写在前面

Java8，最耀眼的明星是Lambda表达式，Lambda表达式和Stream API彻底改变了集合的使用方式。
但是，还有一些很少被人发现的API改动，这里面有很多新的不起眼的改动，对我们的日常的编码有非常有用。

开胃小菜

那些年，我们错过的java7 特性

- try - with - resources

sample pom

Comment： try代码块异常发生时，会先关闭资源，然后调用cache分支。如果有finally分支，也会在关闭资源之后执行

- 文件处理 你是否还在使用万年的File类和我们一直依赖的Apache的文件工具类?

1. Path接口和Paths工具类，可以代表目录名称，也可以代表一个文件名

sample pom

```
Path directory = Paths.get("/", "home", "www");
Path file = Paths.get("/Users/line/nginx.conf");
```

2.Files类，读取和写入文件

sample pom

```
Path directory = Paths.get("/", "Users", "bizhenchao", "test");
Path file = directory.resolve("sample.conf");

//create dieectory
Files.createDirectories(directory);

//create file
Files.createFile(file);

// write date
String content = "hello,world!";
Files.write(file,content.getBytes(StandardCharsets.UTF_8));

// read date
List<String> dataLines = Files.readAllLines(file);
Stream<String> dataStream = Files.lines(file);

// copy/move/delete file
Files.copy(file, out)l
```

```
Files.move(source, target, options);
Files.deleteIfExists(file);

//searching for files in a file tree
Files.walkFileTree(start, visitor)
Files.walk(start, options)
```

3. Comment: 对于size比较大的文本文件，建议使用java.util.Scanner和Apache Commons IO的LineIterator

sample pom

```
java.util.Scanner ：

FileInputStream inputStream = new FileInputStream(path);;
Scanner sc = new Scanner(inputStream, "UTF-8");
while (sc.hasNextLine()) {
 String line = sc.nextLine();
 System.out.println(line);
}


LineIterator ：
LineIterator it = FileUtils.lineIterator(theFile, "UTF-8");
try {
 while (it.hasNext()) {
 String line = it.nextLine();
 // do something with line
 }
} finally {
 LineIterator.closeQuietly(it);
}
```

4. 再延伸一下，看看Spring文件读取。

sample pom

```
1. 基本的Resource
Resource resource = new FileSystemResource(file);
Resource resource = new ServletContextResource(ServletContext servletCo
Resource resource = new ClassPathResource("cn/javass/spring/chapter4/te
Resource resource = new UrlResource(“http://~ ~”);

(InputStreamResource,ByteArrayResource,EncodedResource)

2. 支持通配符的文件读取
ResourcePatternResolver resolver = new PathMatchingResourcePatternResol
Resource[] resources=resolver.getResources("classpath*:application-*.tx

3. 使用ResourceLoader 处理文件读取
ResourceLoader loader = new DefaultResourceLoader();
```

```
Resource resource = loader.getResource("classpath:cn/javass/spring/chap
Resource resource2 = loader.getResource("file:cn/javass/spring/chapter4
Resource resource3 = loader.getResource("cn/javass/spring/chapter4/test

4. 使用ResourceUtils来加载File
File clsFile = ResourceUtils.getFile("classpath:conf/file1.txt");
File httpFile = ResourceUtils.getFile(httpFilePath);

5. PropertiesLoaderUtils
Spring提供的 PropertiesLoaderUtils允许直接通过基于类路径的文件地址加载属性资源
Properties props = PropertiesLoaderUtils.loadAllProperties("jdbc.proper
Properties url = PropertiesLoaderUtils.loadProperties(new EncodedResour
```

5. Apache Common IO

sample pom

```
FileUtils
FilenameUtils
FileSystemUtils
IOUtils
LineIterator
...
```

- ProcessBuilder和Process，带超时的waitFor method

很多年前的噩梦，4年前就在想，如果java能提供timeout设置该有多好。

气哭joda - time的新Date API

Java8中对Date API重新进行了设计，使用新的API（java.time.*），基本上可以抛弃Joda-time api。

Date、Calendar和DateFotmat

一个简单的sample。

sample pom

```
// Defult China
SimpleDateFormat formaterDefault = new SimpleDateFormat("yyyy-MM-dd HH:

// Japan
SimpleDateFormat formaterJP = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss
TimeZone timeZoneJP = TimeZone.getTimeZone("GMT+9:00");
formaterJP.setTimeZone(timeZoneJP);


Date date = new Date();

System.out.println(formaterDefault.format(date));
System.out.println(formaterJP.format(date));
```

```
14
15 ==========Result==========
16 2016-09-12 07:15:28
17 2016-09-12 08:15:28
```

内部构造:

sample pom

```
1  /**
2   * Allocates a <code>Date</code> object and initializes it so that
3   * it represents the time at which it was allocated, measured to the
4   * nearest millisecond.
5   *
6   * @see java.lang.System#currentTimeMillis()
7   */
8  public Date() {
9  this(System.currentTimeMillis());
10 }
11
12  ==========
13
14  /**
15  * Gets a calendar using the default time zone and locale. The
16  * <code>Calendar</code> returned is based on the current time
17  * in the default time zone with the default
18  * {@link Locale.Category#FORMAT FORMAT} locale.
19  *
20  * @return a Calendar.
21  */
22 public static Calendar getInstance()
23 {
24 return createCalendar(TimeZone.getDefault(), Locale.getDefault(Locale.
25 }
26
27  /**
28  * Gets a calendar using the specified time zone and default locale.
29  * The <code>Calendar</code> returned is based on the current time
30  * in the given time zone with the default
31  * {@link Locale.Category#FORMAT FORMAT} locale.
32  *
33  * @param zone the time zone to use
34  * @return a Calendar.
35  */
36
37 public static Calendar getInstance(TimeZone zone)
38 {
39 return createCalendar(zone, Locale.getDefault(Locale.Category.FORMAT))
40 }
41
42  /**
43  * Gets a calendar using the default time zone and specified locale.
44  * The <code>Calendar</code> returned is based on the current time
45  * in the default time zone with the given locale.
46  *
47  * @param aLocale the locale for the week data
```

```
47  * @param aLocale the locale for the week data
48  * @return a Calendar.
49  */
50 public static Calendar getInstance(Locale aLocale)
51 {
52 return createCalendar(TimeZone.getDefault(), aLocale);
53 }
54
55 /**
56  * Gets a calendar with the specified time zone and locale.
57  * The <code>Calendar</code> returned is based on the current time
58  * in the given time zone with the given locale.
59  *
60  * @param zone the time zone to use
61  * @param aLocale the locale for the week data
62  * @return a Calendar.
63  */
64 public static Calendar getInstance(TimeZone zone,
65 Locale aLocale)
66 {
67 return createCalendar(zone, aLocale);
68 }
69
70 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
71 /**
72  * Constructs a <code>SimpleDateFormat</code> using the given pattern a
73  * the default date format symbols for the given locale.
74  * <b>Note:</b> This constructor may not support all locales.
75  * For full coverage, use the factory methods in the {@link DateFormat}
76  * class.
77  *
78  * @param pattern the pattern describing the date and time format
79  * @param locale the locale whose date format symbols should be used
80  * @exception NullPointerException if the given pattern or locale is nu
81  * @exception IllegalArgumentException if the given pattern is invalid
82  */
83 public SimpleDateFormat(String pattern, Locale locale)
84 {
85 if (pattern == null || locale == null) {
86 throw new NullPointerException();
87 }
88
89 initializeCalendar(locale);
90 this.pattern = pattern;
91 this.formatData = DateFormatSymbols.getInstanceRef(locale);
92 this.locale = locale;
93 initialize(locale);
94 }
95
96 private void initializeCalendar(Locale loc) {
97 if (calendar == null) {
98 assert loc != null;
99 // The format object must be constructed using the symbols for this zo
100 // However, the calendar should use the current default TimeZone.
101 // If this is not contained in the locale zone strings, then the zone
102 // will be formatted using generic GMT+/-H:MM nomenclature.
103 calendar = Calendar.getInstance(TimeZone.getDefault(), loc);
104 }
```

```
 }
}
```

Instant 时刻。 An instantaneous point on the time-line.

机器时间。与旧Date API一致，但是提供了丰富的时间数学计算API。

sample pom

```
Instant instant = Instant.now();
long millSeconds = System.currentTimeMillis();
Date date = new Date();

System.out.println("Instant : " + instant.toEpochMilli());
System.out.println("System : " + millSeconds);
System.out.println("Date : " + date.getTime());

= = = = = = = = = = = = = = = = =
Result:
Instant : 1473575877715
System : 1473575877715
Date : 1473575877715
```

本地时间（不带时区） LocalDate LocalTime LocalDateTime

人类时间。

1. LocalDate (2016/01/01)

sample pom

```
//-----------------------------------------------------------------------
 /**
 * Obtains the current date from the system clock in the default time-z
 * <p>
 * This will query the {@link Clock#systemDefaultZone() system clock} i
 * time-zone to obtain the current date.
 * <p>
 * Using this method will prevent the ability to use an alternate clock
 * because the clock is hard-coded.
 *
 * @return the current date using the system clock and default time-zon
 */
 public static LocalDate now() {
 return now(Clock.systemDefaultZone());
 }

 /**
 * Obtains the current date from the system clock in the specified time
 * <p>
 * This will query the {@link Clock#system(ZoneId) system clock} to obt
 * Specifying the time-zone avoids dependence on the default time-zone.
 * <p>
 * Using this method will prevent the ability to use an alternate clock
```

```
24  * because the clock is hard-coded.
25  *
26  * @param zone the zone ID to use, not null
27  * @return the current date using the system clock, not null
28  */
29 public static LocalDate now(ZoneId zone) {
30 return now(Clock.system(zone));
31 }
32
33 /**
34  * Obtains the current date from the specified clock.
35  * <p>
36  * This will query the specified clock to obtain the current date - tod
37  * Using this method allows the use of an alternate clock for testing.
38  * The alternate clock may be introduced using {@link Clock dependency
39  *
40  * @param clock the clock to use, not null
41  * @return the current date, not null
42  */
43 public static LocalDate now(Clock clock) {
44 Objects.requireNonNull(clock, "clock");
45 // inline to avoid creating object and Instant checks
46 final Instant now = clock.instant();  // called once
47 ZoneOffset offset = clock.getZone().getRules().getOffset(now);
48 long epochSec = now.getEpochSecond() + offset.getTotalSeconds();  // c
49 long epochDay = Math.floorDiv(epochSec, SECONDS_PER_DAY);
50 return LocalDate.ofEpochDay(epochDay);
51 }
52
53 //-----------------------------------------------------------------------
54 /**
55  * Obtains an instance of {@code LocalDate} from a year, month and day.
56  * <p>
57  * This returns a {@code LocalDate} with the specified year, month and
58  * The day must be valid for the year and month, otherwise an exception
59  *
60  * @param year the year to represent, from MIN_YEAR to MAX_YEAR
61  * @param month the month-of-year to represent, not null
62  * @param dayOfMonth the day-of-month to represent, from 1 to 31
63  * @return the local date, not null
64  * @throws DateTimeException if the value of any field is out of range,
65  * or if the day-of-month is invalid for the month-year
66  */
67 public static LocalDate of(int year, Month month, int dayOfMonth) {
68 YEAR.checkValidValue(year);
69 Objects.requireNonNull(month, "month");
70 DAY_OF_MONTH.checkValidValue(dayOfMonth);
71 return create(year, month.getValue(), dayOfMonth);
72 }
73
74 /**
75  * Obtains an instance of {@code LocalDate} from a year, month and day.
76  * <p>
77  * This returns a {@code LocalDate} with the specified year, month and
78  * The day must be valid for the year and month, otherwise an exception
79  *
80  * @param year the year to represent, from MIN_YEAR to MAX_YEAR
```

```
81  * @param month the month-of-year to represent, from 1 (January) to 12
82  * @param dayOfMonth the day-of-month to represent, from 1 to 31
83  * @return the local date, not null
84  * @throws DateTimeException if the value of any field is out of range,
85  * or if the day-of-month is invalid for the month-year
86  */
87 public static LocalDate of(int year, int month, int dayOfMonth) {
88 YEAR.checkValidValue(year);
89 MONTH_OF_YEAR.checkValidValue(month);
90 DAY_OF_MONTH.checkValidValue(dayOfMonth);
91 return create(year, month, dayOfMonth);
92 }
```

2. LocalTime (23:59:59.999999999)

sample pom

```
1 /**
2  * Obtains the current time from the system clock in the default time-z
3  * <p>
4  * This will query the {@link Clock#systemDefaultZone() system clock} i
5  * time-zone to obtain the current time.
6  * <p>
7  * Using this method will prevent the ability to use an alternate clock
8  * because the clock is hard-coded.
9  *
10 * @return the current time using the system clock and default time-zon
11 */
12 public static LocalTime now() {
13 return now(Clock.systemDefaultZone());
14 }
15
16 /**
17 * Obtains the current time from the system clock in the specified time
18 * <p>
19 * This will query the {@link Clock#system(ZoneId) system clock} to obt
20 * Specifying the time-zone avoids dependence on the default time-zone.
21 * <p>
22 * Using this method will prevent the ability to use an alternate clock
23 * because the clock is hard-coded.
24 *
25 * @param zone the zone ID to use, not null
26 * @return the current time using the system clock, not null
27 */
28 public static LocalTime now(ZoneId zone) {
29 return now(Clock.system(zone));
30 }
31
32 /**
33 * Obtains the current time from the specified clock.
34 * <p>
35 * This will query the specified clock to obtain the current time.
36 * Using this method allows the use of an alternate clock for testing.
37 * The alternate clock may be introduced using {@link Clock dependency
38 *
```

```
39  * @param clock the clock to use, not null
40  * @return the current time, not null
41  */
42 public static LocalTime now(Clock clock) {
43 Objects.requireNonNull(clock, "clock");
44 // inline OffsetTime factory to avoid creating object and InstantProvi
45 final Instant now = clock.instant();  // called once
46 ZoneOffset offset = clock.getZone().getRules().getOffset(now);
47 long localSecond = now.getEpochSecond() + offset.getTotalSeconds();  /
48 int secsOfDay = (int) Math.floorMod(localSecond, SECONDS_PER_DAY);
49 return ofNanoOfDay(secsOfDay * NANOS_PER_SECOND + now.getNano());
50 }
51
52 //-----------------------------------------------------------------------
53 /**
54  * Obtains an instance of {@code LocalTime} from an hour and minute.
55  * <p>
56  * This returns a {@code LocalTime} with the specified hour and minute.
57  * The second and nanosecond fields will be set to zero.
58  *
59  * @param hour the hour-of-day to represent, from 0 to 23
60  * @param minute the minute-of-hour to represent, from 0 to 59
61  * @return the local time, not null
62  * @throws DateTimeException if the value of any field is out of range
63  */
64 public static LocalTime of(int hour, int minute) {
65 HOUR_OF_DAY.checkValidValue(hour);
66 if (minute == 0) {
67 return HOURS[hour];  // for performance
68 }
69 MINUTE_OF_HOUR.checkValidValue(minute);
70 return new LocalTime(hour, minute, 0, 0);
71 }
72
73 /**
74  * Obtains an instance of {@code LocalTime} from an hour, minute and se
75  * <p>
76  * This returns a {@code LocalTime} with the specified hour, minute and
77  * The nanosecond field will be set to zero.
78  *
79  * @param hour the hour-of-day to represent, from 0 to 23
80  * @param minute the minute-of-hour to represent, from 0 to 59
81  * @param second the second-of-minute to represent, from 0 to 59
82  * @return the local time, not null
83  * @throws DateTimeException if the value of any field is out of range
84  */
85 public static LocalTime of(int hour, int minute, int second) {
86 HOUR_OF_DAY.checkValidValue(hour);
87 if ((minute | second) == 0) {
88 return HOURS[hour];  // for performance
89 }
90 MINUTE_OF_HOUR.checkValidValue(minute);
91 SECOND_OF_MINUTE.checkValidValue(second);
92 return new LocalTime(hour, minute, second, 0);
93 }
94
95 /**
```

```
 96  * Obtains an instance of {@code LocalTime} from an hour, minute, secon
 97  * <p>
 98  * This returns a {@code LocalTime} with the specified hour, minute, se
 99  *
100  * @param hour the hour-of-day to represent, from 0 to 23
101  * @param minute the minute-of-hour to represent, from 0 to 59
102  * @param second the second-of-minute to represent, from 0 to 59
103  * @param nanoOfSecond the nano-of-second to represent, from 0 to 999,9
104  * @return the local time, not null
105  * @throws DateTimeException if the value of any field is out of range
106  */
107 public static LocalTime of(int hour, int minute, int second, int nanoO
108 HOUR_OF_DAY.checkValidValue(hour);
109 MINUTE_OF_HOUR.checkValidValue(minute);
110 SECOND_OF_MINUTE.checkValidValue(second);
111 NANO_OF_SECOND.checkValidValue(nanoOfSecond);
112 return create(hour, minute, second, nanoOfSecond);
113 }
```

3. LocalDateTime

sample pom

```
  1 /**
  2  * Obtains the current date-time from the system clock in the default t
  3  * <p>
  4  * This will query the {@link Clock#systemDefaultZone() system clock} i
  5  * time-zone to obtain the current date-time.
  6  * <p>
  7  * Using this method will prevent the ability to use an alternate clock

  8  * because the clock is hard-coded.
  9  *
 10  * @return the current date-time using the system clock and default tim
 11  */
 12 public static LocalDateTime now() {
 13 return now(Clock.systemDefaultZone());
 14 }
 15
 16 /**
 17  * Obtains the current date-time from the system clock in the specified
 18  * <p>
 19  * This will query the {@link Clock#system(ZoneId) system clock} to obt
 20  * Specifying the time-zone avoids dependence on the default time-zone.
 21  * <p>
 22  * Using this method will prevent the ability to use an alternate clock
 23  * because the clock is hard-coded.
 24  *
 25  * @param zone the zone ID to use, not null
 26  * @return the current date-time using the system clock, not null
 27  */
 28 public static LocalDateTime now(ZoneId zone) {
 29 return now(Clock.system(zone));
 30 }
 31
 32 /**
```

```
/**
* Obtains the current date-time from the specified clock.
* <p>
* This will query the specified clock to obtain the current date-time.
* Using this method allows the use of an alternate clock for testing.
* The alternate clock may be introduced using {@link Clock dependency
*
* @param clock the clock to use, not null
* @return the current date-time, not null
*/
public static LocalDateTime now(Clock clock) {
Objects.requireNonNull(clock, "clock");
final Instant now = clock.instant();  // called once
ZoneOffset offset = clock.getZone().getRules().getOffset(now);
return ofEpochSecond(now.getEpochSecond(), now.getNano(), offset);
}

//-----------------------------------------------------------------------
/**
* Obtains an instance of {@code LocalDateTime} from year, month,
* day, hour and minute, setting the second and nanosecond to zero.
* <p>
* This returns a {@code LocalDateTime} with the specified year, month,
* day-of-month, hour and minute.
* The day must be valid for the year and month, otherwise an exception
* The second and nanosecond fields will be set to zero.
*
* @param year the year to represent, from MIN_YEAR to MAX_YEAR
* @param month the month-of-year to represent, not null
* @param dayOfMonth the day-of-month to represent, from 1 to 31
* @param hour the hour-of-day to represent, from 0 to 23
* @param minute the minute-of-hour to represent, from 0 to 59
* @return the local date-time, not null
* @throws DateTimeException if the value of any field is out of range,
* or if the day-of-month is invalid for the month-year
*/
public static LocalDateTime of(int year, Month month, int dayOfMonth,
LocalDate date = LocalDate.of(year, month, dayOfMonth);
LocalTime time = LocalTime.of(hour, minute);
return new LocalDateTime(date, time);
}

/**
* Obtains an instance of {@code LocalDateTime} from year, month,
* day, hour, minute and second, setting the nanosecond to zero.
* <p>
* This returns a {@code LocalDateTime} with the specified year, month,
* day-of-month, hour, minute and second.
* The day must be valid for the year and month, otherwise an exception
* The nanosecond field will be set to zero.
*
* @param year the year to represent, from MIN_YEAR to MAX_YEAR
* @param month the month-of-year to represent, not null
* @param dayOfMonth the day-of-month to represent, from 1 to 31
* @param hour the hour-of-day to represent, from 0 to 23
* @param minute the minute-of-hour to represent, from 0 to 59
* @param second the second-of-minute to represent, from 0 to 59
* @return the local date-time, not null
```

```
     * @return the local date-time, not null
     * @throws DateTimeException if the value of any field is out of range,
     * or if the day-of-month is invalid for the month-year
     */
    public static LocalDateTime of(int year, Month month, int dayOfMonth,
    LocalDate date = LocalDate.of(year, month, dayOfMonth);
    LocalTime time = LocalTime.of(hour, minute, second);
    return new LocalDateTime(date, time);
    }

    /**
     * Obtains an instance of {@code LocalDateTime} from year, month,
     * day, hour, minute, second and nanosecond.
     * <p>
     * This returns a {@code LocalDateTime} with the specified year, month,
     * day-of-month, hour, minute, second and nanosecond.
     * The day must be valid for the year and month, otherwise an exception
     *
     * @param year the year to represent, from MIN_YEAR to MAX_YEAR
     * @param month the month-of-year to represent, not null
     * @param dayOfMonth the day-of-month to represent, from 1 to 31
     * @param hour the hour-of-day to represent, from 0 to 23
     * @param minute the minute-of-hour to represent, from 0 to 59
     * @param second the second-of-minute to represent, from 0 to 59
     * @param nanoOfSecond the nano-of-second to represent, from 0 to 999,9
     * @return the local date-time, not null
     * @throws DateTimeException if the value of any field is out of range,
     * or if the day-of-month is invalid for the month-year
     */
    public static LocalDateTime of(int year, Month month, int dayOfMonth,
{
    LocalDate date = LocalDate.of(year, month, dayOfMonth);
    LocalTime time = LocalTime.of(hour, minute, second, nanoOfSecond);
    return new LocalDateTime(date, time);
    }
```

带时区时间 ZonedDateTime

人类时间。

sample pom

```
//-----------------------------------------------------------------------
    /**
     * Obtains the current date-time from the system clock in the default t
     * <p>
     * This will query the {@link Clock#systemDefaultZone() system clock} i
     * time-zone to obtain the current date-time.
     * The zone and offset will be set based on the time-zone in the clock.
     * <p>
     * Using this method will prevent the ability to use an alternate clock
     * because the clock is hard-coded.
     *
     * @return the current date-time using the system clock, not null
     */
```

```
14 public static ZonedDateTime now() {
15 return now(Clock.systemDefaultZone());
16 }
17
18 /**
19 * Obtains the current date-time from the system clock in the specified
20 * <p>
21 * This will query the {@link Clock#system(ZoneId) system clock} to obt
22 * Specifying the time-zone avoids dependence on the default time-zone.
23 * The offset will be calculated from the specified time-zone.
24 * <p>
25 * Using this method will prevent the ability to use an alternate clock
26 * because the clock is hard-coded.
27 *
28 * @param zone the zone ID to use, not null
29 * @return the current date-time using the system clock, not null
30 */
31 public static ZonedDateTime now(ZoneId zone) {
32 return now(Clock.system(zone));
33 }
34
35 /**
36 * Obtains the current date-time from the specified clock.
37 * <p>
38 * This will query the specified clock to obtain the current date-time.
39 * The zone and offset will be set based on the time-zone in the clock.
40 * <p>
41 * Using this method allows the use of an alternate clock for testing.
42 * The alternate clock may be introduced using {@link Clock dependency
43 *
44 * @param clock the clock to use, not null
45 * @return the current date-time, not null
46 */
47 public static ZonedDateTime now(Clock clock) {
48 Objects.requireNonNull(clock, "clock");
49 final Instant now = clock.instant();  // called once
50 return ofInstant(now, clock.getZone());
51 }
52
53 //-----------------------------------------------------------------------
54 /**
55 * Obtains an instance of {@code ZonedDateTime} from a local date and t
56 * <p>
57 * This creates a zoned date-time matching the input local date and tim
58 * Time-zone rules, such as daylight savings, mean that not every local
59 * is valid for the specified zone, thus the local date-time may be adj
60 * <p>
61 * The local date time and first combined to form a local date-time.
62 * The local date-time is then resolved to a single instant on the time
63 * This is achieved by finding a valid offset from UTC/Greenwich for th
64 * date-time as defined by the {@link ZoneRules rules} of the zone ID.
65 *<p>
66 * In most cases, there is only one valid offset for a local date-time.
67 * In the case of an overlap, when clocks are set back, there are two v
68 * This method uses the earlier offset typically corresponding to "summ
69 * <p>
70 * In the case of a gap, when clocks jump forward, there is no valid of
```

```
71  * Instead, the local date-time is adjusted to be later by the length o
72  * For a typical one hour daylight savings change, the local date-time
73  * moved one hour later into the offset typically corresponding to "sum
74  *
75  * @param date the local date, not null
76  * @param time the local time, not null
77  * @param zone the time-zone, not null
78  * @return the offset date-time, not null
79  */
80 public static ZonedDateTime of(LocalDate date, LocalTime time, ZoneId
81 return of(LocalDateTime.of(date, time), zone);
82 }
83
84 /**
85  * Obtains an instance of {@code ZonedDateTime} from a local date-time.
86  * <p>
87  * This creates a zoned date-time matching the input local date-time as
88  * Time-zone rules, such as daylight savings, mean that not every local
89  * is valid for the specified zone, thus the local date-time may be adj
90  * <p>
91  * The local date-time is resolved to a single instant on the time-line
92  * This is achieved by finding a valid offset from UTC/Greenwich for th
93  * date-time as defined by the {@link ZoneRules rules} of the zone ID.
94  *<p>
95  * In most cases, there is only one valid offset for a local date-time.
96  * In the case of an overlap, when clocks are set back, there are two v
97  * This method uses the earlier offset typically corresponding to "summ
98  * <p>
99  * In the case of a gap, when clocks jump forward, there is no valid of
100 * Instead, the local date-time is adjusted to be later by the length o
101 * For a typical one hour daylight savings change, the local date-time
102 * moved one hour later into the offset typically corresponding to "sum
103 *
104 * @param localDateTime the local date-time, not null
105 * @param zone the time-zone, not null
106 * @return the zoned date-time, not null
107 */
108 public static ZonedDateTime of(LocalDateTime localDateTime, ZoneId zon
109 return ofLocal(localDateTime, zone, null);
110 }
111
112 /**
113  * Obtains an instance of {@code ZonedDateTime} from a year, month, day
114  * hour, minute, second, nanosecond and time-zone.
115  * <p>
116  * This creates a zoned date-time matching the local date-time of the s
117  * specified fields as closely as possible.
118  * Time-zone rules, such as daylight savings, mean that not every local
119  * is valid for the specified zone, thus the local date-time may be adj
120  * <p>
121  * The local date-time is resolved to a single instant on the time-line
122  * This is achieved by finding a valid offset from UTC/Greenwich for th
123  * date-time as defined by the {@link ZoneRules rules} of the zone ID.
124  *<p>
125  * In most cases, there is only one valid offset for a local date-time.
126  * In the case of an overlap, when clocks are set back, there are two v
127  * This method uses the earlier offset typically corresponding to "summ
```

```
128 * <p>
129 * In the case of a gap, when clocks jump forward, there is no valid of
130 * Instead, the local date-time is adjusted to be later by the length o
131 * For a typical one hour daylight savings change, the local date-time
132 * moved one hour later into the offset typically corresponding to "sum
133 * <p>
134 * This method exists primarily for writing test cases.
135 * Non test-code will typically use other methods to create an offset t
136 * {@code LocalDateTime} has five additional convenience variants of th
137 * equivalent factory method taking fewer arguments.
138 * They are not provided here to reduce the footprint of the API.
139 *
140 * @param year the year to represent, from MIN_YEAR to MAX_YEAR
141 * @param month the month-of-year to represent, from 1 (January) to 12
142 * @param dayOfMonth the day-of-month to represent, from 1 to 31
143 * @param hour the hour-of-day to represent, from 0 to 23
144 * @param minute the minute-of-hour to represent, from 0 to 59
145 * @param second the second-of-minute to represent, from 0 to 59
146 * @param nanoOfSecond the nano-of-second to represent, from 0 to 999,9
147 * @param zone the time-zone, not null
148 * @return the offset date-time, not null
149 * @throws DateTimeException if the value of any field is out of range,
150 * if the day-of-month is invalid for the month-year
151 */
152 public static ZonedDateTime of(
153 int year, int month, int dayOfMonth,
154 int hour, int minute, int second, int nanoOfSecond, ZoneId zone) {
155 LocalDateTime dt = LocalDateTime.of(year, month, dayOfMonth, hour, min
156 return ofLocal(dt, zone, null);
157 }
158
159 /**
160 * Obtains an instance of {@code ZonedDateTime} from a local date-time
161 * using the preferred offset if possible.
162 * <p>
163 * The local date-time is resolved to a single instant on the time-line
164 * This is achieved by finding a valid offset from UTC/Greenwich for th
165 * date-time as defined by the {@link ZoneRules rules} of the zone ID.
166 *<p>
167 * In most cases, there is only one valid offset for a local date-time.
168 * In the case of an overlap, where clocks are set back, there are two
169 * If the preferred offset is one of the valid offsets then it is used.
170 * Otherwise the earlier valid offset is used, typically corresponding
171 * <p>
172 * In the case of a gap, where clocks jump forward, there is no valid o
173 * Instead, the local date-time is adjusted to be later by the length o
174 * For a typical one hour daylight savings change, the local date-time
175 * moved one hour later into the offset typically corresponding to "sum
176 *
177 * @param localDateTime the local date-time, not null
178 * @param zone the time-zone, not null
179 * @param preferredOffset the zone offset, null if no preference
180 * @return the zoned date-time, not null
181 */
182 public static ZonedDateTime ofLocal(LocalDateTime localDateTime, ZoneI
183 Objects.requireNonNull(localDateTime, "localDateTime");
184 Objects.requireNonNull(zone, "zone");
```

```
185 if (zone instanceof ZoneOffset) {
186 return new ZonedDateTime(localDateTime, (ZoneOffset) zone, zone);
187 }
188 ZoneRules rules = zone.getRules();
189 List<ZoneOffset> validOffsets = rules.getValidOffsets(localDateTime);
190 ZoneOffset offset;
191 if (validOffsets.size() == 1) {
192 offset = validOffsets.get(0);
193 } else if (validOffsets.size() == 0) {
194 ZoneOffsetTransition trans = rules.getTransition(localDateTime);
195 localDateTime = localDateTime.plusSeconds(trans.getDuration().getSecon
196 offset = trans.getOffsetAfter();
197 } else {
198 if (preferredOffset != null && validOffsets.contains(preferredOffset))
199 offset = preferredOffset;
200 } else {
201 offset = Objects.requireNonNull(validOffsets.get(0), "offset");  // pr
202 }
203 }
204 return new ZonedDateTime(localDateTime, offset, zone);
205 }
```

ZoneId、ZoneOffset和Clock

时区和时差。

sample pom

```
1 public static final Map<String, String> SHORT_IDS;
2  static {
3  Map<String, String> map = new HashMap<>(64);
4  map.put("ACT", "Australia/Darwin");
5  map.put("AET", "Australia/Sydney");
6  map.put("AGT", "America/Argentina/Buenos_Aires");
7  map.put("ART", "Africa/Cairo");
8  map.put("AST", "America/Anchorage");
9  map.put("BET", "America/Sao_Paulo");
10 map.put("BST", "Asia/Dhaka");
11 map.put("CAT", "Africa/Harare");
12 map.put("CNT", "America/St_Johns");
13 map.put("CST", "America/Chicago");
14 map.put("CTT", "Asia/Shanghai");
15 map.put("EAT", "Africa/Addis_Ababa");
16 map.put("ECT", "Europe/Paris");
17 map.put("IET", "America/Indiana/Indianapolis");
18 map.put("IST", "Asia/Kolkata");
19 map.put("JST", "Asia/Tokyo");
20 map.put("MIT", "Pacific/Apia");
21 map.put("NET", "Asia/Yerevan");
22 map.put("NST", "Pacific/Auckland");
23 map.put("PLT", "Asia/Karachi");
24 map.put("PNT", "America/Phoenix");
25 map.put("PRT", "America/Puerto_Rico");
26 map.put("PST", "America/Los_Angeles");
27 map.put("SST", "Pacific/Guadalcanal");
28 map.put("VST", "Asia/Ho_Chi_Minh");
29 map.put("EST", "-05:00");
```

```
map.put("MST", "-07:00");
map.put("HST", "-10:00");
SHORT_IDS = Collections.unmodifiableMap(map);
}

/**
* Obtains an instance of {@code ZoneId} using its ID using a map
* of aliases to supplement the standard zone IDs.
* <p>
* Many users of time-zones use short abbreviations, such as PST for
* 'Pacific Standard Time' and PDT for 'Pacific Daylight Time'.
* These abbreviations are not unique, and so cannot be used as IDs.
* This method allows a map of string to time-zone to be setup and reus
* within an application.
*
* @param zoneId the time-zone ID, not null
* @param aliasMap a map of alias zone IDs (typically abbreviations) to
* @return the zone ID, not null
* @throws DateTimeException if the zone ID has an invalid format
* @throws ZoneRulesException if the zone ID is a region ID that cannot
*/
public static ZoneId of(String zoneId, Map<String, String> aliasMap) {
Objects.requireNonNull(zoneId, "zoneId");
Objects.requireNonNull(aliasMap, "aliasMap");
String id = aliasMap.get(zoneId);
id = (id != null ? id : zoneId);
return of(id);
}


/**
* Parses the ID, taking a flag to indicate whether {@code ZoneRulesExc
* should be thrown or not, used in deserialization.
*
* @param zoneId the time-zone ID, not null
* @param checkAvailable whether to check if the zone ID is available
* @return the zone ID, not null
* @throws DateTimeException if the ID format is invalid
* @throws ZoneRulesException if checking availability and the ID canno
*/
static ZoneId of(String zoneId, boolean checkAvailable) {
Objects.requireNonNull(zoneId, "zoneId");
if (zoneId.length() <= 1 || zoneId.startsWith("+") || zoneId.startsWit
return ZoneOffset.of(zoneId);
} else if (zoneId.startsWith("UTC") || zoneId.startsWith("GMT")) {
return ofWithPrefix(zoneId, 3, checkAvailable);
} else if (zoneId.startsWith("UT")) {
return ofWithPrefix(zoneId, 2, checkAvailable);
}
return ZoneRegion.ofId(zoneId, checkAvailable);
}

/**
* Obtains a clock that returns the current instant using the best avai
* system clock, converting to date and time using the UTC time-zone.
* <p>
* This clock, rather than {@link #systemDefaultZone()}, should be used
```

```
 87  * you need the current instant without the date or time.
 88  * <p>
 89  * This clock is based on the best available system clock.
 90  * This may use {@link System#currentTimeMillis()}, or a higher resolut
 91  * clock if one is available.
 92  * <p>
 93  * Conversion from instant to date or time uses the {@linkplain ZoneOff
 94  * <p>
 95  * The returned implementation is immutable, thread-safe and {@code Ser
 96  * It is equivalent to {@code system(ZoneOffset.UTC)}.
 97  *
 98  * @return a clock that uses the best available system clock in the UTC
 99  */
100 public static Clock systemUTC() {
101 return new SystemClock(ZoneOffset.UTC);
102 }
103
104 /**
105  * Obtains a clock that returns the current instant using the best avai
106  * system clock, converting to date and time using the default time-zon
107  * <p>
108  * This clock is based on the best available system clock.
109  * This may use {@link System#currentTimeMillis()}, or a higher resolut
110  * clock if one is available.
111  * <p>
112  * Using this method hard codes a dependency to the default time-zone i
113  * It is recommended to avoid this and use a specific time-zone wheneve
114  * The {@link #systemUTC() UTC clock} should be used when you need the
115  * without the date or time.
116  * <p>
117  * The returned implementation is immutable, thread-safe and {@code Ser
118  * It is equivalent to {@code system(ZoneId.systemDefault())}.
119  *
120  * @return a clock that uses the best available system clock in the def
121  * @see ZoneId#systemDefault()
122  */
123 public static Clock systemDefaultZone() {
124 return new SystemClock(ZoneId.systemDefault());
125 }
126
127 /**
128  * Obtains a clock that returns the current instant using best availabl
129  * system clock.
130  * <p>
131  * This clock is based on the best available system clock.
132  * This may use {@link System#currentTimeMillis()}, or a higher resolut
133  * clock if one is available.
134  * <p>
135  * Conversion from instant to date or time uses the specified time-zone
136  * <p>
137  * The returned implementation is immutable, thread-safe and {@code Ser
138  *
139  * @param zone the time-zone to use to convert the instant to date-time
140  * @return a clock that uses the best available system clock in the spe
141  */
142 public static Clock system(ZoneId zone) {
143 Objects.requireNonNull(zone, "zone");
```

```
144  return new SystemClock(zone);
145  }
```

格式化和解析 DateTimeFormatter

sample pom

```
1 DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss
2 DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss
```

并发增强

原子类

java5之后追加的java.util.concurrent.atomic包，提供了无锁可变变量的类。例如AtomicLong，提供了增加、减少、计算等原子操作，还有一个可以计算较复杂的更新操作，这就是出名的compareAndSet。

sample pom

```
1  int newValue = 0;
2  int oldValue = 0;
3  int x = 1000;
4  AtomicInteger largest = new AtomicInteger(100);
5
6
7  Java8 Before:
8  do {
9   oldValue = largest.get();
10  newValue = Math.max(oldValue, x);
11 } while (largest.compareAndSet(oldValue, newValue));
12
13
14
15 Java8: Lambda表达式
16 largest.updateAndGet(oldValuePar -> Math.max(oldValuePar, x));
17
18 Source:
19 /**
20  * Atomically sets the value to the given updated value
21  * if the current value {@code ==} the expected value.
22  *
23  * @param expect the expected value
24  * @param update the new value
25  * @return {@code true} if successful. False return indicates that
26  * the actual value was not equal to the expected value.
27  */
28  public final boolean compareAndSet(int expect, int update) {
29  return unsafe.compareAndSwapInt(this, valueOffset, expect, update);
30  }
31
```

```
32
33  /**
34   * Atomically updates the current value with the results of
35   * applying the given function, returning the updated value. The
36   * function should be side-effect-free, since it may be re-applied
37   * when attempted updates fail due to contention among threads.
38   *
39   * @param updateFunction a side-effect-free function
40   * @return the updated value
41   * @since 1.8
42   */
43  public final int updateAndGet(IntUnaryOperator updateFunction) {
44  int prev, next;
45  do {
46  prev = get();
47  next = updateFunction.applyAsInt(prev);
48  } while (!compareAndSet(prev, next));
49  return next;
50  }
```

内部构造:

sample pom

```
1  /**
2   * Atomically sets the value to the given updated value
3   * if the current value {@code ==} the expected value.
4   *
5   * @param expect the expected value
6   * @param update the new value
7   * @return {@code true} if successful. False return indicates that
8   * the actual value was not equal to the expected value.
9   */
10 public final boolean compareAndSet(int expect, int update) {
11  return unsafe.compareAndSwapInt(this, valueOffset, expect, update);
12 }
13
14 // import sun.misc.Unsafe;
15 // 他提供了大量的低级的内存操作和系统功能 （将操作交给CPU CAS指令处理，锁住CPU总线
16 // setup to use Unsafe.compareAndSwapInt for updates
17 private static final Unsafe unsafe = Unsafe.getUnsafe();
18
19
20 CAS是一种系统原语。
21 所谓原语属于操作系统用语范畴。原语由若干条指令组成的，用于完成一定功能的一个过程。
22 primitive or atomic action是由若干个机器指令构成的完成某种特定功能的一段程序，具
23 原语的执行必须是连续的，在执行过程中不允许被中断。
```

ABA问题:

sample pom

```
ABA问题？AtomicStampedReference：原子更新带有版本号的引用类型，用于解决使用CAS进
```

Comment：原子更新引用类型（AtomicReference）和原子更新引用类型（AtomicReferenceFieldUpdater）中的字段，在之后的并发编程中提及。

ConcurrentHashMap持续改进

你是否还记得被别人面试时，经常问的一个问题，HashTable 和HashMap的区别（其中有一项大家都熟悉的，谁是线程安全的？）
Java5之后，ConcurrentHashMap已经变成了并发编程的主力。Spring LocalCache，是依靠ConcurrentHashMap做成的。
如果我们只能使用一种并发编程的数据结构，那就使用ConcurrentHashMap吧！

Comment ： 本来这一章节我想说的是简单的几个API，比如 putIfAbsent(K key, V value)等原子操作。

put和remove内部实现（Java7版本）：

sample pom

```
======================基本结构======================
 /**
 * The segments, each of which is a specialized hash table.
 */
 final Segment<K,V>[] segments;


/**
 * ConcurrentHashMap list entry. Note that this is never exported
 * out as a user-visible Map.Entry.
 */
 static final class HashEntry<K,V> {
 final int hash;
 final K key;
 volatile V value;
 volatile HashEntry<K,V> next;

 HashEntry(int hash, K key, V value, HashEntry<K,V> next) {
 this.hash = hash;
 this.key = key;
 this.value = value;
 this.next = next;
 }

 /**
 * Sets next field with volatile write semantics. (See above
 * about use of putOrderedObject.)
 */

```

```
final void setNext(HashEntry<K,V> n) {
UNSAFE.putOrderedObject(this, nextOffset, n);
}

}

======================PUT操作=====================

@SuppressWarnings("unchecked")
public V put(K key, V value) {
Segment<K,V> s;

//此处要注意，如果存放Null对象会抛出异常
if (value == null)
throw new NullPointerException();
int hash = hash(key);
int j = (hash >>> segmentShift) & segmentMask;
if ((s = (Segment<K,V>)UNSAFE.getObject  // nonvolatile; recheck
(segments, (j << SSHIFT) + SBASE)) == null) // in ensureSegment
s = ensureSegment(j);
return s.put(key, hash, value, false);
}


@SuppressWarnings("unchecked")
private Segment<K,V> ensureSegment(int k) {
final Segment<K,V>[] ss = this.segments;
long u = (k << SSHIFT) + SBASE; // raw offset
Segment<K,V> seg;
if ((seg = (Segment<K,V>)UNSAFE.getObjectVolatile(ss, u)) == null) {
Segment<K,V> proto = ss[0]; // use segment 0 as prototype
int cap = proto.table.length;
float lf = proto.loadFactor;
int threshold = (int)(cap * lf);
HashEntry<K,V>[] tab = (HashEntry<K,V>[])new HashEntry[cap];
if ((seg = (Segment<K,V>)UNSAFE.getObjectVolatile(ss, u))
== null) { // recheck
Segment<K,V> s = new Segment<K,V>(lf, threshold, tab);
while ((seg = (Segment<K,V>)UNSAFE.getObjectVolatile(ss, u))
== null) {
if (UNSAFE.compareAndSwapObject(ss, u, null, seg = s))
break;
}
}
}
return seg;
}

final V put(K key, int hash, V value, boolean onlyIfAbsent) {
//*****************************
HashEntry<K,V> node = tryLock() ? null :
scanAndLockForPut(key, hash, value);
V oldValue;
try {
HashEntry<K,V>[] tab = table;
int index = (tab.length - 1) & hash;
HashEntry<K,V> first = entryAt(tab, index);
```

```
for (HashEntry<K,V> e = first;;) {
if (e != null) {
K k;
if ((k = e.key) == key ||
(e.hash == hash && key.equals(k))) {
oldValue = e.value;
if (!onlyIfAbsent) {
e.value = value;
++modCount;
}
break;
}
e = e.next;
}
else {
if (node != null)
node.setNext(first);
else
node = new HashEntry<K,V>(hash, key, value, first);
int c = count + 1;
if (c > threshold && tab.length < MAXIMUM_CAPACITY)
rehash(node);
else
setEntryAt(tab, index, node);
++modCount;
count = c;
oldValue = null;
break;
}
}
} finally {
unlock();
}
return oldValue;
}

======================REMOVE操作======================

/**
* Removes the key (and its corresponding value) from this map.
* This method does nothing if the key is not in the map.
*
* @param  key the key that needs to be removed
* @return the previous value associated with <tt>key</tt>, or
* <tt>null</tt> if there was no mapping for <tt>key</tt>
* @throws NullPointerException if the specified key is null
*/
public V remove(Object key) {
int hash = hash(key);
Segment<K,V> s = segmentForHash(hash);
return s == null ? null : s.remove(key, hash, null);
}

/**
* Get the segment for the given hash
*/
@SuppressWarnings("unchecked")
```

```
144 private Segment<K,V> segmentForHash(int h) {
145 long u = (((h >>> segmentShift) & segmentMask) << SSHIFT) + SBASE;
146 return (Segment<K,V>) UNSAFE.getObjectVolatile(segments, u);
147 }
148
149 /**
150 * Remove; match on key only if value null, else match both.
151 */
152 final V remove(Object key, int hash, Object value) {
153 if (!tryLock())
154 scanAndLock(key, hash);
155 V oldValue = null;
156 try {
157 HashEntry<K,V>[] tab = table;
158 int index = (tab.length - 1) & hash;
159 HashEntry<K,V> e = entryAt(tab, index);
160 HashEntry<K,V> pred = null;
161 while (e != null) {
162 K k;
163 HashEntry<K,V> next = e.next;
164 if ((k = e.key) == key ||
165 (e.hash == hash && key.equals(k))) {
166 V v = e.value;
167 if (value == null || value == v || value.equals(v)) {
168 if (pred == null)
169 setEntryAt(tab, index, next);
170 else
171 pred.setNext(next);
172 ++modCount;
173 --count;
174 oldValue = v;
175 }
176 break;
177 }
178 pred = e;
179 e = next;
180 }
181 } finally {
182 unlock();
183 }
184 return oldValue;
185 }
186
187
188 ==========全部segment加锁==========
189 /**
190 * Returns the number of key-value mappings in this map. If the
191 * map contains more than <tt>Integer.MAX_VALUE</tt> elements, returns
192 * <tt>Integer.MAX_VALUE</tt>.
193 *
194 * @return the number of key-value mappings in this map
195 */
196 public int size() {
197 // Try a few times to get accurate count. On failure due to
198 // continuous async changes in table, resort to locking.
199 final Segment<K,V>[] segments = this.segments;
200 int size;
```

```
boolean overflow; // true if size overflows 32 bits
long sum;  // sum of modCounts
long last = 0L;  // previous sum
int retries = -1; // first iteration isn't retry
try {
for (;;) {
if (retries++ == RETRIES_BEFORE_LOCK) {
for (int j = 0; j < segments.length; ++j)
ensureSegment(j).lock(); // force creation
}
sum = 0L;
size = 0;
overflow = false;
for (int j = 0; j < segments.length; ++j) {
Segment<K,V> seg = segmentAt(segments, j);
if (seg != null) {
sum += seg.modCount;
int c = seg.count;
if (c < 0 || (size += c) < 0)
overflow = true;
}
}
if (sum == last)
break;
last = sum;
}
} finally {
if (retries > RETRIES_BEFORE_LOCK) {
for (int j = 0; j < segments.length; ++j)
segmentAt(segments, j).unlock();
}
}
return overflow ? Integer.MAX_VALUE : size;
}
```

put和remove内部实现（Java8内部实现）：

1. 不采用segment而采用node，锁住node来实现减小锁粒度。
2. 设计了MOVED状态 当resize的中过程中 线程2还在put数据，线程2会帮助resize。
3. 使用3个CAS操作来确保node的一些操作的原子性，这种方式代替了锁。

JDK8中使用synchronized而不是ReentrantLock，java8对锁的优化可能没有什么质上的区别。

sample pom

```
public V put(K key, V value) {
 return putVal(key, value, false);
 }

 /** Implementation for put and putIfAbsent */
 final V putVal(K key, V value, boolean onlyIfAbsent) {
 //不允许 key或value为null
 if (key == null || value == null) throw new NullPointerException();
```

```
//计算hash值
int hash = spread(key.hashCode());
int binCount = 0;
//死循环 何时插入成功 何时跳出
for (Node<K,V>[] tab = table;;) {
Node<K,V> f; int n, i, fh;
//如果table为空的话，初始化table
if (tab == null || (n = tab.length) == 0)
tab = initTable();
//根据hash值计算出在table里面的位置
else if ((f = tabAt(tab, i = (n - 1) & hash)) == null) {
//如果这个位置没有值 ，通过CAS直接放进去，不需要加锁
if (casTabAt(tab, i, null,
new Node<K,V>(hash, key, value, null)))
break;  // no lock when adding to empty bin
}
//如果其他线程在做扩容，则暂时停止put操作，帮助其扩容
//这个方法被调用的时候，当前ConcurrentHashMap一定已经有了nextTable对象，
//首先拿到这个nextTable对象，调用transfer方法,当本线程进入扩容方法的时候会直接进
else if ((fh = f.hash) == MOVED)
tab = helpTransfer(tab, f);
else {
//发生hash碰撞
V oldVal = null;
//结点上锁 hash值相同组成的链表的头结点
synchronized (f) {
if (tabAt(tab, i) == f) {
//fh》0 说明这个节点是一个链表的节点 不是树的节点
if (fh >= 0) {
binCount = 1;
//在这里遍历链表所有的结点
for (Node<K,V> e = f;; ++binCount) {
K ek;
//如果hash值和key值相同 则修改对应结点的value值
if (e.hash == hash &&
((ek = e.key) == key ||
(ek != null && key.equals(ek)))) {
oldVal = e.val;
if (!onlyIfAbsent)
e.val = value;
break;
}
Node<K,V> pred = e;
//如果遍历到了最后一个结点，那么就证明新的节点需要插入 就把它插入在链表尾部
if ((e = e.next) == null) {
pred.next = new Node<K,V>(hash, key,
value, null);
break;
}
}
}
//如果这个节点是树节点，就按照树的方式插入值
else if (f instanceof TreeBin) {
Node<K,V> p;
```

```
66  binCount = 2;
67  if ((p = ((TreeBin<K,V>)f).putTreeVal(hash, key,
68  value)) != null) {
69  oldVal = p.val;
70  if (!onlyIfAbsent)
71  p.val = value;
72  }
73  }
74  }
75  }
76  if (binCount != 0) {
77  //如果链表长度已经达到临界值8 就需要把链表转换为树结构
78  if (binCount >= TREEIFY_THRESHOLD)
79  treeifyBin(tab, i);
80  if (oldVal != null)
81  return oldVal;
82  break;
83  }
84  }
85  }
86  //将当前ConcurrentHashMap的元素数量+1
87  addCount(1L, binCount);
88  return null;
89  }
90
91
92  //这个方法用于将过长的链表转换为TreeBin对象。但是他并不是直接转换，而是进行一次容量
93  //直接进行扩容操作并返回；如果满足条件才链表的结构抓换为TreeBin ，这与HashMap不同
94  //它并没有把TreeNode直接放入红黑树，而是利用了TreeBin这个小容器来封装所有的TreeN
95  /**
96  * Replaces all linked nodes in bin at given index unless table is too
97  * small, in which case resizes instead.
98  */
99  private final void treeifyBin(Node<K, V>[] tab, int index) {
100 Node<K, V> b;
101 int n, sc;
102 if (tab != null) {
103 if ((n = tab.length) < MIN_TREEIFY_CAPACITY)
104 tryPresize(n << 1);
105 else if ((b = tabAt(tab, index)) != null && b.hash >= 0) {
106 synchronized (b) {
107 if (tabAt(tab, index) == b) {
108 TreeNode<K, V> hd = null, tl = null;
109 for (Node<K, V> e = b; e != null; e = e.next) {
110 //结点封装成TreeNode
111 TreeNode<K, V> p = new TreeNode<K, V>(e.hash, e.key, e.val, null, null
112 if ((p.prev = tl) == null)
113 hd = p;
114 else
115 tl.next = p;
116 tl = p;
117 }
118 // 放在TreeBin对象中，由TreeBin完成对红黑树的封装
119 setTabAt(tab, index, new TreeBin<K, V>(hd));
120 }
121 }
122 }
```

```
123  }
124  }
125
126
127
128 /**
129  * Removes the key (and its corresponding value) from this map. This me
130  * does nothing if the key is not in the map.
131  *
132  * @param key
133  * the key that needs to be removed
134  * @return the previous value associated with {@code key}, or {@code nu
135  * if there was no mapping for {@code key}
136  * @throws NullPointerException
137  * if the specified key is null
138  */
139  public V remove(Object key) {
140  return replaceNode(key, null, null);
141  }
142
143  /**
144  * Implementation for the four public remove/replace methods: Replaces
145  * value with v, conditional upon match of cv if non-null. If resulting
146  * value is null, delete.
147  */
148  final V replaceNode(Object key, V value, Object cv) {
149  int hash = spread(key.hashCode());
150  for (Node<K, V>[] tab = table;;) {
151  Node<K, V> f;
152  int n, i, fh;
153  if (tab == null || (n = tab.length) == 0 || (f = tabAt(tab, i = (n - 1
154  break;
155  else if ((fh = f.hash) == MOVED)
156  tab = helpTransfer(tab, f);
157  else {
158  V oldVal = null;
159  boolean validated = false;
160  synchronized (f) {
161  if (tabAt(tab, i) == f) {
162  if (fh >= 0) {
163  validated = true;
164  for (Node<K, V> e = f, pred = null;;) {
165  K ek;
166  if (e.hash == hash && ((ek = e.key) == key || (ek != null && key.equal
167  V ev = e.val;
168  if (cv == null || cv == ev || (ev != null && cv.equals(ev))) {
169  oldVal = ev;
170  if (value != null)
171  e.val = value;
172  else if (pred != null) //不是头节点
173  pred.next = e.next;
174  else  // e为头节点，将e的下一个节点变成头节点
175  setTabAt(tab, i, e.next);
176  }
177  break;
178  }
179  pred = e;
```

```
180 if ((e = e.next) == null)
181 break;
182 }
183 } else if (f instanceof TreeBin) {// 以树的方式find、remove
184 validated = true;
185 TreeBin<K, V> t = (TreeBin<K, V>) f;
186 TreeNode<K, V> r, p;
187 if ((r = t.root) != null && (p = r.findTreeNode(hash, key, null)) != n
188 V pv = p.val;
189 if (cv == null || cv == pv || (pv != null && cv.equals(pv))) {
190 oldVal = pv;
191 if (value != null)  // 修改更新
192 p.val = value;
193 else if (t.removeTreeNode(p))  //删除
194 setTabAt(tab, i, untreeify(t.first));
195 }
196 }
197 }
198 }
199 }
200 if (validated) {
201 if (oldVal != null) {
202 if (value == null)
203 addCount(-1L, -1);
204 return oldVal;
205 }
206 break;
207 }
208 }
209 }
210 return null;
211 }
212
213
214
215
216
217
218 //JDK8中的实现也是锁分离的思想，只是锁住的是一个Node，而不是JDK7中的Segment，而
219
220 /* ---------------- Table element access -------------- */
221
222 /*
223 * Volatile access methods are used for table elements as well as eleme
224 * of in-progress next table while resizing. All uses of the tab argume
225 * must be null checked by callers. All callers also paranoically prech
226 * that tab's length is not zero (or an equivalent check), thus ensurin
227 * that any index argument taking the form of a hash value anded with
228 * (length - 1) is a valid index. Note that, to be correct wrt arbitrar
229 * concurrency errors by users, these checks must operate on local
230 * variables, which accounts for some odd-looking inline assignments be
231 * Note that calls to setTabAt always occur within locked regions, and
232 * principle require only release ordering, not full volatile semantics
233 * are currently coded as volatile writes to be conservative.
234 */
235
236 @SuppressWarnings("unchecked")
```

```
237 static final <K, V> Node<K, V> tabAt(Node<K, V>[] tab, int i) {
238 return (Node<K, V>) U.getObjectVolatile(tab, ((long) i << ASHIFT) + AB
239 }
240
241 static final <K, V> boolean casTabAt(Node<K, V>[] tab, int i, Node<K,
242 return U.compareAndSwapObject(tab, ((long) i << ASHIFT) + ABASE, c, v)
243 }

    static final <K, V> void setTabAt(Node<K, V>[] tab, int i, Node<K, V>
    U.putObjectVolatile(tab, ((long) i << ASHIFT) + ABASE, v);
    }
```

再次气哭Guava的CompletableFuture

Future的缺点

Spring和Guava自身的ListenableFuture<T>

java8自身的CompletableFuture

sample pom

```
1
2 Future :
3 V get()
4 throws InterruptedException,
5  ExecutionException
6 Waits if necessary for the computation to complete, and then retrieves
7 Returns:
8 the computed result
9 Throws:
10 CancellationException - if the computation was cancelled
11 ExecutionException - if the computation threw an exception
12 InterruptedException - if the current thread was interrupted while wait
13
14
15 Spring:
16 ListenableFuture<SendResult<String, String>> result = kafkaTemplate.sen
17 result.addCallback(successCallback(), failureCallback());
18
```