愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第1章 Java高级核心玩转JDK8~13新特性课程介绍

第1集 Java新特性玩转JDK8~13课程大纲讲解

简介：讲解 Java新特性玩转JDK8~13课程大纲讲解

- 课程适合人群：中高级后端工程师、全栈工程师、架构师 需要的基础：需要有javase基础，熟悉eclipse或者IDEA，如果不会可以看IDEA教程
- 课程大纲
- 学后水平 掌握JDK8 新增的日期处理、自带加解密、Optional特性 掌握JKD8 最核心的lambda表达式和可以实现自定义函数式接口 掌握 Function、Consumer、Supplier、Predicate四大接口使用 玩转Stream流式编程map、filter、reduce、match等核心方法使用 玩转JKD8 Collector收集器和分组聚合统计 掌握JDK8特性 综合实战之电商订单数据处理 掌握JVM新的内存空间MetaSpace和try-with-resouce使用 掌握JDK9新工具Jshell和增强API 掌握JDK10的局部变量类型推断Var实践 玩转JDK11的标准HttpClient提交GET,POST、异步请求和Http2请求 掌握JDK13新特性多行文本块和增强switch等 ...更多看课程目录和视频

第2集 Java新特性之JDK8相关开发环境准备

简介：讲解JDK8相关特性的开发环境准备

- 官方概览 https://www.oracle.com/technetwork/cn/java/javase/8-whats-new-2157071-zhs.html
    - 重点
        - 日期
        - lambda
        - 函数式编程
        - Stream

        - 其他
- Java 8 是目前最被广泛使用的版本，并且其用户群体非常广，所以Java 8 是一个非常成功的版本，阿里、京东等大厂里面主流应用都是JDK8版本
- 相关环境变量安装
    - 本地有eclipse或者idea
    - jdk8安装包：官方地址 https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html 如果地址过期则使用本集课程资料里面的包即可
    - jdk8安装 Win10 https://blog.csdn.net/Coco_liukeke/article/details/79834680 Win8 https://jingyan.baidu.com/article/7c6fb4282f1f6580642c90e1.html Linux https://www.jianshu.com/p/5cee1c7a4dd4 Mac https://www.jianshu.com/p/a85658902f26

win64位-JDK1.8下载||https://pan.baidu.com/s/1fMNaZ0JgySo2MzBT90T5Tw ||jjw3

Linux64位-JDK1.8||https://pan.baidu.com/s/1CDpW-UNYyje-p0BxaNtncQ ||nwyd

Mac-JDK1.8下载||https://pan.baidu.com/s/1liT9kSLicpXEAd7AdA0nOg ||5bpk

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第2章 Java高级核心玩转JDK8新特性之接口和日期处理

第1集 Java新特性玩转JDK8之default关键字

简介：讲解jdk8新特性default关键字

- 创建一个java基础项目
- 在jdk1.8以前接口里面是只能有抽象方法，不能有任何方法的实现的
- jdk1.8里面打破了这个规定，引入了新的关键字default，使用default修饰方法，可以在接口里面定义具体的方法实现
- 默认方法： 接口里面定义一个默认方法，这个接口的实现类实现了这个接口之后，不用管这个default修饰的方法就可以直接调用，即接口方法的默认实现

```
public interface Animal {

  void run();

  void eat();

  default void breath(){
      System.out.println("使用氧气呼吸");
  }
}
```

- 静态方法: 接口名.静态方法来访问接口中的静态方法

```
public interface Animal {

  void run();

  void eat();

  default void breath(){
      System.out.println("使用氧气呼吸");
  }

  static void test(){
      System.out.println("这是静态方法");
  }
}
```

第2集 Java新特性玩转JDK8之新增base64加解密API

简介：讲解jdk1.8新增Base64 api

- 什么是Base64编码 Base64是网络上最常见的用于传输8Bit字节码的编码方式之一，Base64就是一种基于64个可打印字符来表示二进制数据的方法 基于64个字符A-Z,a-z，0-9，+，/的编码方式，是一种能将任意二进制数据用64种字元组合成字符串的方法，而这个二进制数据和字符串资料之间是可以互相转换的，在实际应用上，Base64除了能将二进制数据可视化之外，也常用来表示字串加密过后的内容

  推荐一个文章：https://blog.csdn.net/wo541075754/article/details/81734770

- 早期java要使用Base64怎么做
  - 使用JDK里sun.misc套件下的BASE64Encoder和BASE64Decoder这两个类

```
        BASE64Encoder encoder = new BASE64Encoder();
        BASE64Decoder decoder = new BASE64Decoder();
        String text = "小滴课堂";
        byte[] textByte = text.getBytes("UTF-8");
       //编码
        String encodedText = encoder.encode(textByte);
        System.out.println(encodedText);
       //解码
       System.out.println(new String(decoder.decodeBuffer(encodedText),
"UTF-8"));
```

    缺点：编码和解码的效率比较差，公开信息说以后的版本会取消这个方法

  - Apache Commons Codec有提供Base64的编码与解码 缺点：是需要引用Apache Commons Codec

- jdk1.8之后怎么玩？(首选推荐)
  - Jdk1.8的java.util包中，新增了Base64的类
  - 好处：不用引包，编解码销量远大于 sun.misc 和 Apache Commons Codec

```
 Base64.Decoder decoder = Base64.getDecoder();
    Base64.Encoder encoder = Base64.getEncoder();
    String text = "小滴课堂";
    byte[] textByte = text.getBytes("UTF-8");
    //编码
    String encodedText = encoder.encodeToString(textByte);
    System.out.println(encodedText);
    //解码
    System.out.println(new String(decoder.decode(encodedText), "UTF-
8"));
```

第3集 Java新特性玩转JDK8之时间日期处理类上集

简介：讲解jdk8之后处理时间的api

- 时间处理再熟悉不过，SimpleDateFormat,Calendar等类 旧版缺点： java.util.Date 是非线程安全的 API设计比较差，日期/时间对象比较，加减麻烦
- Java 8通过发布新的Date-Time API (JSR 310)来进一步加强对日期与时间的处理
  - 新增了很多常见的api，如日期/时间的比较，加减，格式化等
  - 包所在位置 java.time
  - 核心类

    ```
    LocalDate：不包含具体时间的日期。

    LocalTime：不含日期的时间。

    LocalDateTime：包含了日期及时间。
    ```

  - LocalDate 常用API

    ```
    LocalDate today = LocalDate.now();
    System.out.println("今天日期：" + today);
    //获取年，月，日，周几
    System.out.println("现在是哪年:"+today.getYear());
    System.out.println("现在是哪月:"+today.getMonth());
    System.out.println("现在是哪月(数字):"+today.getMonthValue());
    System.out.println("现在是几号:"+today.getDayOfMonth());
    System.out.println("现在是周几:"+today.getDayOfWeek());

    //加减年份，加后返回的对象才是修改后的， 旧的依旧是旧的
    ```

```
LocalDate changeDate =  today.plusYears(1);

System.out.println("加后是哪年:"+changeDate.getYear());
System.out.println("旧的是哪年:"+today.getYear());

//日期比较
System.out.println("isAfter: "+changeDate.isAfter(today));

//getYear()    int    获取当前日期的年份
//getMonth()    Month    获取当前日期的月份对象
//getMonthValue()    int    获取当前日期是第几月
//getDayOfWeek()    DayOfWeek    表示该对象表示的日期是星期几
//getDayOfMonth()    int    表示该对象表示的日期是这个月第几天
//getDayOfYear()    int    表示该对象表示的日期是今年第几天
//withYear(int year)    LocalDate    修改当前对象的年份
//withMonth(int month)    LocalDate    修改当前对象的月份
//withDayOfMonth(int dayOfMonth)    LocalDate    修改当前对象在当月的日期
//plusYears(long yearsToAdd)    LocalDate    当前对象增加指定的年份数
//plusMonths(long monthsToAdd)    LocalDate    当前对象增加指定的月份数
//plusWeeks(long weeksToAdd)    LocalDate    当前对象增加指定的周数
//plusDays(long daysToAdd)    LocalDate    当前对象增加指定的天数
//minusYears(long yearsToSubtract)    LocalDate    当前对象减去指定的年数
//minusMonths(long monthsToSubtract)    LocalDate    当前对象减去注定的月数
//minusWeeks(long weeksToSubtract)    LocalDate    当前对象减去指定的周数
//minusDays(long daysToSubtract)    LocalDate    当前对象减去指定的天数
//compareTo(ChronoLocalDate other)    int    比较当前对象和other对象在时间上的大小，返回值如果为正，则当前对象时间较晚，
//isBefore(ChronoLocalDate other)    boolean    比较当前对象日期是否在other对象日期之前
//isAfter(ChronoLocalDate other)    boolean    比较当前对象日期是否在other对象日期之后
//isEqual(ChronoLocalDate other)    boolean    比较两个日期对象是否相等
```

- LocalTime 常用API
- LocalDateTime 常用API

第4集 Java新特性玩转JDK8之时间日期处理类下集

简介：讲解jdk8之后处理时间的api

- 日期时间格式化
  - JDK8之前：SimpleDateFormat来进行格式化，但SimpleDateFormat并不是线程安全的

- JDK8之后：引入线程安全的日期与时间DateTimeFormatter

```
LocalDateTime ldt = LocalDateTime.now();
System.out.println(ldt);
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
String ldtStr = dtf.format(ldt);
System.out.println(ldtStr);
```

- 获取指定的日期时间对象

```
LocalDateTime ldt = LocalDateTime.of(2020, 11, 11, 8, 20, 30);
System.out.println(ldt);
```

- 计算日期时间差 java.time.Duration

```
    LocalDateTime today = LocalDateTime.now();
    System.out.println(today);
    LocalDateTime changeDate = LocalDateTime.of(2020,10,1,10,40,30);
    System.out.println(changeDate);

    Duration duration = Duration.between( today,changeDate);//第二个参数减第一
个参数
    System.out.println(duration.toDays());//两个时间差的天数
    System.out.println(duration.toHours());//两个时间差的小时数
    System.out.println(duration.toMinutes());//两个时间差的分钟数
    System.out.println(duration.toMillis());//两个时间差的毫秒数
    System.out.println(duration.toNanos());//两个时间差的纳秒数
```

第5集 Java新特性玩转JDK8之Optional类

简介：讲解jdk8新增的Optional类

- Optional 类有啥用
  - 主要解决的问题是空指针异常（NullPointerException）
  - 怎么解决?
    - 本质是一个包含有可选值的包装类，这意味着 Optional 类既可以含有对象也可以为空
- 创建Optional类
  - of()
    - null 值作为参数传递进去,则会抛异常

```
Optional<Student> opt = Optional.of(user);
```

- ofNullable()
  - 如果对象即可能是 null 也可能是非 null，应该使用 ofNullable() 方法

```
Optional<Student> opt = Optional.ofNullable(user);
```

- 访问 Optional 对象的值
  - get() 方法

```
Optional<Student> opt = Optional.ofNullable(student);
Student s = opt.get();
```

  - 如果值存在则isPresent()方法会返回true，调用get()方法会返回该对象一般使用get之前需要先验证是否有值，不然还会报错

```
public static void main(String[] args) {
        Student student = null;
        test(student);
}

public static void  test(Student student){
        Optional<Student> opt = Optional.ofNullable(student);
        System.out.println(opt.isPresent());

}
```

- 兜底 orElse方法
  - orElse()如果有值则返回该值，否则返回传递给它的参数值

```
Student student1 = null;
Student student2 = new Student(2);
Student result = Optional.ofNullable(student1).orElse(student2);
System.out.println(result.getAge());
```

```
Student student = null;
int result = Optional.ofNullable(student).map(obj-
>obj.getAge()).orElse(4);
System.out.println(result);
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第3章 Java高级核心玩转JDK8 Lambda表达式

第1集 Java新特性玩转 JDK8之 lambda表达式

简介：讲解什么是函数式编程和什么是lambda表达式

- 在JDK8之前，Java是不支持函数式编程的，所谓的函数编程，即可理解是将一个函数（也称为“行为”）作为一个参数进行传递， 面向对象编程是对数据的抽象（各种各样的POJO类），而函数式编程则是对行为的抽象（将行为作为一个参数进行传递）
- java创建线程再熟悉不过了
  - jdk8之前创建线程

```
new Thread(new Runnable() {
     @Override
     public void run() {
         System.out.println("小滴课堂学习Java架构教程");
     }
 });
```

  - jdk8之后Lambda表达式则只需要使用一句话

```
  new Thread(() -> System.out.println("小滴课堂学习Java架构教程"));
```

- 集合容器里面的字符串排序 使用前

```
List<String> list =Arrays.asList("aaa","ggg","ffff","ccc");

        Collections.sort(list, new Comparator<String>() {
                    @Override
                    public int compare(String a, String b) {
                        return b.compareTo(a);
                    }
                }
        );
        for (String string : list) {
            System.out.println(string);
        }
```

使用后

```
List<String> list =Arrays.asList("aaa","ggg","ffff","ccc");
   Collections.sort(list, (a,b)->b.compareTo(a)
   );
   for (String string : list) {
       System.out.println(string);
   }
```

- lambda表达式 使用场景(前提)：一个接口中只包含一个方法，则可以使用Lambda表达式，这样的接口称之为“函数接口” 语法： (params) -> expression

```
第一部分为括号内用逗号分隔的形式参数，参数是函数式接口里面方法的参数；第二部分为一个箭头符号：->；第三部分为方法体，可以是表达式和代码块

参数列表 ：
    括号中参数列表的数据类型可以省略不写
    括号中的参数只有一个，那么参数类型和()都可以省略不写

方法体：
    如果{}中的代码只有一行，无论有返回值，可以省略{}，return，分号，要一起省略，其他则需要加上
```

好处： Lambda 表达式的实现方式在本质是以匿名内部类的方式进行实现

```
重构现有臃肿代码，更高的开发效率，尤其是集合Collection操作的时候，后续会讲到
```

第2集 Java新特性玩转JDK8之自定义函数式编程实战

简介：玩转Lambda表达式,自定义lambda接口编程

- 自定义lambda接口流程
  - 定义一个函数式接口 需要标注此接口 @FunctionalInterface，否则万一团队成员在接口上加了其他方法则容易出故障
  - 编写一个方法，输入需要操做的数据和接口
  - 在调用方法时传入数据 和 lambda 表达式，用来操作数据
- 需求，定义一个可以使用加减乘除的接口 以前需要定义4个方法

  使用Lambda表达式后

```
@FunctionalInterface
public interface OperFunction<R,T> {

    R operator(T t1, T t2);

}
```

```
    public static void main(String[] args) throws Exception {

        System.out.println(operator(20, 5, (Integer x, Integer y) -> {
            return x * y;
        }));

        System.out.println(operator(20, 5, (x, y) -> x + y));
        System.out.println(operator(20, 5, (x, y) -> x - y));
        System.out.println(operator(20, 5, (x, y) -> x / y));
    }
    public static Integer operator(Integer x, Integer y,
OperFunction<Integer, Integer> of) {
        return of.operator(x, y);
    }
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第4章 Java高级核心玩转JDK8 函数式编程

第1集 Java新特性玩转JDK8之函数式编程 Function

简介：讲解jdk8里面的函数式编程 Function接口的使用

- Lambda表达式必须先定义接口，创建相关方法之后才可使用，这样做十分不便，其实java8已经内置了许多接口, 例如下面四个功能型接口，所以一般很少会由用户去定义新的函数式接口
- Java8的最大特性就是函数式接口，所有标注了@FunctionalInterface注解的接口都是函数式接口

```
Java8 内置的四大核心函数式接口

Consumer<T> : 消费型接口：有入参，无返回值

    void accept(T t);

Supplier<T> : 供给型接口：无入参，有返回值

    T get();

Function<T, R> : 函数型接口：有入参，有返回值

    R apply(T t);

Predicate<T> : 断言型接口：有入参，有返回值，返回值类型确定是boolean

    boolean test(T t);
```

- Function
  - 传入一个值经过函数的计算返回另一个值
  - T：入参类型，R：出参类型

    调用方法：R apply(T t)

```
//@param <T> the type of the input to the function
//@param <R> the type of the result of the function
```

```
@FunctionalInterface
public interface Function<T, R> {

    /**
     * Applies this function to the given argument.
     *
     * @param t the function argument
     * @return the function result
     */
    R apply(T t);

}
```

- 作用：将转换逻辑提取出来，解耦合
- 不要看过于复杂，就是一个接口,下面是自定义实现

```
public class FunctionObj implements Function {
    @Override
    public Object apply(Object o) {
        return o+"经过apply处理拼接上了";
    }
}
```

- 常规使用

```
// 输出入参的10倍
Function<Integer, Integer> func = p -> p * 100;
func.apply(100);
```

第2集 Java新特性玩转JDK8之函数式编程 BiFunction

简介：讲解jdk8里面的函数式编程 BiFunction接口的使用

- BiFunction Function只能接收一个参数，如果要传递两个参数,则用 BiFunction

```
@FunctionalInterface
public interface BiFunction<T, U, R> {
    R apply(T t, U u);

}
```

- 需求: 上两节课，两个数的四则运算

```
    public static void main(String[] args) {

        System.out.println(operator(10,21,(a,b)->a+b));
        System.out.println(operator(10,2,(a,b)->a-b));
        System.out.println(operator(8,4,(a,b)->a*b));
        System.out.println(operator(10,2,(a,b)->a/b));

    }

    public static Integer operator(Integer a, Integer b, BiFunction<Integer,
Integer, Integer> bf) {

        return bf.apply(a, b);
    }
```

第3集 Java新特性玩转JDK8之函数式编程 Consumer

简介：讲解jdk8里面的函数式编程 Consumer接口的使用

- Consumer 消费型接口：有入参，无返回值
- 将 T 作为输入，不返回任何内容

  调用方法：void accept(T t);

```
@FunctionalInterface
public interface Consumer<T> {
void accept(T t);
}
```

- 用途： 因为没有出参，常用于打印、发送短信等消费动作

```
   public static void main(String[] args) throws Exception {
         Consumer<String> consumer = obj->{
             System.out.println(obj);
             System.out.println("调用短信接口发送短信，或者打印日志");
         };

//         sendMsg("8888888",obj->{
//             System.out.println(obj);
//             System.out.println("调用短信接口发送短信，或者打印日志");
```

```
//        });
        sendMsg("8888888",consumer);
    }
    public static void sendMsg(String phone,Consumer<String> consumer){
        consumer.accept(phone);
    }
```

- 典型应用，集合的foreach

```
List<String> list  = Arrays.asList("aaa","bbb");
list.forEach(obj->{
//TODO
});
```

第4集 Java新特性玩转JDK8之函数式编程 Supplier

简介：讲解jdk8里面的函数式编程 Supplier接口的使用

- Supplier: 供给型接口：无入参，有返回值
- T：出参类型；没有入参

  调用方法：T get();

```
@FunctionalInterface
public interface Supplier<T> {
T get();
}
```

- 用途： 泛型一定和方法的返回值类型是一种类型，如果需要获得一个数据,并且不需要传入参数,可以使用Supplier接口，例如 无参的工厂方法，即工厂设计模式创建对象，简单来说就是 提供者

```
    public static void main(String[] args) {
        //Student student = new Student();
        Student student = newStudent();
        System.out.println(student.getName());
    }

    public static Student newStudent(){
        Supplier<Student> supplier = ()-> {
            Student student = new Student();
```

```
            student.setName("默认名称");
            return student;
        };
        return supplier.get();
    }

    class Student{
        private String name;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
    }
```

第5集 Java新特性玩转JDK8之函数式编程 Predicate

简介：讲解jdk8里面的函数式编程 Predicate接口的使用

- Predicate: 断言型接口：有入参，有返回值，返回值类型确定是boolean
- T：入参类型；出参类型是Boolean

  调用方法：boolean test(T t);

```
@FunctionalInterface
public interface Predicate<T> {
boolean test(T t);
}
```

- 用途： 接收一个参数，用于判断是否满足一定的条件，过滤数据

```
    public static void main(String[] args) {

        List<String> list =
Arrays.asList("awewrwe","vdssdsd","aoooo","psdddsd");
```

```
        List<String> results = filter(list,obj->obj.startsWith("a"));

        System.out.println(results);
    }

    public static List<String> filter(List<String> list,
Predicate<String> predicate) {
        List<String> results = new ArrayList<>();
        for (String str : list) {
            if (predicate.test(str)) {
                results.add(str);
            }
        }

        return results;
    }
```

第6集 Java新特性玩转JDK8之方法与构造函数引用

简介：讲解JDK8新特性中 方法引用与构造函数引用

- 以前方法调用 对象.方法名 或者 类名.方法名
- jdk1.8提供了另外一种调用方式 ::

```
说明:方法引用是一种更简洁易懂的lambda表达式，操作符是双冒号::，用来直接访问类或者实例已经存在的方法或构造方法

通过方法引用，可以将方法的引用赋值给一个变量

语法：左边是容器（可以是类名，实例名），中间是" :: "，右边是相应的方法名

静态方法，则是ClassName::methodName。如 Object ::equals
实例方法，则是Instance::methodName
构造函数，则是 类名::new;

单个参数
```

```
Function<入参1, 返回类型> func = 方法引用
应用 func.apply(入参);

2个参数
BiFunction<入参1,入参2, 返回类型> func = 方法引用
应用 func.apply(入参1,入参2);
```

```
public class TestJdk8 {

public static void main(String[] args) {
    // 使用双冒号::来构造静态函数引用
    Function<String, Integer> fun = Integer::parseInt;
    Integer value = fun.apply("1024");
    System.out.println(value);

    // 使用双冒号::来构造非静态函数引用
    String content = "欢迎来到小滴课堂学习";
    Function<Integer, String> func = content::substring;
    String result = func.apply(1);
    System.out.println(result);

    // 构造函数引用，多个参数
    BiFunction<String, Integer, User> biFunction = User::new;
    User user1 = biFunction.apply("小滴课堂", 28);
    System.out.println(user1.toString());

    //构造函数引用，单个参数
    Function<String, User> function = User::new;
    User user2 = function.apply("小D");
    System.out.println(user2.toString());

    // 函数引用也是一种函数式接口，可以将函数引用作为方法的参数
    sayHello(String::toUpperCase, "xdclass.net");
}

/**
 *
 * @param func 函数引用
 * @param param 对应的参数
     */
    private static void sayHello(Function<String, String> func, String
param) {
        String result = func.apply(param);
        System.out.println(result);
    }
}
```

```
class User {
        private String username;
        private Integer age;

        public User() {

        }
        public User(String username) {
            this.username = username;
        }

        public User(String username, Integer age) {
            this.username = username;
            this.age = age;
        }
    }
```

愿景："让编程不在难学，让技术与生活更加有趣"

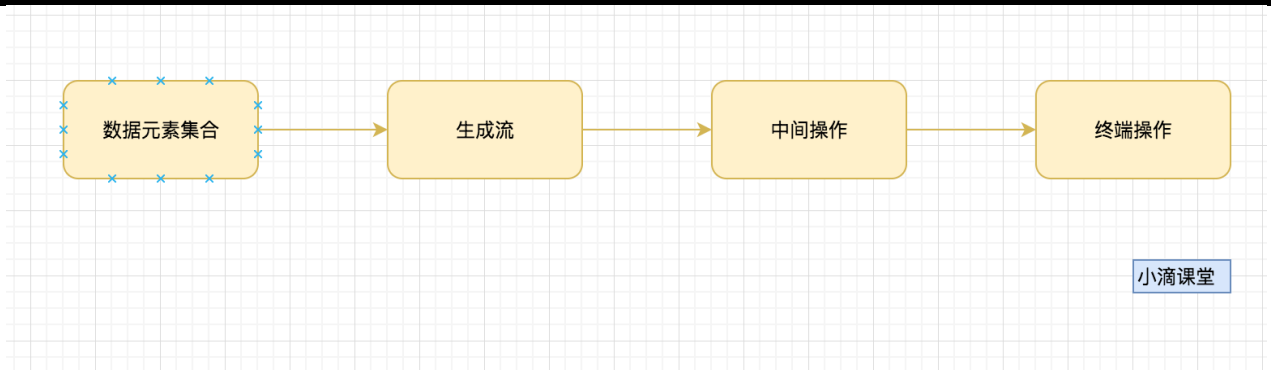
更多架构课程请访问官网： xdclass.net

第5章 Java高级核心之玩转 JDK8 集合框架

第1集 Java新特性玩转JDK8之流Stream实战

简介：讲解jdk8里面的流stream的使用

- 什么是stream
  - Stream 中文称为 “流”，通过将集合转换为这么一种叫做 “流”的元素队列，通过声明性方式，能够对集合中的每个元素进行一系列并行或串行的流水线操作
  - 元素是特定类型的对象，所以元素集合看作一种流， 流在管道中传输， 且可以在管道的节点上进行处理， 比如 排序，聚合，过滤等操作



- 操作详情
  - 数据元素便是原始集合，如List、Set、Map等
  - 生成流，可以是串行流**stream()** 或者并行流 **parallelStream()**
  - 中间操作，可以是 排序，聚合，过滤，转换等
  - 终端操作，很多流操作本身就会返回一个流，所以多个操作可以直接连接起来，最后统一进行收集
  - 概览stream接口源码

- 快速上手

```
List<String> list = Arrays.asList("springboot教程","微服务教程","并发编程","压力测试","架构课程");

List<String> resultList =  list.stream().map(obj->"在小滴课堂学："+obj).collect(Collectors.toList());

System.out.println(resultList);
```

第2集 Java新特性玩转JDK8之流操作map和filter函数

简介：讲解jdk8里面的流stream里的map和filter函数的使用

- map函数
  - 将流中的每一个元素 T 映射为 R（类似类型转换）
  - 上堂课的例子就是，类似遍历集合，对集合的每个对象做处理
  - 场景：转换对象，如javaweb开发中集合里面的DO对象转换为DTO对象

```
List<String> list = Arrays.asList("springboot教程","微服务教程","并发编
程","压力测试","架构课程");

List<String> resultList =  list.stream().map(obj->"在小滴课堂
学："+obj).collect(Collectors.toList());

System.out.println(resultList);
```

```
   List<User> list = Arrays.asList(new User(1,"小东","123"),new
User(21,"jack","rawer"),
              new User(155,"tom","sadfsdfsdfsd"),
        new User(231,"marry","234324"),new User(100,"小
D","122223"));
      List<UserDTO> userDTOList = list.stream().map(obj->{
          UserDTO userDTO = new UserDTO(obj.getId(),obj.getName());
          return userDTO;
      }).collect(Collectors.toList());
      System.out.println(userDTOList);
```

- filter函数
  - 用于通过设置的条件过滤出元素
  - 需求：过滤出字符串长度大于5的字符串

```
List<String> list = Arrays.asList("springboot", "springcloud",
"redis", "git", "netty", "java", "html", "docker");

List<String> resultList = list.stream().filter(obj -> obj.length() >
5).collect(Collectors.toList());

System.out.println(resultList);
```

  - 场景：主要用于筛选过滤出符合条件的元素

第3集 Java新特性玩转JDK8之流操作limit和sorted函数

简介：讲解jdk8里面的流stream里的limit和sorted函数的使用

- sorted函数
  - sorted() 对流进行自然排序, 其中的元素必须实现Comparable 接口

```
List<String> list = Arrays.asList("springboot", "springcloud",
"redis", "git", "netty", "java", "html", "docker");

List<String> resultList =
list.stream().sorted().collect(Collectors.toList());
```

- sorted(Comparator<? super T> comparator) 用来自定义升降序

```
List<String> list = Arrays.asList("springboot", "springcloud",
"redis", "git", "netty", "java", "html", "docker");

//根据长度进行排序
//List<String> resultList =
list.stream().sorted(Comparator.comparing(obj ->
obj.length())).collect(Collectors.toList());
//List<String> resultList =
list.stream().sorted(Comparator.comparing(obj ->
obj.length(),Comparator.reverseOrder())).collect(Collectors.toList()
);
//List<String> resultList =
list.stream().sorted(Comparator.comparing(String::length).reversed()
).collect(Collectors.toList());


System.out.println(resultList);
```

- limit函数
  - 截断流使其最多只包含指定数量的元素

```
List<String> list = Arrays.asList("springboot", "springcloud",
"redis", "git", "netty", "java", "html", "docker");

//limit截取
List<String> resultList =
list.stream().sorted(Comparator.comparing(String::length).reversed()
).limit(3).collect(Collectors.toList());
System.out.println(resultList);
```

第4集 Java新特性玩转JDK8之流操作allMatch和anyMatch函数

简介：讲解jdk8里面的流stream里的allMatch和anyMatch函数的使用

- allMatch函数

- 检查是否匹配所有元素，只有全部符合才返回true

```
List<String> list = Arrays.asList("springboot", "springcloud", "redis",
"git", "netty", "java", "html", "docker");

boolean flag = list.stream().allMatch(obj->obj.length()>1);

System.out.println(flag);
```

- anyMatch函数
  - 检查是否至少匹配一个元素

```
List<String> list = Arrays.asList("springboot", "springcloud", "redis",
"git", "netty", "java", "html", "docker");

boolean flag = list.stream().anyMatch(obj->obj.length()>18);

System.out.println(flag);
```

- 先看方法入参，返回值，再看方法描述

第5集 Java新特性玩转JDK8之流操作max和min函数

简介：讲解jdk8里面的流stream里的max和min函数的使用

- max和min函数
  - 最大值和最小值

```
List<Student> list = Arrays.asList(new Student(32),new
Student(33),new Student(21),new Student(29),new Student(18));

//list.stream().max(Comparator.comparingInt(Student::getAge));

//最大
Optional<Student> optional = list.stream().max((s1, s2)-
>Integer.compare(s1.getAge(),s2.getAge()));

//最小
Optional<Student> optional = list.stream().min((s1, s2)-
>Integer.compare(s1.getAge(),s2.getAge()));

System.out.println(optional.get().getAge());
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第6章 Java高级核心之玩转 JDK8 集合框架进阶

第1集 Java新特性玩转JDK8之并行流parallelStream

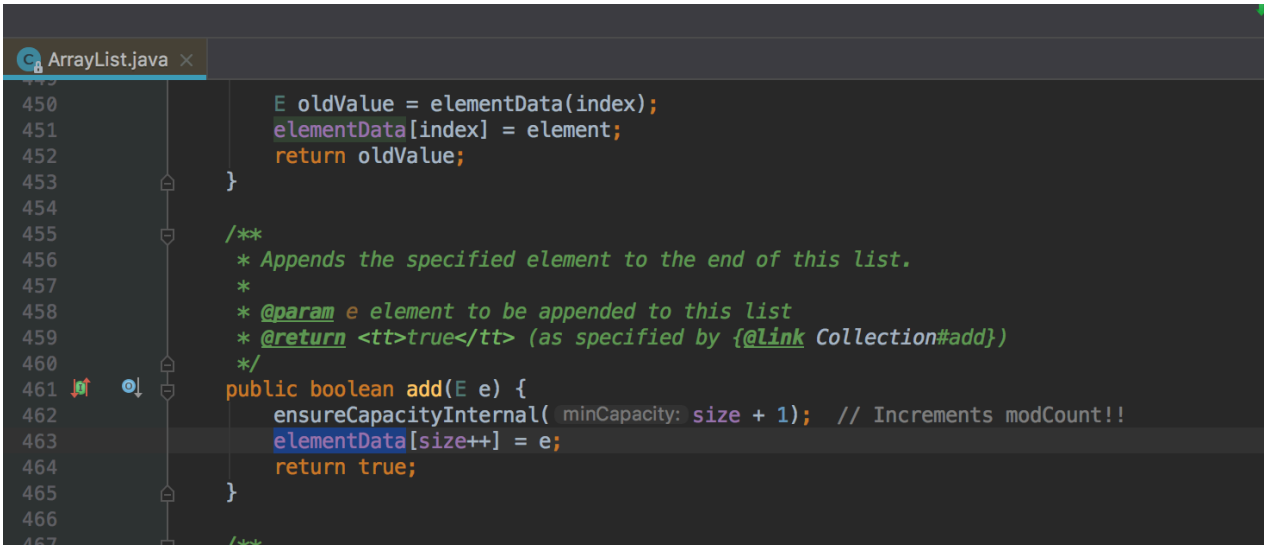
简介：讲解jdk8里面的并行流parallelStream

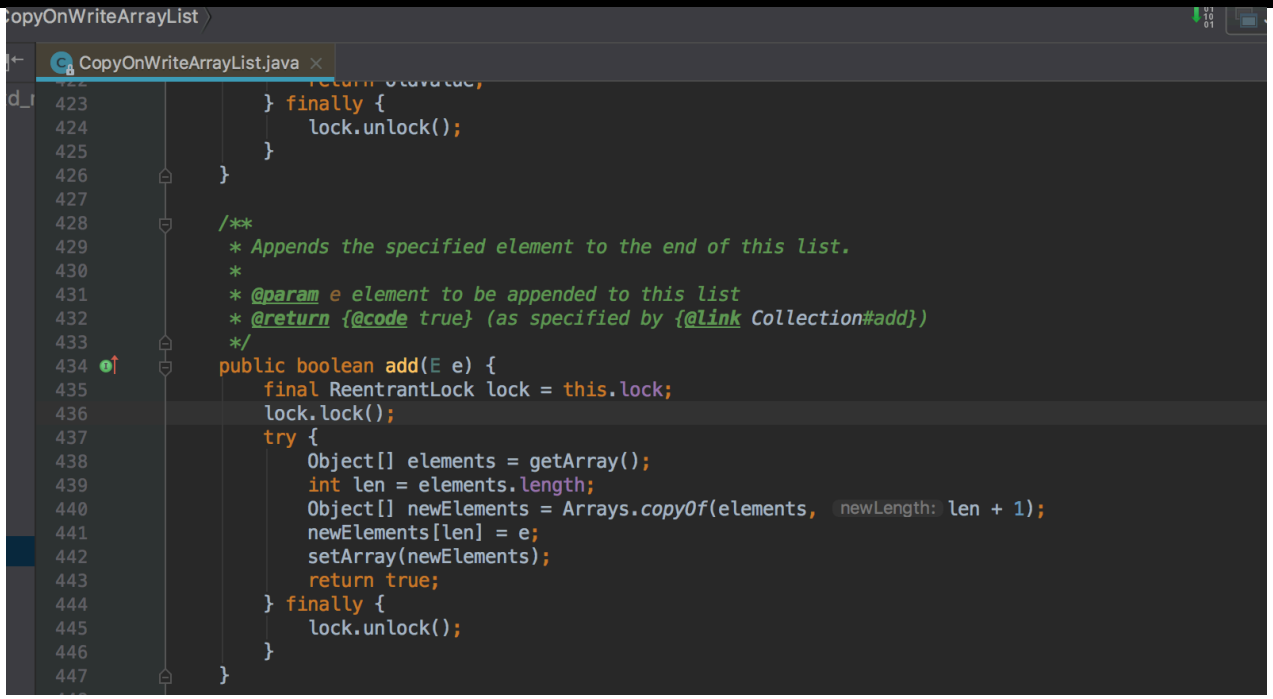
- 为什么会有这个并行流
  - 集合做重复的操作，如果使用串行执行会相当耗时，因此一般会采用多线程来加快， Java8的paralleStream用fork/join框架提供了并发执行能力
  - 底层原理
    - 线程池(ForkJoinPool)维护一个线程队列
    - 可以分割任务，将父任务拆分成子任务，完全贴合分治思想
- 两个区别

```
//顺序输出
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
numbers.stream().forEach(System.out::println);

//并行乱序输出
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
numbers.parallelStream().forEach(System.out::println);
```

- 问题
  - paralleStream并行是否一定比Stream串行快?
    - 错误，数据量少的情况，可能串行更快，ForkJoin会耗性能
  - 多数情况下并行比串行快，是否可以都用并行
    - 不行，部分情况会有线程安全问题，parallelStream里面使用的外部变量，比如集合一定要使用线程安全集合，不然就会引发多线程安全问题

```
for(int i=0;i<10;i++) {
       List list = new ArrayList();
       //List list = new CopyOnWriteArrayList();
       IntStream.range(0, 100).parallel().forEach(list::add);
       System.out.println(list.size());
    }
```





第2集 Java新特性玩转JDK8之reduce操作

简介：讲解jdk8里面的reduce操作

- 什么是reduce操作
  - 聚合操作，中文意思是 “减少”
  - 根据一定的规则将Stream中的元素进行计算后返回一个唯一的值
  - 常用方法一

```
Optional<T> reduce(BinaryOperator<T> accumulator);
```

    - accumulator 计算的累加器
    - 例子: 第一个元素相加和第二个元素相加，结果再和第三个元素相加，直到全部相加完成

```
int value = Stream.of(1, 2, 3, 4, 5).reduce((item1, item2) -> item1
+ item2).get();
```

    - 不用lambda的写法

```
int result =  Stream.of(1,2,3,4,5).reduce(new
BinaryOperator<Integer>() {
            @Override
            public Integer apply(Integer item1, Integer item2) {
                return item1 + item2;
            }
        }).get();
```

- 常用方法二

```
T reduce(T identity, BinaryOperator<T> accumulator);
```

  - identity 用户提供一个循环计算的初始值
  - accumulator 计算的累加器
  - 例子： 100作为初始值，然后和第一个元素相加，结果在和第二个元素相加，直到全部相加完成

```
int value = Stream.of(1, 2, 3, 4,5).reduce(100, (sum, item) -> sum +
item);
```

- 练习： 求最大值

```
int value = Stream.of(1645, 234345, 32,
44434,564534,435,34343542,212).reduce( (item1, item2) -> item1 >
item2 ? item1 : item2 ).get();

System.out.println(value);
```

第3集 Java新特性玩转JDK8之集合的foreach

简介：讲解jdk8里面的list的foreach操作

- 集合遍历的方式
  - for循环
  - 迭代器 Iterator

- Jdk8里面的新增接口

```
default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
```

```
List<Student> results = Arrays.asList(new Student(32),new
Student(33),new Student(21),new Student(29),new Student(18));

results.forEach(obj->{
      System.out.println(obj.toString());
});
```

- 注意点
  - 不能修改包含外部的变量的值
  - 不能用break或者return或者continue等关键词结束或者跳过循环

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第7章 Java高级核心之玩转 JDK8 收集器和集合统计

第1集 Java新特性玩转JDK8之collector收集器

简介：讲解jdk8里面的收集器collector

- collect()方法的作用
  - 一个终端操作, 用于对流中的数据进行归集操作，collect方法接受的参数是一个Collector
  - 有两个重载方法，在Stream接口里面

```
 //重载方法一
 <R> R collect(Supplier<R> supplier, BiConsumer<R, ? super T>
accumulator, BiConsumer<R, R>combiner);

  //重载方法二
 <R, A> R collect(Collector<? super T, A, R> collector);
```

- **Collector**的作用
  - 就是收集器，也是一个接口, 它的工具类Collectors提供了很多工厂方法

- **Collectors** 的作用
  - 工具类，提供了很多常见的收集器实现
    - Collectors.toList()

```
public static <T> Collector<T, ?, List<T>> toList() {

return new CollectorImpl<>((Supplier<List<T>>)
ArrayList::new,        List::add,(left, right) -> {
left.addAll(right); return left; }, CH_ID);

}
```

      - ArrayList::new，创建一个ArrayList作为累加器
      - List::add，对流中元素的操作就是直接添加到累加器中
      - reduce操作, 对子任务归集结果addAll，后一个子任务的结果直接全部添加到前一个子任务结果中
      - CH_ID 是一个unmodifiableSet集合
    - Collectors.toMap()
    - Collectors.toSet()
    - Collectors.toCollection()： 用自定义的实现Collection的数据结构收集
      - Collectors.toCollection(LinkedList::new)
      - Collectors.toCollection(CopyOnWriteArrayList::new)
      - Collectors.toCollection(TreeSet::new)

第2集 Java新特性玩转JDK8之joining函数

简介：讲解jdk8里面的收集器joining函数

- 拼接函数 Collectors.joining

```
//3种重载方法
Collectors.joining()
Collectors.joining("param")
Collectors.joining("param1", "param2", "param3")
```

- 其中一个的实现

```
    public static Collector<CharSequence, ?, String> joining() {
        return new CollectorImpl<CharSequence, StringBuilder, String>(
                StringBuilder::new, StringBuilder::append,
                (r1, r2) -> { r1.append(r2); return r1; },
                StringBuilder::toString, CH_NOID);
    }
```

- 说明：
  - 该方法可以将Stream得到一个字符串， joining函数接受三个参数，分别表示 元素之间的连接符、前缀和后缀。

```
String result = Stream.of("springboot", "mysql", "html5",
"css3").collect(Collectors.joining(",", "[", "]"));
```

第3集 Java新特性玩转JDK8之收集器 partitioningBy分组

简介：讲解jdk8里面的收集器partitioningBy分组

- Collectors.partitioningBy 分组，key是boolean类型

```
public static <T>
Collector<T, ?, Map<Boolean, List<T>>> partitioningBy(Predicate<? super
T> predicate) {
    return partitioningBy(predicate, toList());
}
```

- 练习： 根据list里面进行分组，字符串长度大于4的为一组，其他为另外一组

```
List<String> list = Arrays.asList("java", "springboot",
"HTML5","nodejs","CSS3");
Map<Boolean, List<String>> result =
list.stream().collect(partitioningBy(obj -> obj.length() > 4));
```

第4集 Java新特性玩转JDK8之收集器 group by分组

简介：讲解jdk8里面的收集器 group by分组

- 分组 Collectors.groupingBy()

```
public static <T, K> Collector<T, ?, Map<K, List<T>>> groupingBy(Function<?
super T, ? extends K> classifier) { return groupingBy(classifier, toList());
}
```

- 练习：根据学生所在的省份，进行分组

```
List<Student> students = Arrays.asList(new Student("广东", 23), new
Student("广东", 24), new Student("广东", 23),new Student("北京", 22), new
Student("北京", 20), new Student("北京", 20),new Student("海南", 25));


Map<String, List<Student>> listMap =
students.stream().collect(Collectors.groupingBy(obj ->
obj.getProvince()));

        listMap.forEach((key, value) -> {
            System.out.println("========");
            System.out.println(key);
            value.forEach(obj -> {
                System.out.println(obj.getAge());
            });

        });



class Student {
    private String province;
    private int age;

    public String getProvince() {
        return province;
    }

    public void setProvince(String province) {
        this.province = province;
    }

    public int getAge() {
        return age;
```

```
    }

    public void setAge(int age) {
        this.age = age;
    }


    public Student(String province, int age) {
        this.age = age;
        this.province = province;
    }

}
```

第5集 Java新特性玩转JDK8之收集器 group by进阶

简介：讲解jdk8里面的收集器 group by进阶

- 分组统计
  - 聚合函数进行统计查询，分组后统计个数
  - Collectors.counting() 统计元素个数

```
public static <T, K, A, D>  Collector<T, ?, Map<K, D>>
groupingBy(Function<? super T, ? extends K> classifier,Collector<? super
T, A, D> downstream) {
        return groupingBy(classifier, HashMap::new, downstream);
    }
```

- 需求：统计各个省份的人数

```
 List<Student> students = Arrays.asList(new Student("广东", 23), new
Student("广东", 24), new Student("广东", 23),new Student("北京", 22), new
Student("北京", 20), new Student("北京", 20),new Student("海南", 25));


Map<String, Long> listMap =
students.stream().collect(Collectors.groupingBy(Student::getProvince,
Collectors.counting()));

listMap.forEach((key, value) -> {System.out.println(key+"省人数有
"+value);});
```

第6集 Java新特性玩转JDK8之summarizing集合统计

简介：讲解jdk8里面的收集器 group by进阶

- summarizing 统计相关, 下面是summarizingInt的源码

```
public static <T> Collector<T, ?, IntSummaryStatistics>
summarizingInt(ToIntFunction<? super T> mapper) { return new
CollectorImpl<T, IntSummaryStatistics, IntSummaryStatistics>(
                IntSummaryStatistics::new,
                (r, t) -> r.accept(mapper.applyAsInt(t)),
                (l, r) -> { l.combine(r); return l; }, CH_ID);
    }
```

  - 作用：可以一个方法把统计相关的基本上都完成
  - 分类
    - summarizingInt
    - summarizingLong
    - summarizingDouble
- 需求：统计学生的各个年龄信息

```
 List<Student> students = Arrays.asList(new Student("广东", 23), new
Student("广东", 24), new Student("广东", 23),new Student("北京", 22), new
Student("北京", 20), new Student("北京", 20),new Student("海南", 25));

IntSummaryStatistics summaryStatistics =
students.stream().collect(Collectors.summarizingInt(Student::getAge));
System.out.println("平均值：" + summaryStatistics.getAverage());
System.out.println("人数：" + summaryStatistics.getCount());
System.out.println("最大值：" + summaryStatistics.getMax());
System.out.println("最小值：" + summaryStatistics.getMin());
System.out.println("总和：" + summaryStatistics.getSum());


class Student {
    private String province;
    private int age;

    public String getProvince() {
        return province;
    }

    public void setProvince(String province) {
        this.province = province;
    }
```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }


    public Student(String province, int age) {
        this.age = age;
        this.province = province;
    }

}
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第8章 Java高级核心之玩转 JDK8 Collection和Lambda实战

第1集 Collection和Lambda电商数据处理实战需求说明

简介：综合jdk8新特性，collection和lambda完成数据处理需求整理

需求描述：电商订单数据处理，根据下面的list1和list2 各10个订单

- 统计出同时被两个人购买的商品列表(交集)
- 统计出两个人购买商品的差集
- 统计出全部被购买商品的去重并集
- 统计两个人的分别购买订单的平均价格

- 统计两个人的分别购买订单的总价格

```
//总价 35
        List<VideoOrder> videoOrders1 = Arrays.asList(
            new VideoOrder("20190242812", "springboot教程", 3),
            new VideoOrder("20194350812", "微服务SpringCloud", 5),
            new VideoOrder("20190814232", "Redis教程", 9),
            new VideoOrder("20190523812", "网页开发教程", 9),
            new VideoOrder("201932324", "百万并发实战Netty", 9));

        //总价 54
        List<VideoOrder> videoOrders2 = Arrays.asList(
            new VideoOrder("2019024285312", "springboot教程", 3),
            new VideoOrder("2019081453232", "Redis教程", 9),
            new VideoOrder("20190522338312", "网页开发教程", 9),
            new VideoOrder("2019435230812", "Jmeter压力测试", 5),
            new VideoOrder("2019323542411", "Git+Jenkins持续集成", 7),
            new VideoOrder("2019323542424", "Idea全套教程", 21));
```

```
public class VideoOrder {

    private String tradeNo;

    private int money;

    private String title;

    public VideoOrder(String tradeNo,String title, int money ){
        this.tradeNo = tradeNo;
        this.title = title;
        this.money = money;
    }
    public String getTradeNo() {
        return tradeNo;
    }

    public void setTradeNo(String tradeNo) {
        this.tradeNo = tradeNo;
    }

    public int getMoney() {
        return money;
    }

    public void setMoney(int money) {
        this.money = money;
    }
```

```
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

}
```

第2集 JDK8新特性处理电商订单数据之答案讲解上集

简介：讲解使用jdk8新特性完成订单需求一、二、三

```
    @Override
    public boolean equals(Object obj) {
        if(obj instanceof VideoOrder) {
            VideoOrder o1 = (VideoOrder)obj;
            return title.equals(o1.getTitle());
        }
        return super.equals(obj);
    }

    @Override
    public String toString() {
        return "VideoOrder{" +
            "money=" + money +
            ", title='" + title + '\'' +
            '}';
    }

    @Override
    public int hashCode() {
        return title.hashCode();
    }
```

```
 //交集
        List<VideoOrder> intersectionList =
videoOrders1.stream().filter(videoOrders2::contains).collect(
            Collectors.toList());

        System.out.println("交集="+intersectionList);

        //差集1
```

```
        List<VideoOrder> diffList1 = videoOrders1.stream().filter(obj->!videoOrders2.contains(obj)).collect(Collectors.toList());
        System.out.println("差集1="+diffList1);
        //差集2
        List<VideoOrder> diffList2 = videoOrders2.stream().filter(obj->!videoOrders1.contains(obj)).collect(Collectors.toList());
        System.out.println("差集2="+diffList2);

        //并集
        List<VideoOrder> allVideoOrder = videoOrders1.parallelStream().collect(Collectors.toList());
        allVideoOrder.addAll(videoOrders2);
        System.out.println("并集 = "+allVideoOrder);

        //去重并集
        List<VideoOrder> allVideoOrderDistinct = allVideoOrder.stream().distinct().collect(Collectors.toList());
        System.out.println("去重并集 = "+allVideoOrderDistinct);
```

第3集 JDK8新特性处理电商订单数据之答案讲解下集

简介：讲解使用jdk8新特性完成订单需求四、五

```
        //两个订单平均价格
        double videoOrderAvg1 = videoOrders1.stream().collect(Collectors.averagingInt(VideoOrder::getMoney)).doubleValue();
        System.out.println("订单列表1平均价格="+videoOrderAvg1);

        double videoOrderAvg2 = videoOrders2.stream().collect(Collectors.averagingInt(VideoOrder::getMoney)).doubleValue();
        System.out.println("订单列表2平均价格="+videoOrderAvg2);


        //订单总价
        int totalMoney1 = videoOrders1.stream().collect(Collectors.summingInt(VideoOrder::getMoney)).intValue();
        int totalMoney2 = videoOrders2.stream().collect(Collectors.summingInt(VideoOrder::getMoney)).intValue();
        System.out.println("订单列表1总价="+totalMoney1);
        System.out.println("订单列表2总价="+totalMoney2);
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第9章 Java高级核心之玩转 JDK8 新的内存空间和异常处理

第1集 JDK8新特性之新内存空间Matespace

简介：讲解JDK8里面的新的内存空间MetaSpace

- JVM 种类有很多，比如 Oralce-Sun Hotspot, Oralce JRockit, IBM J9, Taobao JVM，我们讲的是Hotspot才有，JRockit以及J9是没有这个区域
- JVM内存知识 在JDK8之前的HotSpot JVM，有个区域叫做"永久代(permanent generation)， 通过在命令行设置参数-XX:MaxPermSize来设定永久代最大可分配的内存空间

  如果JDK8里面设置了PermSize 和 MaxPermSize 会被忽略并给出警告
- 作用：该块内存主要是被JVM用来存放 class 和 mate 信息的，当 class 被加载 loader 的时候就会被存储到该内存区中，如方法的编译信息及字节码、常量池和符号解析、类的层级信息，字段，名字等
- 有大项目经验的同学对下面这个异常应该熟悉 java.lang.OutOfMemoryError: PermGen space

  原因是: 永久代空间不够，类太多导致

- jdk8的修改 JDK8 HotSpot JVM 使用本地内存来存储类元数据信息，叫做 元空间（Metaspace）在默认情况下Metaspace的大小只与本地内存大小有关

  常用的两个参数 -XX:MetaspaceSize=N 指Metaspace扩容时触发FullGC的初始化阈值

  -XX:MaxMetaspaceSize=N 指用于限制Metaspace增长的上限，防止因为某些情况导致Metaspace无限的使用本地内存

  不管两个参数如何设置，都会从20.8M开始，然后随着类加载越来越多不断扩容调整直到最大
- 查看大小 jstat -gc pid MC: current metaspace capacity MU: mateaspace utilization 单位是KB

第2集 JDK7新特性之try-with-resources

简介：讲解JDK7里面的try-with-resources资源处理

- 什么是try-with-resources

  资源的关闭很多人停留在旧的流程上，jdk7新特性就有，但是很多人以为是jdk8的 在try(...)里声明的资源，会在try-catch代码块结束后自动关闭掉

旧的

```
public static void main(String[] args) throws IOException {
        String path = "/Users/jack/Desktop/t.txt";
        test(path);

    }

    private static void test(String filepath) throws FileNotFoundException {
        OutputStream out = new FileOutputStream(filepath);
        try {
            out.write((filepath+"可以学习java架构课程").getBytes());
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
```

新的

```
 private static void test(String filepath){
       try(OutputStream out = new FileOutputStream(filepath);) {
           out.write((filepath+"可以学习java架构课程").getBytes());
       } catch (Exception e) {
           e.printStackTrace();
       }

    }
```

- 注意点 1、实现了AutoCloseable接口的类，在try()里声明该类实例的时候，try结束后自动调用的close方法，这个动作会早于finally里调用的方法

2、不管是否出现异常，try()里的实例都会被调用close方法

3、try里面可以声明多个自动关闭的对象，越早声明的对象，会越晚被close掉

愿景："让编程不在难学，让技术与生活更加有趣"

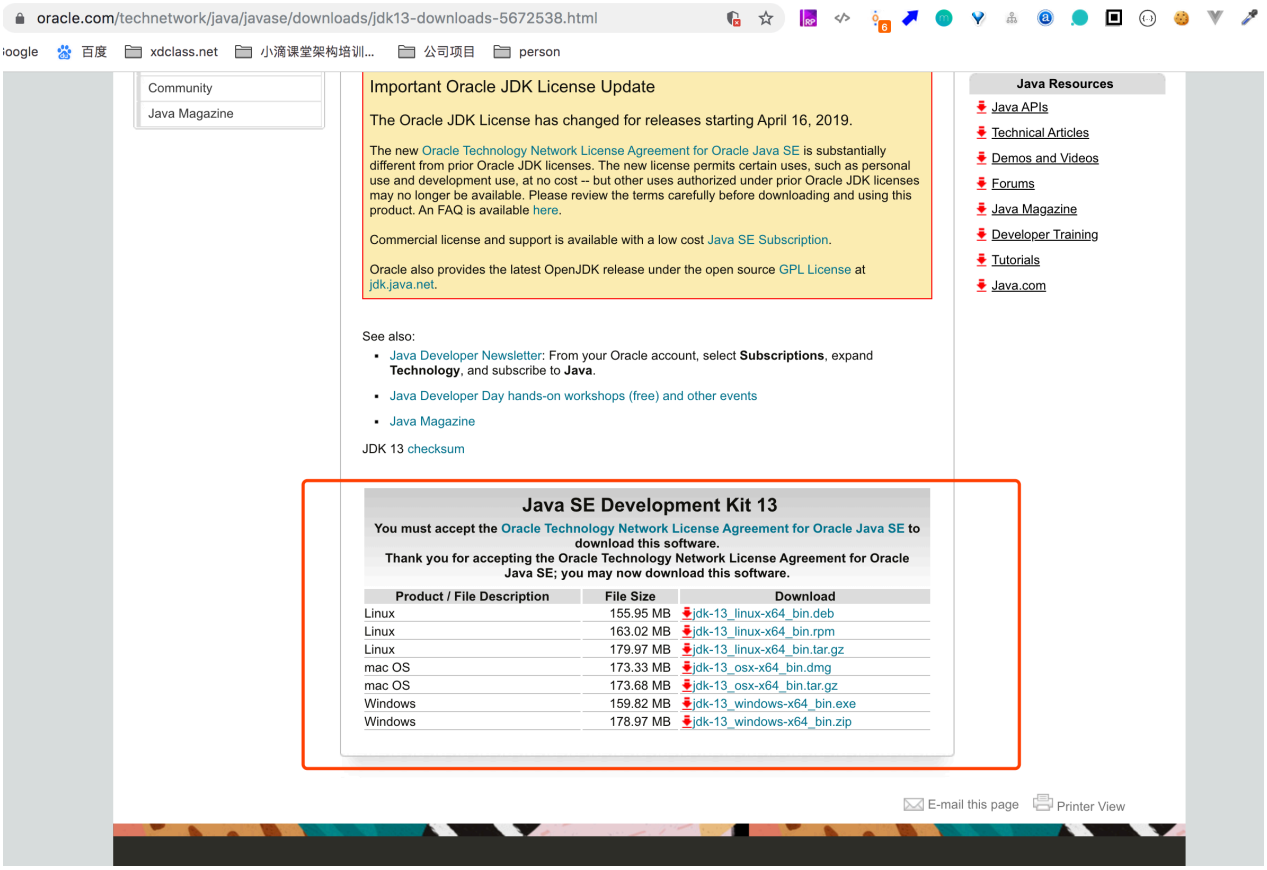
更多架构课程请访问官网： xdclass.net

第10章 Java高级核心之玩转JDK9常见特性讲解

第1集 新版JDK13下载和本章课程说明

简介：讲解JDK13版本的下载和后续课程说明

- 为什么用JDK13
  - 版本太多，直接用最新的进行安装讲解
- JDK 13的官方地址（如果地址过期，则百度搜索，官方经常改动） https://www.oracle.com/technetwork/java/javase/downloads/index.html



- 环境变量配置

```
解压到指定目录
vim .bash_profile

JAVA_HOME="/Users/xdclass/Documents/software/jdk13/Contents/Home"
```

- 相关环境变量配置
  - idea配置各个版本jdK
    - https://blog.csdn.net/blueboz/article/details/81270242
  - window7配置环境变量
    - https://www.cnblogs.com/sunzhentian/p/11577154.html
  - window10配置环境变量
    - https://www.cnblogs.com/huainanhai/p/11592729.html
    - https://www.cnblogs.com/rever/p/7792826.html

- JDK9~13新增特性很多但不会全部都讲解
  - 会讲解： 常见的有利于提高开发效率，常见API增强
  - 不讲解：不常用的API, 或者只是试验性的功能, 如AOT静态编译

第2集 java高级核心之JDK9常用Jshell实战

简介：讲解jdk9新增测试工具jshell实战

- 什么是jshell
  - 从java9开始，jdk引入了交互式 REPL（Read-Eval-Print-Loop，读取-求值-输出-循环）
  - 官方文档
    - https://docs.oracle.com/en/java/javase/12/jshell/introduction-jshell.html#GUID-630F27C8-1195-4989-9F6B-2C51D46F52C8
- 常用命令
  - 帮助命令
    - /help
    - /help intro
  - 列出输入的源
    - /list
  - 编辑某个源
    - /edit
  - 删除
    - /drop
  - 退出jshell命令

        - /exit
    - 重置
        - /reset
    - 查看历史编辑
        - /history
- 自动化补齐功能
    - Tab键

第3集 接口方法进阶之JDK9私有方法

简介：讲解jdk9新增的接口私有方法

- 什么是jdk9新增的接口私有方法
    - JDK8新增了静态方法和默认方法，但是不支持私有方法
    - jdk9中新增了私有方法

```
public interface OrderPay {
    void pay();

    default void defaultPay(){
        privateMethod();
    }

    //接口的私有方法可以在JDK9中使用
    private void privateMethod(){
        System.out.println("调用接口的私有方法");
    }
}

public class OrderPayImpl implements OrderPay {
    @Override
    public void pay() {
        System.out.println("我实现了接口");
    }
}

    public static void main(String[] args) throws Exception {

        OrderPay orderPay = new OrderPayImpl();
        orderPay.defaultPay();
        orderPay.pay();

    }
```

- 注意点（面试题！！！）：
  - 接口中的静态方法不能被实现类继承和子接口继承，但是接口中的非静态的默认方法可以被实现类继承
  - 例如List.of() 方法，ArrayList虽然继承了List，但是不能用ArrayList.of()方法
  - 类的静态方法可以被继承

第4集 JDK9新特性之增强try-with-resource

简介：讲解jdk9新增的接口私有方法

- 什么是try-with-resource
  - 在JDK7中，新增了try-with-resources语句，可以在try后的括号中初始化资源，可以实现资源自动关闭

```
OutputStream out = new FileOutputStream(filepath);
try(OutputStream temp = out;) {
            temp.write((filepath+"可以学习java架构课程").getBytes());
        }catch (Exception e){
            e.printStackTrace();
        }
```

- 什么是增强try-with-resource
  - 在JDK9中，改进了try-with-resources语句，在try外进行初始化，在括号内引用，即可实现资源自动关闭，多个变量则用分号进行分割
  - 不需要声明资源 out 就可以使用它，并得到相同的结果

```
 public static void main(String[] args) throws Exception {

        String path = "/Users/xdclass/Desktop/t.txt";
        test(path);
    }

    private static void test(String filepath) throws FileNotFoundException {
        OutputStream out = new FileOutputStream(filepath);
        try (out) {
            out.write((filepath + "可以学习java架构课程").getBytes());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

愿景："让编程不在难学，让技术与生活更加有趣"

第11章 Java高级核心之玩转JDK9的Stream和集合API

第1集 JDK9之快速创建只读集合

简介：讲解jdk9里面新增的集合容器API，快速创建只读集合

- 什么是只读集合
  - 集合只能读取，不能增加或者删除里面的元素

- JDK9之前创建只读集合

```
        List<String> list = new ArrayList<>();
        list.add("SpringBoot课程");
        list.add("架构课程");
        list.add("微服务SpringCloud课程");
        //设置为只读List集合
        list = Collections.unmodifiableList(list);
        System.out.println(list);
        Set<String> set = new HashSet<>();
        set.add("Mysql教程");
        set.add("Linux服务器教程");
        set.add("Git教程");
        //设置为只读Set集合
        set = Collections.unmodifiableSet(set);
        System.out.println(set);

        Map<String, String> map = new HashMap<>();
        map.put("key1", "课程1");
        map.put("key2", "课程2");
        //设置为只读Map集合
        map = Collections.unmodifiableMap(map);
        System.out.println(map);
```

- JDK9后创建只读集合
  - 查看of()源码

```
List<String> list = List.of("SpringBoot课程", "架构课程", "微服务SpringCloud课
程");
System.out.println(list);
Set<String> set = Set.of("Mysql教程", "Linux服务器教程", "Git教程");
System.out.println(set);
Map<String, String> map = Map.of("key1", "课程1", "key2", "课程2");
System.out.println(map);
```

第2集 JDK9之新增Stream API讲解

简介：讲解jdk9里面新增的Stream API

- JDK8里面新增的Stream流式编程，方便了很多数据的处理
  - jdk9里面新增了部分API

- takeWhile
  - 有序的集合：从 Stream 中获取一部分数据, 返回从头开始的尽可能多的元素, 直到遇到第一个false结果，如果第一个值不满足断言条件，将返回一个空的 Stream

```
List<String> list =
List.of("springboot","java","html","","git").stream().takeWhile(obj-
>!obj.isEmpty()).collect(Collectors.toList());


//无序集合，返回元素不固定，暂无无实际使用场景
Set<String> set =
Set.of("springboot","java","html","","git").stream().takeWhile(obj-
>!obj.isEmpty()).collect(Collectors.toList());
```

- dropWhile
  - 与 takeWhile相反，返回剩余的元素，和takeWhile方法形成互补

```
List<String> list =
List.of("springboot","java","html","","git").stream().dropWhile(obj-
>!obj.isEmpty()).collect(Collectors.toList());
```

  - 无序Stream里面也无实际使用场景

- bug ,计划在jdk10里面进行修复
  - https://bugs.openjdk.java.net/browse/JDK-8193856

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第12章 Java高级核心之玩转JDK10和JDK11常见特性

第1集 JDK10之局部变量类型推断var讲解

简介：讲解JDK10新增局部变量类型推断var

- Java是一种强类型, 许多流行的编程语言都已经支持局部变量类型推断，如js，Python，C++等
- JDK10 可以使用var作为局部变量类型推断标识符
  - Local-Variable Type Inference（局部变量类型推断），顾名思义只能用做为局部变量
  - 注意
    - 仅适用于局部变量，如 增强for循环的索引，传统for循环局部变量
    - 不能使用于方法形参、构造函数形参、方法返回类型或任何其他类型的变量声明
    - 标识符var不是关键字，而是一个保留类型名称，而且不支持类或接口叫var,也不符合命名规范
    - 可以用jshell试验或者IDEA

```
//根据推断为 字符串类型
var strVar = "springboot";
System.out.println(strVar instanceof String);

//根据10L 推断long 类型
var longVar = 10L;

//根据 true推断 boolean 类型
var flag = true;
//var flag = Boolean.valueOf("true");
//System.out.println(flag instanceof Boolean);

// 推断 ArrayList<String>
var listVar = new ArrayList<String>();
System.out.println(listVar instanceof ArrayList);

// 推断 Stream<String>
var streamVar = Stream.of("aa", "bb", "cc");
System.out.println(streamVar instanceof Stream);

if(flag){
    System.out.println("这个是 flag 变量，值为true");
}

for (var i = 0; i < 10; i++) {
    System.out.println(i);
}

try (var input = new FileInputStream("validation.txt")) {

}
```

第2集 JDK11之新增HttpClient客户端快速入门

简介：讲解JDK11新增Http客户端

- 这个功能在JDK 9中引入并在JDK 10中得到了更新
- 最终JDK11正式发布，支持 HTT/1.1, HTTP/2， （JDK10相关课程里面未讲解该知识点）
- 官方文档 http://openjdk.java.net/jeps/321
- 常用类和接口讲解
  - HttpClient.Builder
    - HttpClient 构建工具类
  - HttpRequest.Builder
    - HttpRequest 构建工具类
  - HttpRequest.BodyPublisher
    - 将java 对象转换为可发送的HTTP request body字节流, 如form表单提交
  - HttpResponse.BodyHandler
    - 处理接收到的 Response Body

- 创建HttpClient, 下面结果是一致的底层调用

```
  //var httpClient = HttpClient.newBuilder().build();
    var httpClient = HttpClient.newHttpClient();
```

- 创建get请求

```
//private static final String targetUrl =
"http://api.xdclass.net/pub/api/v1/web/all_category";
 private static final URI uri = URI.create(targetUrl);

    //GET请求
    private static void testGet() {
        //var httpClient = HttpClient.newHttpClient();
        //设置建立连接超时 connect timeout
        var httpClient =
HttpClient.newBuilder().connectTimeout(Duration.ofMillis(5000)).build();


        //设置读取数据超时 read timeout
    var request =
HttpRequest.newBuilder().timeout(Duration.ofMillis(3000))
                .header("key1", "v1")
                .header("key2", "v2")
                .uri(uri).build();
        try {
```

```
            var response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
            System.out.println(response.body());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

第3集 JDK11之标准HttpClient提交Post和异步请求

简介：讲解JDK11新增Http客户端提交post请求和异步请求

- 使用HttpClient提交Post请求

```
private static final String targetUrl
="https://api.xdclass.net/pub/api/v1/web/web_login";
private static final URI uri = URI.create(targetUrl);

//POST请求
    private static void testPost() {
        var httpClient = HttpClient.newHttpClient();
        var request = HttpRequest.newBuilder()
                .uri(uri)

                //json格式则使用下面数据
                //.header("Content-Type", "application/json")
                //.POST(HttpRequest.BodyPublishers.ofString("
{\"phone\":\"13113777337\",\"pwd\":\"1234567890\"}"))

                //form表单则使用下面配置
                .header("Content-Type", "application/x-www-form-
urlencoded")

.POST(HttpRequest.BodyPublishers.ofString("phone=13113777337&pwd=1234567
890"))
                .build();

        try {
            var response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
            System.out.println(response.body());
        } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
```

- 使用HttpClient提交异步GET请求

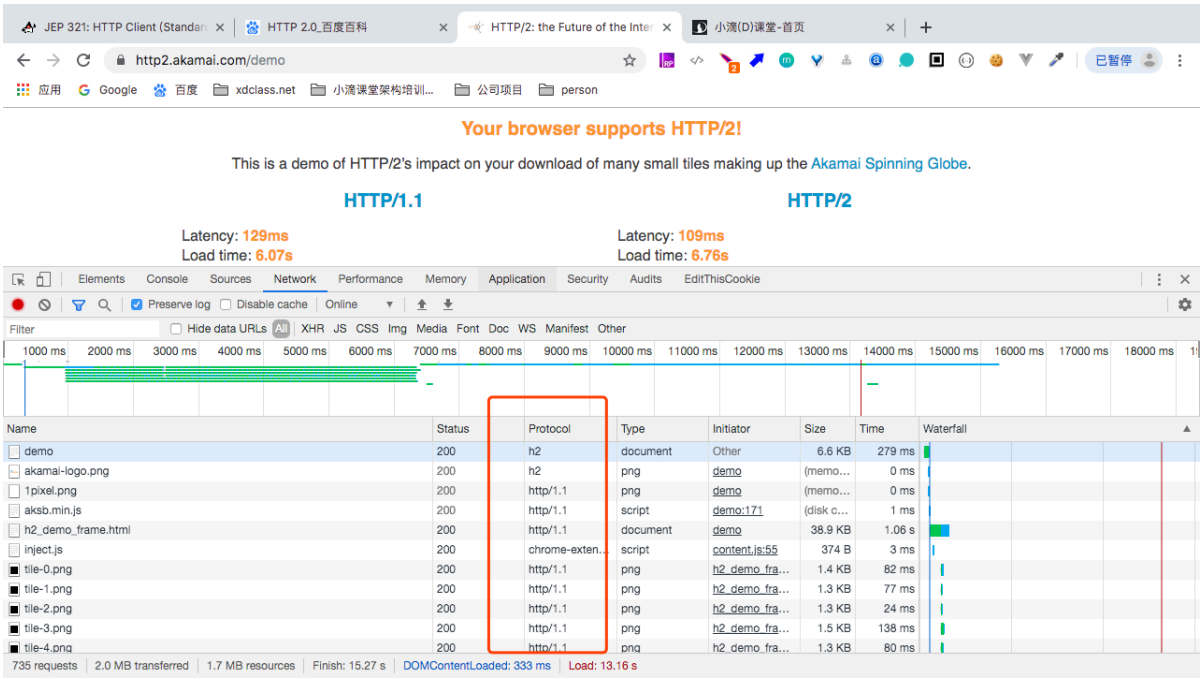
```
 //异步GET请求
 //private static final String targetUrl =
"http://api.xdclass.net/pub/api/v1/web/all_category";
 private static final URI uri = URI.create(targetUrl);

 //异步请求通过CompletableFuture实现。
    private static void testAsynGet() {
        var httpClient = HttpClient.newBuilder().build();

        var request =
HttpRequest.newBuilder().timeout(Duration.ofMillis(3000))
                .header("key1", "v1")
                .header("key2", "v2")
                .uri(uri).build();
        try {
//CompletableFuture<String> result = httpClient.sendAsync(request,
HttpResponse.BodyHandlers.ofString()).thenApply(HttpResponse::body);

var result = httpClient.sendAsync(request,
HttpResponse.BodyHandlers.ofString())
                    .thenApply(HttpResponse::body);

            System.out.println(result.get());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

第4集 JDK11之标准HttpClient提交Http2请求

简介：讲解JDK11新增Http客户端提交http2请求

- HTTP2 百科
    - https://baike.baidu.com/item/HTTP%202.0/12520156?fr=aladdin

- HTTP2协议的强制要求https，如果目标URI是HTTP的，则无法使用HTTP 2协议
- 如何判断网站是否是http2协议，浏览器，network面板，选择protocol
-




```
private static final String targetUrl = "https://http2.akamai.com/demo";
private static final URI uri = URI.create(targetUrl);

private static void testHttp2() {
        var httpClient = HttpClient.newBuilder()
                .connectTimeout(Duration.ofMillis(3000))
                .version(HttpClient.Version.HTTP_2)
                .build();

        var request = HttpRequest.newBuilder()
                .timeout(Duration.ofMillis(3000))
                .header("key1", "v1")
                .header("key2", "v2")
                .uri(uri)
                .build();
        try {
            var response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
            System.out.println(response.body());
            System.out.println(response.version());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

第5集 JDK11之javac和java命令优化

简介：讲解java编译运行命令在jdk11中的改善

- jdk11前运行java程序
  - 编译
    - javac xxx.java
  - 运行
    - java xxx
- jdk11后运行java程序(本地不会生成class文件)
  - java xxxx.java

```
public class Main {

    public static void main(String[] args) throws Exception {
        String text = "xdclass.net";
        //String类新增repeat(int num) 方法，用于字符串循环输出
        System.out.println(text.repeat(2));
    }

}
```

愿景："让编程不在难学，让技术与生活更加有趣"

更多架构课程请访问官网： xdclass.net

第13章 Java高级核心之大话JDK12和JDK13

第1集 大话JDK各个版本常见问题讲解

简介：讲解JDK一些基础知识科普

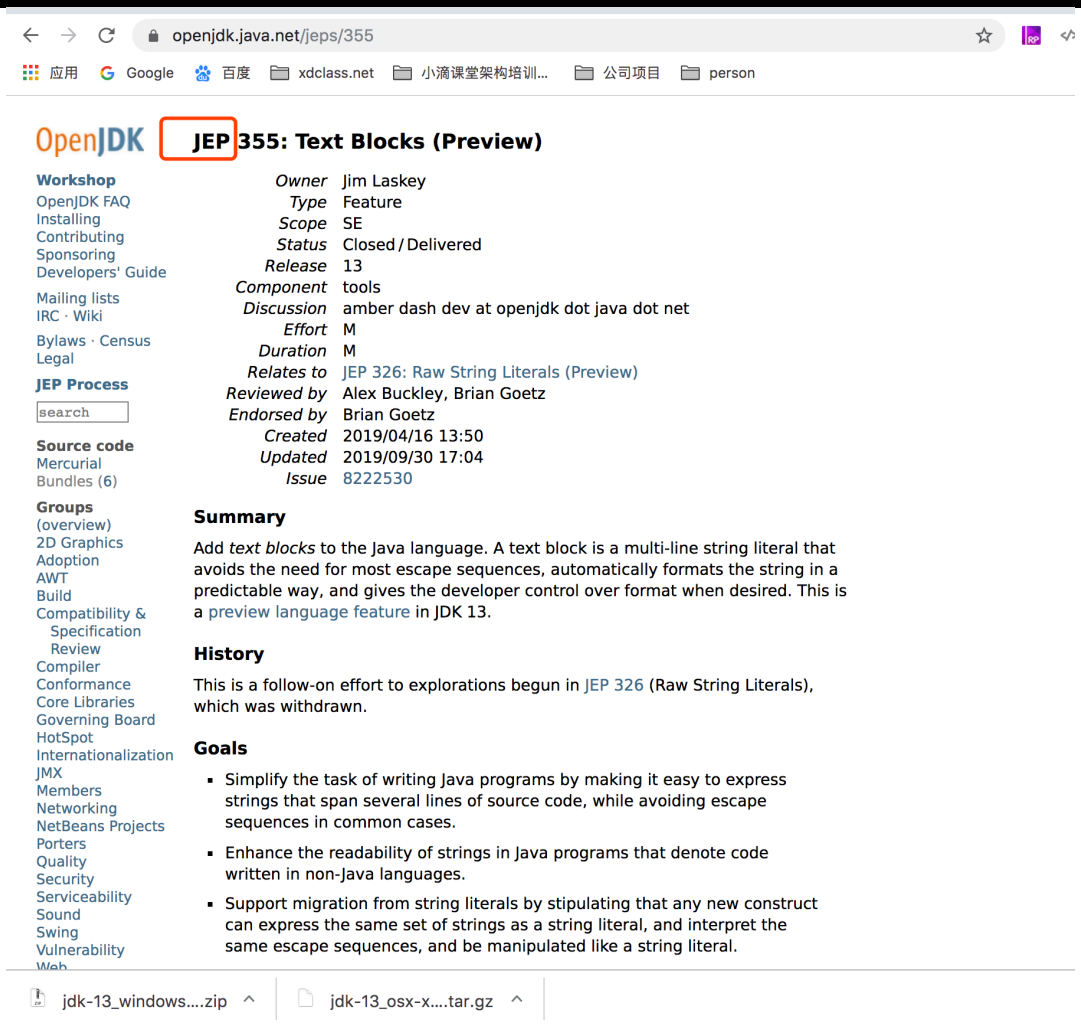
- OpenJDK和OracleJDK版本区别

    - OpenJDK是JDK的开放源码版本，以GPL协议的形式发布（General Public License）
    - Oracle JDK采用了商业实现
- LTS 是啥意思?
    - Long Term Support 长期支持的版本，如JDK8、JDK11都是属于LTS
    - JDK9 和 JDK10 这两个被称为“功能性的版本”不同, 两者均只提供半年的技术支持
    - 甲骨文释出Java的政策，每6个月会有一个版本的释出，长期支持版本每三年发布一次，根据后续的发布计划，下一个长期支持版 Java 17 将于2021年发布
- 8u20、11u20是啥意思?
    - 就是Java的补丁，比如JDK8的 8u20版本、8u60版本; java11的 11u20、11u40版本
- JDK要收费了？？？？
    - 问题的产生由来
        - Oracle 宣布 Java8 在 2019 年 1月之后停止更新，另外 Java11 及以后版本将不再提供免费的 long-term support (LTS) 支持，猜测未来将有越来越多 Java 开发者转向使用 OpenJDK
            - OpenJDK是免费的，想要不断体验新特性的developer来说是不错的选择
            - OracleJDK不是免费的，对于企业用户来说，有钱的情况下就选择OracleJDK
- 对应oracleJDK ,我们可以自己用来写代码，调试，学习即可

第2集 玩转JDK13新特性之多行文本块

简介：讲解JDk13里面新增的多行文本块

- JDK13发布，主要更新点
    - https://openjdk.java.net/projects/jdk/13/
    - https://www.oracle.com/technetwork/java/javase/13-relnote-issues-5460548.html#NewFeature
    - JEP全称： JDK Enhancement Proposal 特性增强提议



- 多行文本块
  - 旧：在java代码里面编写多行源码带有特殊字符则需要转义，如HTML，sql等
  - 新：原生字符串文字（raw string literals），它可以跨多行源码而不对转义字符进行转义

旧：

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";


String query = "SELECT `EMP_ID`, `LAST_NAME` FROM `EMPLOYEE_TB`\n" +
               "WHERE `CITY` = 'INDIANAPOLIS'\n" +
               "ORDER BY `EMP_ID`, `LAST_NAME`;\n";
```

新：

```
String html = """
              <html>
                  <body>
                      <p>Hello, world</p>
```

```
                    </body>
                </html>
                """;


String query = """
                SELECT `EMP_ID`, `LAST_NAME` FROM `EMPLOYEE_TB`
                WHERE `CITY` = 'INDIANAPOLIS'
                ORDER BY `EMP_ID`, `LAST_NAME`;
                """;
```

对比一

```
"""
line 1
line 2
line 3
"""
```

等效于字符串文字：

```
"line 1\nline 2\nline 3\n"
```

对比二

```
"""
line 1
line 2
line 3"""
```

等效于字符串文字：

```
"line 1\nline 2\nline 3"
```

对比三

文本块可以表示空字符串，但不建议这样做，因为它需要两行代码：

```
String empty = """
""";
```

注意：错误例子

```
String a = """""";   // no line terminator after opening delimiter
String b = """ """;  // no line terminator after opening delimiter
String c = """
           ";
```

- 开启新版支持 jshell --enable-preview

第3集 玩转JDK13新特性之增强switch表达式

简介：讲解JDk13里面优化了switch表达式

- 旧：没有break，则匹配的case后面会一直输出， value类型 可以是 byte、short、int 、char、String 类型

```
public void testOldSwitch1(){
    int i = 1;
        switch(i){
           case 0:
              System.out.println("zero");
           case 1:
              System.out.println("one");
           case 2:
              System.out.println("two");
           default:
              System.out.println("default");
        }

}

public void testOldSwitch2(int i){
      switch(i){
         case 0:
            System.out.println("zero");
            break;
         case 1:
            System.out.println("one");
            break;
         case 2:
            System.out.println("two");
            break;
         default:
            System.out.println("default");
      }

}
```

- 新：使用箭头函数，不用声明break,会自动终止，支持多个值匹配,使用逗号分隔

```
public void testNewSwitch(int i){
    switch(i){
        case 0 -> {
        System.out.println("zero");
        System.out.println("这是多行语句");
        }
        case 1,11,111 -> System.out.println("one");
        case 2 -> System.out.println("two");
        default -> System.out.println("default");
    }
}
```

更多架构课程请访问官网： xdclass.net

第14章 Java高级核心之JDK8~13课程总结

第1集 玩转JDK8~13新特性课程总结和学习路线规划

简介：JDK8~13课程总结和学习路线规划

- 回顾课程大纲
  - 多关注LTS版本如JDK8,JDK11,JDK17(未发布)，这个是重点需要学的；功能性的则挑选常用的特性进行掌握即可；
  - 预览性的可以先不学或者简单知道就行，例如 AOT，ZGC,CDS 这些，未来的LTS版本中会更新，所以可以关注未来的LTS版本
- 学习建议
  - 看视频+记笔记+加实操+官方文档

- 后端主流技术栈学习
  - 已有的路线图
  - 高级工程师路线图

小滴课堂，愿景：让编程不在难学，让技术与生活更加有趣

相信我们，这个是可以让你学习更加轻松的平台，里面的课程绝对会让你技术不断提升

欢迎加小D讲师的微信： jack794666918

我们官方网站：https://xdclass.net

千人IT技术交流QQ群： 718617859

重点来啦：免费赠送你干货文档大集合，包含前端，后端，测试，大数据，运维主流技术文档（持续更新）

https://mp.weixin.qq.com/s/qYnjcDYGFDQorWmSfE7lpQ