Jdk7&8新特性介绍

钟伟坚

Jdk7语法

1.二进制变量
2.Switch语句支持string类型
3.try-with-resource语句
4.Catch多个异常
5.更佳的整数串
6.简化泛型
7.在可变参数方法中传递非具体化参数,改进编译警告和错误
8.信息更丰富的回溯追踪

整数的二进制

```
// 所有整数 int， short，long，byte都可以用二进制表示
// An 8-bit 'byte' value:
byte aByte = (byte) 0b00100001;

// A 16-bit 'short' value:
short aShort = (short) 0b1010000101000101;

// Some 32-bit 'int' values:
int anInt1 = 0b10100001010001011010000101000101;
int anInt2 = 0b101;
int anInt3 = 0B101; // The B can be upper or lower case.

// A 64-bit 'long' value. Note the "L" suffix:
long aLong =
0b10100001010001011010000101000101101000010100010110100001010001010L;

// 二进制在数组等的使用
final int[] phases = { 0b00110001, 0b01100010, 0b11000100, 0b10001001,
0b00010011, 0b00100110, 0b01001100, 0b10011000 };
```

Switch语句支持string类型

```
public static String getTypeOfDayWithSwitchStatement(String dayOfWeekArg) {
    String typeOfDay;
    switch (dayOfWeekArg) {
       case "Monday":
          typeOfDay = "Start of work week";
          break;
       case "Tuesday":
       case "Wednesday":
       case "Thursday":
          typeOfDay = "Midweek";
          break;
       case "Friday":
          typeOfDay = "End of work week";
          break;
       case "Saturday":
       case "Sunday":
          typeOfDay = "Weekend";
          break;
       default:
          throw new IllegalArgumentException("Invalid day of the week: " +
dayOfWeekArg);
    }
    return typeOfDay;
}
```

Try-with-resource语句
1.实现java.lang.AutoCloseable接口的资源；
2.按照声明逆序关闭资源
3.Try块抛出的异常通过Throwable.getSuppressed获取

```
try (java.util.zip.ZipFile zf = new java.util.zip.ZipFile(zipFileName);
java.io.BufferedWriter writer = java.nio.file.Files
.newBufferedWriter(outputFilePath, charset)) {

    // Enumerate each entry

    for (java.util.Enumeration entries = zf.entries(); entries
    .hasMoreElements();) {

    // Get the entry name and write it to the output file

    String newLine = System.getProperty("line.separator");
    String zipEntryName = ((java.util.zip.ZipEntry) entries
    .nextElement()).getName() + newLine;
    writer.write(zipEntryName, 0, zipEntryName.length());
    }
}
```

Catch多个异常

1.Catch异常类型为final；
2.生成Bytecode 会比多个catch小；
3.Rethrow时保持异常类型

```
public static void main(String[] args) throws Exception {
try {
testthrows();
} catch (IOException | SQLException ex) {
throw ex;
}

}

public static void testthrows() throws IOException, SQLException {

}
```

数字类型的下划线表示

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
//float pi1 = 3_.1415F;      // Invalid; cannot put underscores adjacent to a decimal point
//float pi2 = 3._1415F;      // Invalid; cannot put underscores adjacent to a decimal point
//long socialSecurityNumber1= 999_99_9999_L;        // Invalid; cannot put underscores
prior to an L suffix
//int x1 = _52;             // This is an identifier, not a numeric literal
int x2 = 5_2;             // OK (decimal literal)
//int x3 = 52_;             // Invalid; cannot put underscores at the end of a literal
int x4 = 5_______2;       // OK (decimal literal)
//int x5 = 0_x52;           // Invalid; cannot put underscores in the 0x radix prefix
//int x6 = 0x_52;           // Invalid; cannot put underscores at the beginning of a number
int x7 = 0x5_2;           // OK (hexadecimal literal)
//int x8 = 0x52_;           // Invalid; cannot put underscores at the end of a number
int x9 = 0_52;            // OK (octal literal)
int x10 = 05_2;           // OK (octal literal)
//int x11 = 052_;           // Invalid; cannot put underscores at the end of a number
```

泛型实例的创建可以通过类型推断来简化

```
//使用泛型前
List strList = new ArrayList();
List<String> strList4 = new ArrayList<String>();
List<Map<String, List<String>>> strList5 =  new ArrayList<Map<String, List<String>>>();


//编译器使用尖括号 (<>) 推断类型
List<String> strList0 = new ArrayList<String>();
List<Map<String, List<String>>> strList1 =  new ArrayList<Map<String, List<String>>>();
List<String> strList2 = new ArrayList<>();
List<Map<String, List<String>>> strList3 = new ArrayList<>();


List<String> list = new ArrayList<>();
list.add("A");
  // The following statement should fail since addAll expects
  // Collection<? extends String>
//list.addAll(new ArrayList<>());
```

在可变参数方法中传递非具体化参数,改进编译警告和错误

Heap pollution 指一个变量被指向另外一个不是相同类型的变量。例如

```
List l = new ArrayList<Number>();
List<String> ls = l;       // unchecked warning
l.add(0, new Integer(42)); // another unchecked warning
String s = ls.get(0);      // ClassCastException is thrown
```

Jdk7:

```
public static <T> void addToList (List<T> listArg, T... elements) {
for (T x : elements) {
listArg.add(x);
}
}
```

你会得到一个warning

warning: [varargs] Possible heap pollution from parameterized vararg type

要消除警告，可以有三种方式

1.加 annotation @SafeVarargs

2.加 annotation @SuppressWarnings({"unchecked", "varargs"})

3.使用编译器参数 –Xlint:varargs;

信息更丰富的回溯追踪

java.io.IOException
- at Suppress.write(Suppress.java:19)
- at Suppress.main(Suppress.java:8)
- Suppressed: java.io.IOException
- at Suppress.close(Suppress.java:24)
- at Suppress.main(Suppress.java:9)
- Suppressed: java.io.IOException
- at Suppress.close(Suppress.java:24)
- at Suppress.main(Suppress.java:9)

IO and New IO

1.java.nio.file 和java.nio.file.attribute包 支持更详细属性，比如权限，所有者
2. symbolic and hard links支持
3. Path访问文件系统，Files支持各种文件操作
4.高效的访问metadata信息
5.递归查找文件树，文件扩展搜索
6.文件系统修改通知机制
7.File类操作API兼容
8.文件随机访问增强 mapping a region,locl a region,绝对位置读取
9. AIO Reactor（基于事件）和Proactor

IO and New IO 监听文件系统变化通知

```
private WatchService watcher;
public TestWatcherService(Path path) throws IOException {
watcher = FileSystems.getDefault().newWatchService();
path.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
}
public void handleEvents() throws InterruptedException {
while (true) {
WatchKey key = watcher.take();
for (WatchEvent<?> event : key.pollEvents()) {
WatchEvent.Kind kind = event.kind();
if (kind == OVERFLOW) {// 事件可能lost or discarded
continue;
}
WatchEvent<Path> e = (WatchEvent<Path>) event;
Path fileName = e.context();
System.out.printf(
"Event %s has happened,which fileName is %s%n",
kind.name(), fileName);
}
if (!key.reset()) {
break;
}
```

IO and New IO遍历文件树

```
private void workFilePath() {
Path listDir = Paths.get("/tmp"); // define the starting file
ListTree walk = new ListTree();
…Files.walkFileTree(listDir, walk);…
// 遍历的时候跟踪链接
EnumSet opts = EnumSet.of(FileVisitOption.FOLLOW_LINKS);
try {
Files.walkFileTree(listDir, opts, Integer.MAX_VALUE, walk);
} catch (IOException e) {
System.err.println(e);
}
class ListTree extends SimpleFileVisitor<Path> {// NIO2 递归遍历文件目录的接口
@Override
public FileVisitResult postVisitDirectory(Path dir, IOException exc) {
System.out.println("Visited directory: " + dir.toString());
return FileVisitResult.CONTINUE;
}
@Override
public FileVisitResult visitFileFailed(Path file, IOException exc) {
System.out.println(exc);
return FileVisitResult.CONTINUE;
}
}
```

AIO异步IO 文件和网络

```
// 使用AsynchronousFileChannel.open(path, withOptions(),
   // taskExecutor))这个API对异步文件IO的处理
   public static void asyFileChannel2() {
      final int THREADS = 5;
      ExecutorService taskExecutor = Executors.newFixedThreadPool(THREADS);
      String encoding = System.getProperty("file.encoding");
      List<Future<ByteBuffer>> list = new ArrayList<>();
      int sheeps = 0;
      Path path = Paths.get("/tmp",
            "store.txt");
      try (AsynchronousFileChannel asynchronousFileChannel = AsynchronousFileChannel
            .open(path, withOptions(), taskExecutor)) {
         for (int i = 0; i < 50; i++) {
            Callable<ByteBuffer> worker = new Callable<ByteBuffer>() {
               @Override
               public ByteBuffer call() throws Exception {
                  ByteBuffer buffer = ByteBuffer
                        .allocateDirect(ThreadLocalRandom.current()
                              .nextInt(100, 200));
                  asynchronousFileChannel.read(buffer, ThreadLocalRandom
……
```

JDBC 4.1

1.可以使用try-with-resources自动关闭Connection, ResultSet, 和 Statement资源对象
2. RowSet 1.1：引入RowSetFactory接口和RowSetProvider类，可以创建JDBC driver支持的各种 row sets
3. **JDBC-ODBC**驱动会在jdk8中删除

```
try (Statement stmt = con.createStatement()) {
```

```
RowSetFactory aFactory = RowSetProvider.newFactory();
 CachedRowSet crs = aFactory.createCachedRowSet();

RowSetFactory rsf = RowSetProvider.newFactory("com.sun.rowset.RowSetFactoryImpl", null);
WebRowSet wrs = rsf.createWebRowSet();
```

createCachedRowSet
createFilteredRowSet
createJdbcRowSet
createJoinRowSet
createWebRowSet

并发工具增强

1.fork-join

2.ThreadLocalRandon

3. **phaser**

并发工具增强fork-join

```
class Fibonacci extends RecursiveTask<Integer> {
final int n;
Fibonacci(int n) {
this.n = n;
}
private int compute(int small) {
final int[] results = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };
return results[small];
}
public Integer compute() {
if (n <= 10) {
return compute(n);
}
Fibonacci f1 = new Fibonacci(n - 1);
Fibonacci f2 = new Fibonacci(n - 2);
System.out.println("fork new thread for " + (n - 1));
f1.fork();
System.out.println("fork new thread for " + (n - 2));
f2.fork();
return f1.join() + f2.join();
}
}
```

并发工具增强threadLocalRandon

```
final int MAX = 100000;
ThreadLocalRandom threadLocalRandom = ThreadLocalRandom.current();

long start = System.nanoTime();
for (int i = 0; i < MAX; i++) {
threadLocalRandom.nextDouble();
}
long end = System.nanoTime() - start;
System.out.println("use time1 : " + end);

long start2 = System.nanoTime();
for (int i = 0; i < MAX; i++) {
Math.random();
}
long end2 = System.nanoTime() - start2;
System.out.println("use time2 : " + end2);
```

并发工具增强

Phaser

```
void runTasks(List<Runnable> tasks) {
final Phaser phaser = new Phaser(1); // "1" to register self
// create and start threads
for (final Runnable task : tasks) {
phaser.register();
new Thread() {
public void run() {
phaser.arriveAndAwaitAdvance(); // await all creation
task.run();
}
}.start();
}

// allow threads to start and deregister self
phaser.arriveAndDeregister();
}
```

Networking增强

新增URLClassLoader close方法
URLClassLoader.newInstance(new URL[]{}).close();

新增Sockets Direct Protocol
绕过操作系统，网络传输从一台机器的内存数据直接传输到另外一台机器的内存中

Multithreaded Custom Class Loaders

Class Hierarchy:
class A extends B
class C extends D

ClassLoader Delegation Hierarchy:
Custom Classloader CL1:
directly loads class A
delegates to custom ClassLoader CL2 for class B

Custom Classloader CL2:
directly loads class C
delegates to custom ClassLoader CL1 for class D

Thread 1:
Use CL1 to load class A (locks CL1)
defineClass A triggers
loadClass B (try to lock CL2)

Thread 2:
Use CL2 to load class C (locks CL2)
defineClass C triggers
loadClass D (try to lock CL1)
Synchronization in the ClassLoader class wa

jdk7前：

jdk7

Thread 1:
Use CL1 to load class A (locks CL1+A)
defineClass A triggers
loadClass B (locks CL2+B)

Thread 2:
Use CL2 to load class C (locks CL2+C)
defineClass C triggers
loadClass D (locks CL1+D)

Security 增强

1.提供几种 ECC-based algorithms (ECDSA/ECDH) **Elliptic Curve Cryptography (ECC)**
2.禁用**CertPath Algorithm Disabling**
3. JSSE (SSL/TLS)的一些增强

Internationalization 增强

1. New Scripts and Characters from Unicode 6.0.0
2. Extensible Support for ISO 4217 Currency Codes

Currency类添加：

getAvailableCurrencies
getNumericCode
getDisplayName
getDisplayName(Locale)

3. Category Locale Support

getDefault(Locale.Category)FORMAT DISPLAY

4. Locale Class Supports BCP47 and UTR35

UNICODE_LOCALE_EXTENSION
PRIVATE_USE_EXTENSION
Locale.Builder
getExtensionKeys()
getExtension(char)
getUnicodeLocaleType(String
……

5. New NumericShaper Methods

NumericShaper.Range
getShaper(NumericShaper.Range)
getContextualShaper(Set<NumericShaper.Range>)……

Jvm

1.Jvm支持非java的语言 invokedynamic 指令

2. Garbage-First Collector 适合server端，多处理器下大内存，将heap分成大小相等的多个区域，mark阶段检测每个区域的存活对象，compress阶段将存活对象最小的先做回收，这样会腾出很多空闲区域，这样并发回收其他区域就能减少停止时间，提高吞吐量。

3. HotSpot性能增强

Tiered Compilation -XX:+UseTieredCompilation

Compressed Oops

Zero-Based Compressed Ordinary Object Pointers (oops)

4. Escape Analysis

5. NUMA Collector Enhancements

NUMA(Non Uniform Memory Access),NUMA在多种计算机系统中都得到实现,简而言之,就是将内存分段访问,类似于硬盘的RAID,Oracle中的分簇

Java 2D Enhancements

1. XRender-Based Rendering Pipeline - Dsun.java2d.xrender=True
2. Support for OpenType/CFF Fonts GraphicsEnvironment.getAvailableFontFamilyNames
3. TextLayout Support for Tibetan Script
4. Support for Linux Fonts

Swing Enhancements

1. JLayer
2. Nimbus Look & Feel
3. Heavyweight and Lightweight Components
4. Shaped and Translucent Windows
5. Hue-Saturation-Luminance (HSL) Color Selection in JColorChooser Class

Jdk8 lambda表达式

```
btn.setOnAction(new EventHandler<ActionEv
ent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

```
btn.setOnAction(
    event -> System.out.println("Hello World!")
);
```

```
public class Utils {
    public static int compareByLength(String in, Str
ing out){
        return in.length() - out.length();
    }
}

public class MyClass {
    public void doSomething() {
        String[] args = new String[] {"microsoft","ap
ple","linux","oracle"}
        Arrays.sort(args, Utils::compareByLength);
    }
}
```

Jdk8

用Metaspace代替PermGen
动态扩展，可以设置最大值，限制于本地内存的大小

Parallel array sorting 新APIArrays#parallelSort.

New Date & Time API

Clock clock = Clock.systemUTC(); //return the current time based on your system clock and set to UTC.

Clock clock = Clock.systemDefaultZone(); //return time based on system clock zone

long time = clock.millis(); //time in milliseconds from January 1st, 1970