4);

concurrentLinkedDeque14.addFirst(5);

concurrentLinkedDeque14.addFirst(6);

Integer pollFirstResult = concurrentLinkedDeque14.pollFirst();

System.out.println("pollFirstResult: " + pollFirstResult);

System.out.println("concurrentLinkedDeque14是否还包含6？" + concurrentLinkedDeque14.contains(6));

/\*\*

\* 17、pollLast():检索并删除此deque的最后一个元素，如果此deque为空，则返回 null 。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque15 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque15.addFirst(7);

concurrentLinkedDeque15.addFirst(8);

concurrentLinkedDeque15.addFirst(9);

Integer pollLast = concurrentLinkedDeque15.pollLast();

System.out.println("pollLast: " + pollLast);

System.out.println("concurrentLinkedDeque15是否还包含7？" + concurrentLinkedDeque15.contains(7));

/\*\*

\* 18、pop():从这个deque表示的堆栈中弹出一个元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque16 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque16.addLast(7);

concurrentLinkedDeque16.addLast(8);

concurrentLinkedDeque16.addLast(9);

Integer popResult = concurrentLinkedDeque16.pop();

System.out.println("popResult: " + popResult);

System.out.println("concurrentLinkedDeque16是否还包含7？" + concurrentLinkedDeque16.contains(7));

/\*\*

\* 19、push(E e):将元素推送到由此deque代表的堆栈（换句话说，在该deque的头部），

\* 如果可以立即执行，而不违反容量限制，

\* 则抛出 IllegalStateException如果当前没有可用空间）。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque17 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque17.push(7);

concurrentLinkedDeque17.push(8);

concurrentLinkedDeque17.push(9);

System.out.println("=================");

Iterator<Integer> iterator6 = concurrentLinkedDeque17.iterator();

while (iterator6.hasNext()){

System.out.println("iterator的结果：" + iterator6.next());

}

/\*\*

\* 20、remove():检索并删除由此deque表示的队列的头（换句话说，该deque的第一个元素）。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque18 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque18.push(1);

concurrentLinkedDeque18.push(2);

concurrentLinkedDeque18.push(3);

Integer removeResult = concurrentLinkedDeque18.remove();

System.out.println("removeResult: " + removeResult);

System.out.println("concurrentLinkedDeque18是否还包含3？" + concurrentLinkedDeque18.contains(3));

/\*\*

\* 21、remove(Object o):删除第一个元素 e ，如果这样一个元素存在于这个deque,返回值为Boolean。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque19 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque19.push(4);

concurrentLinkedDeque19.push(5);

concurrentLinkedDeque19.push(6);

Boolean removeBoolean = concurrentLinkedDeque19.remove(5);

System.out.println("是否移除5成功？" + removeBoolean);

System.out.println("concurrentLinkedDeque19是否还包含3？" + concurrentLinkedDeque19.contains(5));

/\*\*

\* 22、removeFirst():检索并删除此deque的第一个元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque20 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque20.push(7);

concurrentLinkedDeque20.push(8);

concurrentLinkedDeque20.push(9);

Integer removeFirstResult = concurrentLinkedDeque20.removeFirst();

System.out.println("deque的第一个元素: " + removeFirstResult);

/\*\*

\* 23、removeLast():检索并删除此deque的最后一个元素。

\*/

ConcurrentLinkedDeque<Integer> concurrentLinkedDeque21 = new ConcurrentLinkedDeque<>();

concurrentLinkedDeque21.push(10);

concurrentLinkedDeque21.push(11);

concurrentLinkedDeque21.push(12);

Integer removeLastResult = concurrentLinkedDeque21.removeLast();

System.out.println("deque的最后一个元素: " + removeLastResult);

/\*\*

\* 24、size()：返回此deque中的元素数。

\*/

Integer size = concurrentLinkedDeque21.size();

System.out.println("concurrentLinkedDeque21的元素个数为：" + size);

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

四、总结

①、基于链接节点的无界并发deque 。 并发插入，删除和访问操作可以跨多个线程安全执行。 一个 ConcurrentLinkedDeque是许多线程将共享对公共集合的访问的适当选择。 像大多数其他并发集合实现一样，此类不允许使用null元素， ConcurrentLinkedDeque是一个双向链表 。

②、ConcurrentLinkedDeque使用了自旋+CAS的非阻塞算法来保证线程并发访问时的数据一致性。由于队列本身是一种双链表结构，所以虽然算法看起来很简单，但其实需要考虑各种并发的情况，实现复杂度较高，并且ConcurrentLinkedDeque不具备实时的数据一致性，实际运用中，如果需要一种线程安全的栈结构，可以使用ConcurrentLinkedDeque。

③、关于ConcurrentLinkedDeque还有以下需要注意的几点：

1、ConcurrentLinkedDeque的迭代器是弱一致性的，这在并发容器中是比较普遍的现象，主要是指在一个线程在遍历队列结点而另一个线程尝试对某个队列结点进行修改的话不会抛出ConcurrentModificationException，这也就造成在遍历某个尚未被修改的结点时，在next方法返回时可以看到该结点的修改，但在遍历后再对该结点修改时就看不到这种变化。

2、size方法需要遍历链表，所以在并发情况下，其结果不一定是准确的，只能供参考。

————————————————

版权声明：本文为CSDN博主「住手丶让我来」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_42146366/java/article/details/88012792