

解

央乐崇拜234 阅读数：7970

原创文章，未经博主允许不得转载，转载请注明出处。博主博客地址是 <http://blog.csdn.net/liubenlong007> <https://blog.csdn.net/fgyibupi/article/details/62039628>

中也在使用。其最重要的特性就是Lambda表达式和函数式编程，这让我们的代码可以大大简化，更加优雅。

门java8，并且可以开始进行java8的程序编写了。讲的还算细致，但是限于篇幅原理讲的不是太深入，原理以后有时间再单独写博客。

u

body

明参数的类型。编译器可以从该参数的值推断。

在括号中声明参数。对于多个参数，括号是必需的。

体没有必要使用大括号，如果主体中含有一个单独的语句。

编译器会自动返回值，如果主体有一个表达式返回的值。花括号是必需的，以表明表达式返回一个值。

ong on 2016/11/7.

Test {

测试

```
oid main(String args[]){
    est tester = new LambdaTest();
```

declaration

```
ration addition = (int a, int b) -> a + b;
```

type declaration

```
ration subtraction = (a, b) -> a - b;
```

rn statement along with curly braces

```
ration multiplication = (int a, int b) -> { return a * b; };
```

eturn statement and without curly braces

```
ration division = (int a, int b) -> a / b;
```

```
out.println("10 + 5 = " + tester.operate(10, 5, addition));
out.println("10 - 5 = " + tester.operate(10, 5, subtraction));
out.println("10 x 5 = " + tester.operate(10, 5, multiplication));
out.println("10 / 5 = " + tester.operate(10, 5, division));
```

anthesis

```
gService greetService1 = message -> System.out.println("Hello " + message);
```

arentthesis

```
gService greetService2 = (message) -> System.out.println("Hello " + message);
```

```
rvic1.sayMessage("Mahesh");
rvic2.sayMessage("Suresh");
```

Python怎么学

转型AI人工智能指南

Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下

```
MathOperation {
    operation(int a, int b);

}

greetingService {
    Message(String message);

}

//rate(int a, int b, MathOperation mathOperation){
//    mathOperation.operation(a, b);
//}
```

👍
4

💬
2

📄

🔖

📱

<

>

也是重要的观点:

于定义内联执行的功能的接口，即只有一个单一的方法接口。在上面的例子中，我们使用不同类型的lambda表达式定义MathOperation接口的operation方法。：：：的sayMessage实现。

：：：名的需求，并给出了一个非常简单但功能强大的函数式编程能力。

：：：式的典型用法

```
b", "d" ).forEach( e -> System.out.println( e ) );

b", "d" ).sort( ( e1, e2 ) -> e1.compareTo( e2 ) );
```

：：：da表达式，JDK设计了函数式接口。

：：：一个方法的普通接口

：：：象为lambda表达式。 java.lang Runnable 与 java.util.concurrent.Callable 是函数式接口最典型的两个例子。在实际使用过程中，函数式接口是容易出错：：：了另一个方法，这时，这个接口就不再是函数式的了，并且编译过程也会失败。为了克服函数式接口的这种脆弱性并且能够明确声明接口作为函数式接口的意：：：主解 @FunctionalInterface （Java 8中所有类库的已有接口都添加了@FunctionalInterface注解）

```
@
able<V> {

    ult, or throws an exception if unable to do so.

    ied result
    tion if unable to compute a result

    ows Exception;
```

```
.Test;

.Arrays;
.List;
.function.BiFunction;
.function.Predicate;
```

ong on 2016/11/8.

```
nInterfaceTest {

    st1(){
teger> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

    dicate
    st, abc -> true);
    st, abc -> false);
    st, a -> a % 2 == 0);

    nction
    ion<Integer, Integer , Integer> biFunction = (a, b) -> {return a * b;};
    out.println(biFunction.apply(10, 2));

    oid eval(List<Integer> list, Predicate<Integer> predicate){
    rEach(integer -> {if(predicate.test(integer)) System.out.println(integer);});
}
```

lciate 都是函数式接口。读者可以自行运行，查看结果。

n

下都是函数式接口，我们可以直接拿来使用：

- 两个输入参数和不返回结果的操作。
- 一个参数，并产生一个结果的函数。
- 相同类型的两个操作数的操作，生产相同类型的操作数的结果。
- 数值谓词（布尔值函数）。
- 布尔值结果的提供者。
- 一个输入参数和不返回结果的操作。
- 代表在两个double值操作数的运算，并产生一个double值结果。
- 接受一个double值参数，不返回结果的操作。
- 接受double值参数，并产生一个结果的函数。
- 一个double值参数谓词（布尔值函数）。
- double值结果的提供者。
- 表示接受double值参数，并产生一个int值结果的函数。
- ：代表接受一个double值参数，并产生一个long值结果的函数。
- 表示上产生一个double值结果的单个double值操作数的操作。
- 参数，并产生一个结果的函数。
- ：对两个int值操作数的运算，并产生一个int值结果。
- 单个int值的参数并没有返回结果的操作。
- 一个int值参数，并产生一个结果的函数。
- ：数值参数谓词（布尔值函数）。
- 的提供者。
- 表示接受一个int值参数 并产生一个double值结果的函数



4



2



操作，并产生一个long值结果。

不返回结果的操作。

产生一个结果的函数。

（布尔值函数）。

表示接受double参数，并产生一个double值结果的函数。

示接受long值参数，并产生一个int值结果的函数。

示上产生一个long值结果单一的long值操作数的操作。

表示接受对象值和double值参数，并且没有返回结果的操作。

接受对象值和整型值参数，并返回没有结果的操作。

示接受对象的值和long值的说法，并没有返回结果的操作。

谓词（布尔值函数）。

旨的结果。

表示接受两个参数，并产生一个double值结果的功能。

表示一个产生一个double值结果的功能。

表示接受两个参数，并产生一个int值结果的函数。

表示一个int值结果的功能。

表示接受两个参数，并产生long值结果的功能。

表示一个产生long值结果的功能。

表示生相同类型的操作数的结果的单个操作数的操作。

默认方法和静态方法

的声明（为了节省篇幅，我删掉了注释）：

```
il.function;

.Objects;

@
e Predicate<T> {

    t(T t);

    cate<T> and(Predicate<? super T> other) {
        .requireNonNull(other);
        -> test(t) && other.test(t);

    cate<T> negate() {
        -> !test(t);

    cate<T> or(Predicate<? super T> other) {
        .requireNonNull(other);
        -> test(t) || other.test(t);

    Predicate<T> isEqual(Object targetRef) {
        ull == targetRef)
        ? Objects::isNull
        : object -> targetRef.equals(object);
```

3/16/2019

java8新特性详解 - 刘本龙的专栏 - CSDN博客

之处在于抽象方法必须要求实现，但是默认方法则没有这个要求。相反，每个接口都必须提供一个所谓的默认实现，这样人实现)

是非常高效的，并且通过字节码指令为方法调用提供了支持。默认方法允许继续使用现有的Java接口，而同时能够保障正常的编译过程。这方面好 4 子是大

tion接口中去：stream(), parallelStream(), forEach(), removeIf(),

青仔细思考是不是真的有必要使用默认方法，因为默认方法会带给程序歧义，并且在复杂的继承体系中容易产生编译错误。更多详情请参考 [官方文档](#)

一个函数式接口

变成了普通方法

```

    e
    o{

    o();

```

有多个

```
ayDefault(){  
out.println("我是default方法");
```

```
ayDefault11(){  
    out.println("我是default11方法");  
}
```

```

yStatic(){
    out.println("我是static方法");
}

```

```
01 extends Hello {
```

```
npl implements Hello1{  
  
yHello() {  
out.println("interface demo.HelloImpl.sayHello");  
}
```

```
st(){
    impl impl = new Hello1Impl();
    yHello();
    yDefault();
    ult();
    ul+11();
}
```

[Python怎么学](#)
[转型AI人工智能指南](#)
[Java学习路线](#)
[28 天算法训练营](#)
[2019 Python 开发者日](#)
[什么笔记本好用](#)
[mac系统下](#)

<https://blog.csdn.net/liubenlong007/article/details/62039628>

5/30

不可被继承哦
tic());

loImpl.sayHello

⋮

默认方法：

```
@
o2 extends Hello {

    sayDefault() {
        out.println("我是重写的Hello的default方法。");
    }
}

impl implements Hello2{

    yHello() {
        out.println("interface_demo.Hello2Impl.sayHello");
    }
}

st(){
    mpl impl = new Hello2Impl();
    yHello();

    ult11();
    ult();
}
```

lo2Impl.sayHello

⋮

efault方法。

有了默认方法：

o3 extends Hello {

普通抽象方法

ult();

👍
4

💬
2

📄

🔖

📱

<

>

```
impl implements Hello3{

    yHello() {
        out.println("interface_demo.Hello3Impl.sayHello");

        sayDefault已经不是一个默认方法

        yDefault() {
            System.out.println("interface_demo.Hello3Impl.sayDefault");

            st(){
                ult();
                o();
                ult11();

                hello3Impl.sayDefault
                hello3Impl.sayHello
            }
        }
    }
}
```

内语法，可以直接引用已有Java类或对象（实例）的方法或构造器。与lambda联合使用，方法引用可以使语言的构造更紧凑简洁，减少冗余代码。

注意构造器没有参数。比如 `HashSet::new`。下面详细实例中会有。

ethod。这个方法接受一个**Class**类型的参数

方法引用
这个方法没有参数

od。这个方法接受一个instance对应的Class类型的参数

```
.Test;

.*;
.function.Supplier;
```

ong on 2016/11/7.

```
ReferencesTest {

    nt(){
        out.println("sdfs");
    }
}
```

```
st1(){
ring> names = new ArrayList<String>();
dd("Mahesh");
dd("Suresh");
dd("Ramesh");
dd("Naresh");
dd("Kalpesh");
```

```
orEach(System.out::println);
```

```
st2(){
} array = {"gjyg", "ads", "jk"};
sort(array, String::compareToIgnoreCase);
```

```
ing s : array) {
tem.out.println(s);
```

```
risonProvider{
compareByName(Person a,Person b){
m a.getName().compareTo(b.getName());
```

```
compareByAge(Person a,Person b){
m a.getAge() - b.getAge();
```

```
st3(){
[] persons = initPerson();
son person : persons) person.printPerson();

out.println("*****以下是lambda表达式写法");
sort(persons, (a, b) -> a.getAge() - b.getAge());
son person : persons) person.printPerson();
```

```
out.println("*****以下是引用静态方法, 比lambda表达式写法简单");
sort(persons, Person::compareByAge);
son person : persons) person.printPerson();
```

```
out.println("*****以下是引用实例方法");
sonProvider provider = new ComparisonProvider();
sort(persons, provider::compareByAge);
son person : persons) person.printPerson();
```

```
out.println("*****使用lambda表达式-引用的是构造方法");
t<Person> personList = Arrays.asList(persons);
son> personSet = transferElements(personList,()-> new HashSet<>());
et.forEach(Person::printPerson);
```

```
out.println("*****使用方法引用-引用的是构造方法");
son> personSet2 = transferElements(personList, HashSet::new);
et2.forEach(Person::printPerson);
```

```
T, SOURCE extends Collection<T>, DEST extends Collection<T>> DEST transferElements(SOURCE sourceCollections, Supplier<DEST> colltionFacto
sult = colltionFactory.get();
olletions.forEach(o -> result.add(o));
```



```

n [] initPerson(){
[] persons = new Person[3];
person = new Person();
setName("张三");
setAge(10);
[0] = person;

= new Person();
setName("李四");
setAge(50);
[1] = person;

= new Person();
setName("王五");
setAge(2);
[2] = person;
persons;
}
```

了，读者自行运行该程序。

置只能声明一次，不能声明多次。在java8中，允许同一个位置声明多次注解。

`@Repeatable` 注解。事实上，这并不是语言层面的改变，更多的是编译器的技巧，底层的原理保持不变。让我们看一个快速入门的例子：

```

.annotation.ElementType;
.annotation.Repeatable;
.annotation.Retention;
.annotation.RetentionPolicy;
.annotation.Target;

ingAnnotations {
mentType.TYPE )
RetentionPolicy.RUNTIME )
ce Filters {
] value();

mentType.TYPE )
RetentionPolicy.RUNTIME )
Filters.class )
ce Filter {
value();

1" )
2" )
Filterable {

oid main(String[] args) {
ter filter: Filterable.class.getAnnotationsByType( Filter.class ) ) {
tem.out.println( filter.value() );
}
```

个使用@Repeatable(Filters.class)注解的注解类Filter，Filters仅仅是Filter注解的数组，但Java编译器并不想让程序员意 登录 的 注册 ，接口Filterable Filter) 注解。

新的函数getAnnotationsByType()来返回重复注解的类型（请注意Filterable.class.getAnnotation(Filters.class)经编译器处理后将会返回Filters的

决臭名昭著的 空指针异常 问题.

它可以保存类型T的值，或者仅仅保存null。Optional提供很多有用的方法，这样我们就不用显式进行空值检测

5:

一个描述指定值的Optional；否则返回空的Optional

一个描述指定值的Optional；否则报异常

TRUE，否则返回false。

前值；如果没有，则返回 other

前值；否则返回 NoSuchElementException

```
.Test;  
.Optional;
```

ong on 2016/11/8.

```
ilTest {  
  
st1(){  
lTest java8Tester = new OptionalTest();  
  
value1 = null;  
value2 = new Integer(10);  
!ofNullable - allows passed parameter to be null.  
l<Integer> a = Optional.ofNullable(value1);  
!of - throws NullPointerException if passed parameter is null  
l<Integer> b = Optional.of(value2);  
  
out.println(java8Tester.sum(a,b));  
  
r sum(Optional<Integer> a, Optional<Integer> b){  
!isPresent[判断值是否存在] - checks the value is present or not  
out.println("First parameter is present: " + a.isPresent());  
out.println("Second parameter is present: " + b.isPresent());  
  
!orElse - returns the value if present otherwise returns  
ult value passed.  
return a.isPresent() ? a : b;  
}
```

```
value2 = b.get();

value1 + value2;
```

👍
4

💬
2

📄

🔖

📱

<

>

```
is present: false
^ is present: true
```

新特性之一，真正的函数式编程风格引入到Java中，它大大简化了我们的编码量，让程序员写出高效率、干净、简洁的代码。

们这里只将常用的做一下说明

人对这个新特性会比较容易掌握，思想比较像。

集合作为其源。

并行数据流，集合作为其源。

于标准元素

个元素对应的结果

流中的每个元素

的大小

序

作，这是通常出现在管道传输操作结束标记流的结束。


回前三个：

```
st(){
ring> strings = Arrays.asList("abc", "", "bc", "efg", "abcd", "", "jkl");
ring> list = strings.stream().filter(n -> !"".equals(n)).limit(3).collect(Collectors.toList());
rEach(n -> System.out.println(n));
```


平方，并且去重，然后将值为81的去掉，输出排序后的数据

```
st2(){
teger> ints = Arrays.asList(1,5,9,6,5,4,2,5,9);
ream().map(n->n*n).distinct().filter(n->n!=81).sorted().collect(Collectors.toList()).forEach(n->System.out.println(n));
```


4

2











行流，求和。

```
st3(){
teger> ints = Arrays.asList(1,5,9,6,5,4,2,5,9);
out.println(ints.parallelStream().filter(n->n>2).distinct().count());
sum = ints.parallelStream().map(n -> n * n).filter(n -> n != 81).reduce(Integer::sum).get();
out.println(sum);
```

apToInt 将其转换为可以进行统计的数值型。类似的还有 mapToLong 、 mapToDouble

```
st5(){
teger> ints = Arrays.asList(1,5,9,6,5,4,2,5,9);
aryStatistics statistics = ints.stream().mapToInt(n -> n).summaryStatistics();
out.println(statistics.getAverage());
out.println(statistics.getCount());
out.println(statistics.getMax());
out.println(statistics.getMin());
```

11

间（Date/Time）

的Java日期/时间API

间API之前，让我们先来看一下为什么我们需要这样一个新的API。在Java中，现有的与日期和时间相关的类存在诸多问题，其中有：

定义并不一致，在java.util和java.sql的包中都有日期类，此外用于格式化和解析的类在java.text包中定义。

日期和时间，而java.sql.Date仅包含日期，将其纳入java.sql包并不合理。另外这两个类都有相同的名字，这本身就是一个非常糟糕的设计。

格式化以及解析，并没有一些明确定义的类。对于格式化和解析的需求，我们有java.text.DateFormat抽象类，但通常情况下，SimpleDateFormat类被用于此类需求，因此他们都不是线程安全的，这是Java日期类最大的问题之一。

时区，没有时区支持，因此Java引入了 java.util.Calendar和java.util.TimeZone类，但他们同样存在上述所有的问题。

此外，定义的方法还存在一些其他的问题，但以上问题已经很清晰地表明：Java需要一个健壮的日期/时间类。这也是为什么 Joda Time在Java日期/时间需求中扮演了

日期时间API中，所有的类都是不可变的，这对多线程环境有好处。

日期时间API将人可读的日期时间和机器时间（unix timestamp）明确分离，它为日期（Date）、时间（Time）、日期时间（DateTime）、时间戳（unix timestamp）以及日期时间（Date and Time）提供了不同的类。日期时间API的方法都被明确定义用以完成相同的行为。举个例子，要拿到当前实例我们可以使用now()方法，在所有的类中都定义了format()和parse()方法，而不再像以前那样需要不同的方法。有了更好的处理问题，所有的类都使用了工厂模式和策略模式，一旦你使用了其中某个类的方法，与其他类协同工作并不困难。日期/时间API类都实现了一系列方法用以完成通用的任务，如：加、减、格式化、解析、从日期/时间中提取单独部分，等等。日期时间API是工作在ISO-8601日历系统上的，但我们也可以将其应用在非IOS的日历上。

日期时间API包含以下相应的包。

- java.time包：新的Java日期/时间API的基础包，所有的主要基础类都是这个包的一部分，如：LocalDate, LocalTime, LocalDateTime, Instant, Period, Duration等。在绝大多数情况下，这些类能够有效地处理一些公共的需求。
- java.time.zone包：这个包为非ISO的日历系统定义了一些泛化的API，我们可以扩展AbstractChronology类来创建自己的日历系统。
- java.time.format包：这个包包含能够格式化和解析日期时间对象的类，在绝大多数情况下，我们不应该直接使用它们，因为java.time包中相应的类已经提供了格式化和解析的方法。
- java.time.temporal包：这个包包含一些时态对象，我们可以用其找出关于日期/时间对象的某个特定日期或时间，比如说，可以找到某月的第一天或最后一天。你可以非常容易地找到具有“withXXX”的格式。
- java.time.chronology包：这个包包含支持不同时区以及相关规则的类。

时区，然后就可以获取到当前的时刻，日期与时间。Clock可以替换 System.currentTimeMillis() 与 TimeZone.getDefault() 。

```
clock.systemUTC();
tln( clock.instant() );
tln( clock.millis() );
```

18.019Z

LocalDate和LocalTime的信息，但是不包含ISO-8601日历系统中的时区信息

```
stLocalDate(){
    current date and time
    teTime currentTime = LocalDateTime.now();
    out.println("Current DateTime: " + currentTime);

    te date1 = currentTime.toLocalDate();
    out.println("date1: " + date1);
    onth = currentTime.getMonth();
    = currentTime.getDayOfMonth();
    rds = currentTime.getSecond();
    out.println("Month: " + month
        + " day: " + day
        + " seconds: " + seconds
```

mber 2014

```
te date3 = LocalDate.of(2014, Month.DECEMBER, 12);
out.println("date3: " + date3);
```

15 minutes

```
me date4 = LocalTime.of(22, 15);
out.println("date4: " + date4);
```

string

```
me date5 = LocalTime.parse("20:15:30");
out.println("date5: " + date5);
```

2017-03-14T16:23:08.997

day: 14 seconds: 8
T16:23:08.997

处理-ZonedDateTime

```
stZonedDateTime(){
    current date and time
    teTime now = ZonedDateTime.now();
    out.println("now: " + now);
    teTime date1 = ZonedDateTime.parse("2007-12-03T10:15:30+05:30[Asia/Karachi]");
    out.println("date1: " + date1);
    id = ZoneId.of("Europe/Paris");
    out.println("Zoneld: " + id);
    currentZone = ZoneId.systemDefault();
    out.println("CurrentZone: " + currentZone);
```

09:33:10.108+08:00[Asia/Shanghai]
T10:15:30+05:00[Asia/Karachi]
ris
Shanghai

操作

```
stChronoUnits(){
    current date
    te today = LocalDate.now();
    out.println("Current date: " + today);
    week to the current date
    te nextWeek = today.plus(1, ChronoUnit.WEEKS);
    out.println("Next week: " + nextWeek);
    month to the current date
    te nextMonth = today.plus(1, ChronoUnit.MONTHS);
    out.println("Next month: " + nextMonth);
```

```
out.println("Next year: " + nextYear);
ears to the current date
te nextDecade = today.plus(1, ChronoUnit.DECADES);
ears to the current date
te nextDecade20 = today.plus(2, ChronoUnit.DECADES);
out.println("Date after 20 year: " + nextDecade20);
```

17-03-15
03-22
-04-15
03-15
ar: 2037-03-15



4



2



|入来处理时间差。

时间的日期数量。

于时间的时间量。

```
stPeriod(){
current date
te date1 = LocalDate.now();
out.println("Current date: " + date1);

onth to the current date
te date2 = date1.plus(3, ChronoUnit.DAYS);
out.println("Next month: " + date2);

period = Period.between(date1, date2);
out.println("Period: " + period);
out.println("Period.getYears: " + period.getYears());
out.println("Period.getMonths: " + period.getMonths());
out.println("Period.getDays: " + period.getDays());

stDuration(){
me time1 = LocalTime.now();
n twoHours = Duration.ofHours(2);

out.println("twoHours.getSeconds(): " + twoHours.getSeconds());

me time2 = time1.plus(twoHours);
out.println("time2: " + time2);

n duration = Duration.between(time1, time2);
out.println("Duration: " + duration);
out.println("Duration.getSeconds: " + duration.getSeconds());
```

看结果

er

明数学计算。例如，要获得“本月第二个星期六”或“下周二”。

```
te date1 = LocalDate.now();
out.println("Current date: " + date1);
```

```
next tuesday
te nextTuesday = date1.with(TemporalAdjusters.next(DayOfWeek.TUESDAY));
out.println("Next Tuesday on : " + nextTuesday);
```

```
二个周六
te firstInMonth = LocalDate.of(date1.getYear(),date1.getMonth(), 1);
te secondSaturday = firstInMonth.with(
    TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY)).with(
    TemporalAdjusters.next(DayOfWeek.SATURDAY));
out.println("Second saturday on : " + secondSaturday);
```

017-03-15
: 2017-03-21
on : 2017-03-11

oInstant方法

用于将它们转换到新的日期时间的API原始日期和日历对象。使用ofInstant(Insant,ZoneId)方法得到一个LocalDateTime或ZonedDateTime对象。

```
stBackwardCompatability(){
current date
rrentDate = new Date();
out.println("Current date: " + currentDate);

instant of current date in terms of milliseconds
now = currentDate.toInstant();
currentZone = ZoneId.systemDefault();

teTime localDateTime = LocalDateTime.ofInstant(now, currentZone);
out.println("Local date: " + localDateTime);

teTime zonedDateTime = ZonedDateTime.ofInstant(now, currentZone);
out.println("Zoned date: " + zonedDateTime);
```

Mar 15 09:51:17 CST 2017
J3-15T09:51:17.745
-03-15T09:51:17.745+08:00[Asia/Shanghai]

法可以将 Instant 转化为旧版本的Date对象。这两个方法都是jdk8以后新加的。

tter

代以前的dateformat，使用起来方便一些。

线程安全的，因为他是final的类

```
testDateTimeFormatter(){
teTime currentDate = LocalDateTime.now();
ut println("Current date: " + currentDate);
```



```
ut.println(DateTimeFormatter.ISO_LOCAL_DATE_TIME.format(currentDate));

ut.println(currentDate.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG)));
ut.println(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG).format(currentDate));

ut.println(currentDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
ut.println(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss").format(currentDate));
```

登录

注册

×

4

2

<

>

7-03-15T11:22:25.026

5.026

5.026

午11时22分25秒

午11时22分25秒

25

25

已经成为Java类库的标准。它的使用十分简单

了在Java整合。Java8现在有内置编码器和解码器的Base64编码。在Java8中，我们可以使用三种类型的Base64编码。

符在A-ZA-Z0-9+/.。编码器不添加任何换行输出和解码器拒绝在A-Za-z0-9+/.以外的任何字符。

字符在A-Za-z0-9+_.。输出URL和文件名安全。

ME友好的格式。输出表示在每次不超过76个字符行和使用'\r'后跟一个换行符'\n'回车作为行分隔符。无行隔板的存在是为了使编码的结束输出。

```
st(){

    [] bytes = "liubenlong?java8".getBytes();

    //code using basic encoder

    ing base64encodedString = Base64.getEncoder().encodeToString(bytes);
    tem.out.println("Base64 Encoded String (Basic) : " + base64encodedString);

    //code

    [] base64decodedBytes = Base64.getDecoder().decode(base64encodedString);
    tem.out.println("Original String : "+new String(base64decodedBytes, "utf-8"));

    e64encodedString = Base64.getUrlEncoder().encodeToString(bytes);
    tem.out.println("Base64 Encoded String (URL) : " + base64encodedString);

    ingBuilder stringBuilder = new StringBuilder();

    (int i = 0; i < 10; ++i) {
        stringBuilder.append(UUID.randomUUID().toString());
    }

    [] mimeBytes = stringBuilder.toString().getBytes("utf-8");
    ing mimeEncodedString = Base64.getMimeEncoder().encodeToString(mimeBytes);
    tem.out.println("Base64 Encoded String (MIME) : "+mimeEncodedString);
    nsupportedEncodingException e){
    tem.out.println("Error : "+e.getMessage());
}
```

```
String (Basic) : bGl1YmVubG9uZz9qYXZh0A==
```

$$1 \leq k \leq 1 \leq 2 \leq \dots \leq n$$

Python怎么学

转型AI人工智能指南

Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下

5N2ViZWZmMmYwMzgtMTewYS00NGE3LWFlnMmtNGQyMDJlOTA5MTM0OTMxMWRh
5LWI0MGYtMmI3ZWZmMzEzMGMxMzE5NTZhMjYtMDAyZi00ZmFhLWI2ZTEtNDY3
3N2Y5NjQtMWRiYy00ZTJlLWFhOGMtYzdkNzlmYTM3MGQzYmJlYTk0ZmUtZWFl
tNmQwNmExNjI3NzNmYmU5Yzc5MwItZjAyNy00NjI1LTliYmEtMjhmNWQyZDk3
tYzYxNC00YTFlLWI2NmUtZjc0MWE3MzM5ODJiODU0ZmFlMmQtNjFiZi00NWJl
3ZGJiNzNm

登录

注册

×

4

2

4

4

4

4

4

数组

去来对数组进行并行处理。可以说，最重要的是parallelSort()方法，因为它可以在多核机器上极大提高数组排序的速度

100的数组，并且填充1000000内的随机数，然后并行排序

```
Duration;  
LocalTime;  
rrays;  
ncurrent.ThreadLocalRandom;  
  
parallelArrays {  
ic void main( String[] args ) {  
me begin = LocalTime.now();  
arrayOfLong = new long [ 20000 ];  
  
parallelSetAll(arrayOfLong, index -> ThreadLocalRandom.current().nextInt(1000000));  
stream(arrayOfLong).limit(10).forEach(i -> System.out.print(i + " "));  
ut.println();  
  
parallelSort( arrayOfLong );  
stream(arrayOfLong).limit(10).forEach(i -> System.out.print(i + " "));  
ut.println();  
ut.println(Duration.between(begin, LocalTime.now()));  
}
```

36377 442743 107942 904354 253278 17119 131953 86974
3 269 297 405 420 469

rency)

虽：LongAddr

计数器，比atomicLong性能还要高。
a高并发：CAS无锁原理及广泛应用。
，简单来说就是解决了伪共享的问题。具体我会单独再写一篇文章讲解这个伪共享以及LongAddr的实现原理。

类，atomicLong，atomicLong三种原子计数性能进行一下比较：

```
;  
  
.concurrent.CountDownLatch;  
.concurrent.ExecutorService;  
.concurrent.Executors;  
.concurrent.atomic.AtomicInteger;
```

ong on 2017/1/22.

Python怎么学

转型AI人工智能指南

Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下

```
ldrTest {
xValue = 10000000;
ownLatch countDownLatch = new CountDownLatch(10);
long count = 0;

nized void incr(){
lue() < maxValue) count++;

tValue(){
unt;

oid main(String[] args) throws InterruptedException {

rTest longAddrTest = new LongAddrTest();

in = System.currentTimeMillis();
rService executorService = Executors.newFixedThreadPool(10);

= 0 ; i < 10 ; i ++){
cutorService.execute(() -> {
while(longAddrTest.getValue() < maxValue){
longAddrTest.incr();
}
countDownLatch.countDown();

wnLatch.await();
rService.shutdown();

out.println("总共耗时:"+(System.currentTimeMillis() - begin)+"毫秒, count="+longAddrTest.getValue());

;

, concurrent.CountDownLatch;
, concurrent.ExecutorService;
, concurrent.Executors;
, concurrent.atomic.AtomicLong;
```

ong on 2017/1/22.
micLong性能更高的原子类

omicLong, LongAddr 三种无锁

```
ldrTest1 {
oid main(String[] args) throws InterruptedException {
wnLatch countDownLatch = new CountDownLatch(10);
ong atomicLong = new AtomicLong(0);

Value = 10000000;
in = System.currentTimeMillis();
rService executorService = Executors.newFixedThreadPool(10);

= 0 ; i < 10 ; i ++){
cutorService.execute(() -> {
while(atomicLong.get() < maxValue){
```

```
wnLatch.await();
rService.shutdown();

out.println("总共耗时:"+(System.currentTimeMillis() - begin)+"毫秒, count="+atomicLong.get());

;

.concurrent.CountDownLatch;
.concurrent.ExecutorService;
.concurrent.Executors;
.concurrent.atomic.LongAdder;
```

ong on 2017/1/22.
micLong性能更高的原子类

omicLong, LongAddr 三种无锁

```
ldrTest2 {
    oid main(String[] args) throws InterruptedException {
        wnLatch countDownLatch = new CountDownLatch(10);
        er longAdder = new LongAdder();

        Value = 10000000;
        in = System.currentTimeMillis();
        rService executorService = Executors.newFixedThreadPool(10);

        = 0 ; i < 10 ; i ++){
            cutorService.execute() -> {
                while(longAdder.sum() < maxValu){
                    longAdder.increment();
                }
                countDownLatch.countDown();
            }
        }

        wnLatch.await();
        rService.shutdown();

        out.println("总共耗时:"+(System.currentTimeMillis() - begin)+"毫秒, count="+longAdder.sum());
    }
}
```

看出来，AtomicLong，LongAddr两个比加锁性能提高了很多，LongAddr又比AtomicLong性能高。所以我们以后在遇到线程安全的原子计数器的時候，首先考虑

的优化

过“ java.lang.OutOfMemoryError: PermGen space ”这一问题。这往往是由类加载器相关的内存泄漏以及新类加载器的创建导致的，通常出现于代码热部署以

余永久区，使用本地内存来存储类元数据信息并称之为：元空间（Metaspace）

n 持久区，在java8中去掉了这个区域，取而代之的是将原本要放在持久区的类元数据信息、字节码信息及 static final 的常量，转移到了计算机本地内存中。
n space 的异常，而且不需要关心元空间的大小，它只受限于就算几内存。

Python怎么学

转要AI人工智能指南

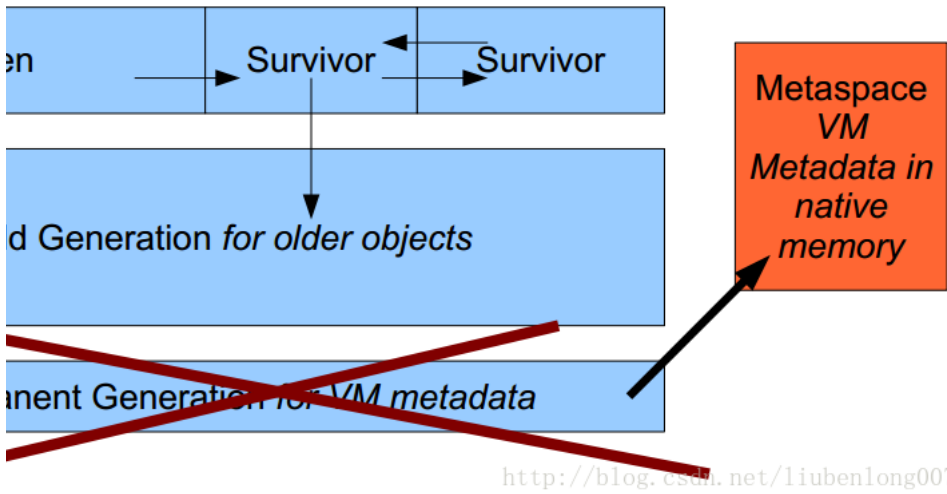
Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下



👍
4

💬
2

📄

🔖

📱

<

>

想一想，真的不需要关心这部分数据量的大小吗？显然不可以。
据过多，依旧会有类卸载的需求。

ce的改变

JVM的参数：PermSize 和 MaxPermSize 会被忽略并给出警告（如果在启用时设置了这两个参数）。

本地内存中分配。

ceSize）用于限制本地内存分配给类元数据的大小。如果没有指定这个参数，元空间会在运行时根据需要动态调整。

：对于僵死的类及类加载器的垃圾回收将在元数据使用达到“MaxMetaspaceSize”参数的设定值时进行。适时地监控和调整元空间对于减小垃圾回收频率和减少延
空间垃圾回收说明，可能存在类、类加载器导致的内存泄漏或是大小设置不合适。

以不关注这个元空间，因为JVM会在运行时自动调校为“合适的大小”；元空间提高Full GC的性能，在Full GC期间，Metadata到Metadata pointers之间不需要扫描了，别小看这几纳秒
存在内存泄露，像OOMTest那样，不停的扩展metaspace的空间，会导致机器的内存不足，所以还是要有必要的调试和监控。

uture

用来描述一个异步计算的结果。你可以使用isDone方法检查计算是否完成，或者使用get阻塞住调用线程，直到计算完成返回结果，你也可以使用cancel方法停

方法提供了异步执行任务的能力，但是对于结果的获取却是很不方便，只能通过阻塞或者轮询的方式得到任务的结果。阻塞的方式显然和我们的异步编程的初衷
与CPU资源，而且也不能及时地得到计算结果，为什么不能用观察者设计模式当计算结果完成及时通知监听者呢？

包含50个方法左右的类: CompletableFuture，提供了非常强大的Future的扩展功能，可以帮助我们简化异步编程的复杂性，提供了函数式编程的能力，可以通
提供了转换和组合CompletableFuture的方法。

用，可以用于入门。

后续使用：

```
er square(Integer a){  
    //模拟耗时任务  
    Thread.sleep(1000);  
    try {  
        //抛出异常  
        throw new InterruptedException();  
    } catch (InterruptedException e) {  
        //打印堆栈  
        e.printStackTrace();  
    }  
    return a * a;  
}
```

/ b;

```
re<Integer> completableFuture = CompletableFuture.supplyAsync(()->square(10));
```

任务已提交完成，等待结果。”);

```
ompletableFuture.get();
teger);
```

任务已提交完成，等待结果。”。等待几秒钟才输出结果 100 .

线程的执行结果。如果还没有执行完，则阻塞。
monPool()线程池中执行任务，该线程池中的线程都是daemon的，所以必须调用get()方法等待返回结果。

并且拼接上”!”：

```
re
Async(()->square(20))
a + ""
> str + "!!")
stem.out::println)
```

与0异常，通过 exceptionally 捕获异常进行处理。

```
re
Async(() -> division(10, 0))
e) -> {捕获异常，处理完成后返回了0
race();
```

```
stem.out::println)
```

```
ompletionException: java.lang.ArithmeticException: / by zero
ncurrent.CompletableFuture.encodeThrowable(CompletableFuture.java:273)
ncurrent.CompletableFuture.completeThrowable(CompletableFuture.java:280)
ncurrent.CompletableFuture$AsyncSupply.run(CompletableFuture.java:1592)
ncurrent.CompletableFuture$AsyncSupply.exec(CompletableFuture.java:1582)
ncurrent.ForkJoinTask.doExec(ForkJoinTask.java:289)
ncurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1056)
ncurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1692)
ncurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
.lang.ArithmeticException: / by zero
mpletableFutureTest.division(CompletableFutureTest.java:23)
mpletableFutureTest.lambda$main$4(CompletableFutureTest.java:49)
ncurrent.CompletableFuture$AsyncSupply.run(CompletableFuture.java:1590)
```

Future组合：thenCombine

Future组合使用 方式一

```
CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> square(10));
CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> division(200, returnVal));
return future1.thenCombine(future2, (i, j) -> i * j).get();
```

Future组合使用 方式二

```
CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() -> square(10));
CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() -> division(200, 10));
return future1.thenCombine(future2, (i, j) -> i * j).get();
```

1、跟OOM：Permgen说再见吧

与

ure 详解

cpu machdep.cpu.core_count: 2 machdep.cpu.thread_count: 4 (9个月前 #2楼)

数组那块，我写了个demo比较了一下，传统的方式和并行方式性能上，跑了几次，结果是不确定的

```
stParallel(){
    longs = new Long[20000];
    time start = LocalTime.now();

    for (int i = 0; i < longs.length; i++) {
        longs[i] = ThreadLocalRandom.current().nextLong(1000000);
    }

    .sort(longs);
    println(Duration.between(start, LocalTime.now()));

    longs = new Long[20000];

    time begin = LocalTime.now();
    .parallelSetAll(longs, index -> ThreadLocalRandom.current().nextLong(1000000));

    .parallelSort(longs);
    println(Duration.between(begin, LocalTime.now()));
}
```

阅读数 275

bda表达式，也可称为闭包，它是推动Java8发布的最重要新特性。Lamb... 博文 来自： 新新新的博客、

4

1bda 表达式与函数式接口

阅读数 252

mp;nbsp;&amp;nbsp;最近学习集合的时候了解到la... 博文 来自： 全力奔跑，梦...

除四则运算程序

儿童四则运算程序，能够根据用户的设定（题目数量和题目类型）自动出题， ... 论坛

Bruce Eckel 以及Thinking in Java 5th edition 《On Java 8》

阅读数 5722

搜了一下BruceEckel的信息，发现了这篇文章和Java编程思想(4thedition... 博文 来自： ananhao的博客

dder的区别

阅读数 1万+

里面使用到了LongAdder这个类，而并非AtomicLong，很是困惑，于是专... 博文 来自： yao123long的...

阅读数 5568

mportjava.util.function.Supplier;/**函数式接口Supplier*/publicclassTest1... 博文 来自： lgx06的专栏

i特性

阅读数 1539

017/07/21/keep-up-with-java8-features.html如有好文章投稿，请点击→这... 博文 来自： lz710117239的...

Supplier

阅读数 2739

些需要延迟计算的情形，比如某些运算非常消耗资源，如果提前算出来却... 博文 来自： 十点半睡觉

阅读数 94

一次重大的版本升级，有一些新的特性值得一记。特性一：Lambda表达... 博文 来自： 杜



wang123459

147篇文章

排名:千里之外

关注



火腿编程

196篇文章

排名:2000+

关注



可乐代码君

18篇文章

排名:千里之外

关注

阅读数 522

一大把，但是网上说的玄乎但是我们也要自己去实践到底有多么的神乎... 博文 来自： Coder的不平凡

新特性java8新特性java8新特性

下载

！

阅读数 2万+

新增了default方法和static方法，这两种方法可以有方法体1、static方法示... 博文 来自： cdw8131197的...

阅读数 2万+

JDK1.8版本中的一些新特性，乃作者视频观后笔记，仅供参考。jdk1.8新... 博文 来自： 随波的博客

阅读数 1740

列出，并将使用简单的代码示例来指导你如何使用默认接口方法，lambda... 博文 来自： world_snow的...

7

阅读数 6万+

cle/48304.htm本教程将Java8的新特新逐一列出，并将使用简单的代码示... 博文 来自： Haiyoung

阅读数 78

va语言开发的一个主要版本。Oracle公司于2014年3月18日发布Java8， ... 博文 来自： 李章勇的博客

Java8新特性)

04-11

Java8新特性)

Python怎么学

转型AI人工智能指南

Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下

表达式中是无法访问到默认方法的，以下代码将无法编译：复制代码代码... 博文来自： wang123459...	阅读数 232	
视频 理，ConcurrentHashMap原理。Lambda表达式与StreamAPI详解，接口新特性，时间日期API用法与原理详解 下载	08-26	
Lambda表达式详解 数，可以把lamda表达式理解为一段可以传递的代码，可以写出简洁，更... 博文来自： 你看到的逆袭...	阅读数 207	
Optional详解 作类。Optional。下面介绍下他的方法。静态方法：Optional.of(T) 返回O... 博文来自： fw118958的专栏	阅读数 6401	
属性，以下定义了行被选中时的响应：varlastSel;jQuery("#gridid").jqGrid... 博文来自： wpcxyking的专栏	阅读数 921	
工具类 ***由于Java的简单类型不能够精确的对浮点数进行运算，这个工具类提... 博文来自： NFA_YY的博客	阅读数 762	
运算，今天我们看下itk中的有哪些基本的数学运算。//1、加，减，乘（注... 博文来自： 就这样吧	阅读数 1176	
详解 文：阅读原文访问接口的默认方法Lambda表达式中是无法访问到默认方... 博文来自： JAVA葵花宝典	阅读数 494	
50896975前言：Java8已经发布很久了，很多报道表... 博文来自： tigerhu256的...	阅读数 798	
介绍 va8简明教程中文API：中文APIJAVA8十大新特性详解：JAVA8十大新特... 博文来自： 不想当码农的...	阅读数 607	
指南 经验的资深程序员Per-ÅkeMinborg，主要介绍如何灵活地解析Java中的... 博文来自： wangpeng198...	阅读数 1359	
不要再用for循环了 3是替代for循环，因为java8推出了强大的流stream，关于流的用法很多，... 博文来自： williamtsang的...	阅读数 1万+	
/**@Author:cxh*@CreateTime:18/3/815:54*@ProjectName:JavaBaseT... 博文来自： iCoding91	阅读数 2319	
df va8 新特性api 尚硅谷 Java8 新特性api 下载	12-05	
la基础语法 入了一个新的操作符“->”,该操作符称为箭头操作符,lambd操作符,该箭头... 博文来自： qq_40794266...	阅读数 142	
新特性Java8 新特性Java8 新特性Java8 新特性 下载	04-27	
教程 ;与集合类bulkoperation教程。迄今为止最全面的中文原创javalambda表... 博文来自： 自由之子	阅读数 16万+	
!) .1语法...31.2 Lambda表达式实例...3Java8Tester.java文件...31.3变量作... 博文来自： yitian_66的博客	阅读数 6万+	
lambda Stream Function Consumer Predicate Supplier ...		

的 Lambda表达式

阅读数 2万+

登录

注册

×

达式在这一章，我们说一说Lambda表达式的语法。我们将从经典的Java... 博文 来自： Larry Wang的...

4

引lambda表达式

阅读数 4万+

这次发布的改动比较大，很多人将这次改动与Java5的升级相提并论。J... 博文 来自： bitcarmanlee...

2

elStream

阅读数 3666

己很普及了。当然不排除有些公司由于太大积重难返，还在使用老版本... 博文 来自： Elementary Co...

4

阅读数 916
p.Entrymap:paramsMap.entrySet()){params.add(newBasicNameValueP... 博文 来自： 亦轩、Javer的...

4

阅读数 763
在方法体二：支持Lambda表达式首先看看在老版本的Java中是如何排列... 博文 来自： uhgagnu的博客

4

内时间和日期API（终章）

阅读数 1303

和Calendar这两个类处理时间，但有的特性只在某一个类有提供，比如用... 博文 来自： 一大三千的博客

4

的讲解视频

06-25

Java8新特性视频 最新的讲解视频

下载

百度云

11-21

演示的源代码，，，，，

下载

实战视频教程完整版

阅读数 3730

资源，大家一起学习qq群：377215114Java8实战视频-01让方法参数具备... 博文 来自： IT资源分享

rn、awaitTermination、shutdownNow的作用与区别

阅读数 803

xecutorService，当此方法被调用时，ExecutorService停止接收新的任务... 博文 来自： 坚持，让梦想...

CompletableFuture 详解

阅读数 8952

CompletableFuture类实现了CompletionStage和Future接口。Future是Java5添... 博文 来自： mrxiky的专栏

mpletableFuture：组合式、异步编程（七）

阅读数 3万+

异步应用Future接口的局限性Future接口可以构建异步应用，但依然有其... 博文 来自： 一大三千的博客

横版RPG游戏制作（二）

阅读数 6755

二）现在我们到了di 博文 来自： sinolzeng的专栏

0150002失败

阅读数 2万+

释时没有任何错误，但是运行时就是提示“应用程序正常初始化失败”！！ ... 博文 来自： DDR的专栏

(超详细)

阅读数 1万+

open，fclose 表头文件 #include 定义函数 FILE * fopen(const char * path... 博文 来自： 独旅天涯

一）：简介

阅读数 5万+

图像素概念是2003年Xiaofeng Ren提出和发展起来的图像分割技术，是指... 博文 来自： electech6的博客

阅读数 2万+
://vv.video.qq.com/geturl?vid=v00149uf4ir&otype=json 高清视频（分段视... 博文 来自： 专注于互联网...

阅读数 4575
两种方式POST方式和GET方式。前者通过Http消息实体发送数据给服务... 博文 来自： 借你一秒

n算法解析

阅读数 49万+

Python怎么学

转置AI人工智能指南

Java学习路线

28 天算法训练营

2019 Python 开发者日

什么笔记本好用

mac系统下

- ent方法

阅读数 9196

F本地组策略编辑器： 在计算机配置->管理模板->Windows组件->Internet... 博文 来自： .
- linux下的tomcat（后附resin的配置）

阅读数 1827

所以想远程监控一下jvm的运行情况，我在网上也找了不少文章和办法， ... 博文 来自： keketrr的专栏
- 序

阅读数 1万+

程序。使用“%{”和“%}”。例如 %{。。。}%} 即可。经典方法是用 if 0, ... 博文 来自： 知识小屋
- 搭建遇到的问题解决方案汇总

阅读数 4万+

socket.server.WsServerContainer cannot be cast to javax.websocket.se... 博文
- 安全套接字层的超文本传输协议 或者是 HTTP over SSL) 是一个 Netsca...

阅读数 7万+

博文 来自： whatday的专栏
- 无法注入的问题（与文件包位置有关）

阅读数 13万+

aven构建，大致结构如下： 核心Spring框架一个module spring-boot-base... 博文 来自： 开发随笔
- 九：android studio导出jar包（Module）并获得手机信息

阅读数 6万+

程，但是可以在普通的Project中加入Module。所谓的Module就是我们通... 博文 来自： 懒人的技术笔记
- 之源码完美解析（上）

阅读数 8193

不是为了解决滑动冲突的。实际上，如果仅仅是为了解决滑动冲突的， ... 博文 来自： qq_36523667...
- 化

阅读数 2万+

博文 来自： hero_fantao的...
- means源码分析（二）

阅读数 4362

源码分析（二） 博文 来自： 锐之锋芒
- l error compiling: 无效的目标发行版

阅读数 1万+

119999990123162/31622105.html http://lyking2001.iteye.com/blog/8374... 博文 来自： zhao1949的专栏
- 同时调用多个倒计时(最新的)

阅读数 37万+

用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都... 博文 来自： Websites
- 的几个重要概念介绍

阅读数 2万+

重要概念介绍 发布于 2015-09-23 23:02:48 | 11 次阅读 | 评论: 0 | 来源: 网... 博文 来自： ichsonx的专栏
- 总

阅读数 4万+

每天都需要学习，需要get新技能，才能不被淘汰，成功的人总是贵在坚... 博文 来自： 安辉
- ，防止用户未登录访问

阅读数 1万+

，比如有时，我们不希望用户没有进行登录访问后台的操作页面，而且这... 博文 来自： 沉默的鲨鱼的...
- 信聊天机器人

阅读数 4347

/www.tuling123.com/ python3环境下安装wxpy pip install wxpy linux下还... 博文 来自： getcomputerst...
- 支付PHP教程(thinkPHP5公众号支付)/JSSDK的使用

阅读数 10万+

分享 本文承接之前发布的博客《微信支付V3微信公众号支付PHP教程/thi... 博文 来自： Marswill
- 阅读数 4366

例： 命令1： /sbin/iptables -I INPUT -p tcp --dport 3306-j ACCEPT 命令... 博文 来自： 少侠,...
- ： 让城市社

.....



快乐崇拜234

 博客专家

关注

原创

粉丝

喜欢

评论

159

472

103

96

等级:  博客 6

访问: 63万+

积分: 6647

排名: 5835

勋章:  

关于作者

多年工作经验 javaer, 一直专注于java领域的学习研究。
精通java分布式高并发等系统架构。
曾就职于 京东、网易 等公司。
联系方式: **3089008201** 技术交流QQ群:
684457529

最新文章

spring boot 2.1学习笔记【二十】
SpringBoot 2 freemarker bootstrap 集成
CMS收集器几个参数详解 -
XX:CMSInitiatingOccupancyFraction,
CMSFullGCsBeforeCompaction
CentOS 7安装并启动Google浏览器
CentOS 7 安装配置 OpenVPN 客户端
jdk11源码--CopyOnWriteArrayList源码分析

博主专栏

 akka学习教程

文章数: 14 篇 访问量: 11万+

 redis学习教程

文章数: 11 篇 访问量: 16万+

 Java11源码分析

文章数: 0 篇 访问量: 33

 spring boot 2.X/spring cloud
Greenwich

文章数: 43 篇 访问量: 262

个人分类

Java11源码分析	3篇
分布式+高并发	27篇
dubbo	2篇
springboot	31篇
java/java8	34篇

展开

归档

2019年3月	5篇
2019年2月	28篇
2019年1月	26篇
2018年12月	7篇
2018年7月	2篇

展开

热门文章

Redis学习笔记（五） jedis(JedisCluster)操作Redis集群 redis-cluster

阅读数 59927

服务化实战之 dubbo、dubbox、motan、thrift、grpc等RPC框架比较及选型

阅读数 52138

akka学习教程(一)简介

阅读数 28144

使用spring-data-redis实现incr自增

阅读数 24713

akka学习教程(十四) akka分布式实战

阅读数 17716

最新评论

fullgc问题解决： Full G...

fgyibupi： [reply]zw547779770[/reply] 那个不太好用，还不如直接使用命令 【java -XX:+PrintFla ...

fullgc问题解决： Full G...

fgyibupi： [reply]qq_36596145[/reply] 那个不太好用，还不如直接使用命令 【java -XX:+PrintFla ...

fullgc问题解决： Full G...

qq_36596145： 这个是什么app？

spring boot 2.1学习...

fgyibupi： [reply]weixin_43836141[/reply] 谢谢 ...

spring boot 2.1学习...

weixin_43836141： 这才是spring炉火纯青的回答，答主大牛，先拜了



CSDN学院




CSDN企业招聘

 客服

 kofu@csdn.net

[关于我们](#) [招聘](#) [广告服务](#) [网站地图](#)

 百度提供站内搜索 京ICP证19004658号
©1999-2019 北京创新乐知网络技术有限公司

网络110报警服务 [经营性网站备案信息](#)
[北京互联网违法和不良信息举报中心](#)
[中国互联网举报中心](#)

[登录](#)

[注册](#)

×

 4

 2







←

→