极客大学

极客大学 Java 进阶训练营
第 11 课
Java相关框架（3）

KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

极客大学

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

极客大学

目录

1. Java8 Lambda*

什么是 Lambda 表达式

Lambda 表达式（lambda expression）是一个匿名函数，Lambda 表达式基于数学中的 λ 演算得名，直接对应于其中的 lambda 抽象（lambda abstraction），是一个匿名函数，即没有函数名的函数。

从动态引用到动态定义，简化写法

Java Lambda 表达式

面向对象与面向函数。

Java 里，函数不是第一等公民，需要封装到接口里。

从而 Java Lambda 表达式 --> 内部匿名类。

方法签名。

两种函数。

只有一行时可以省略大括号

(parameters) -> expression

或

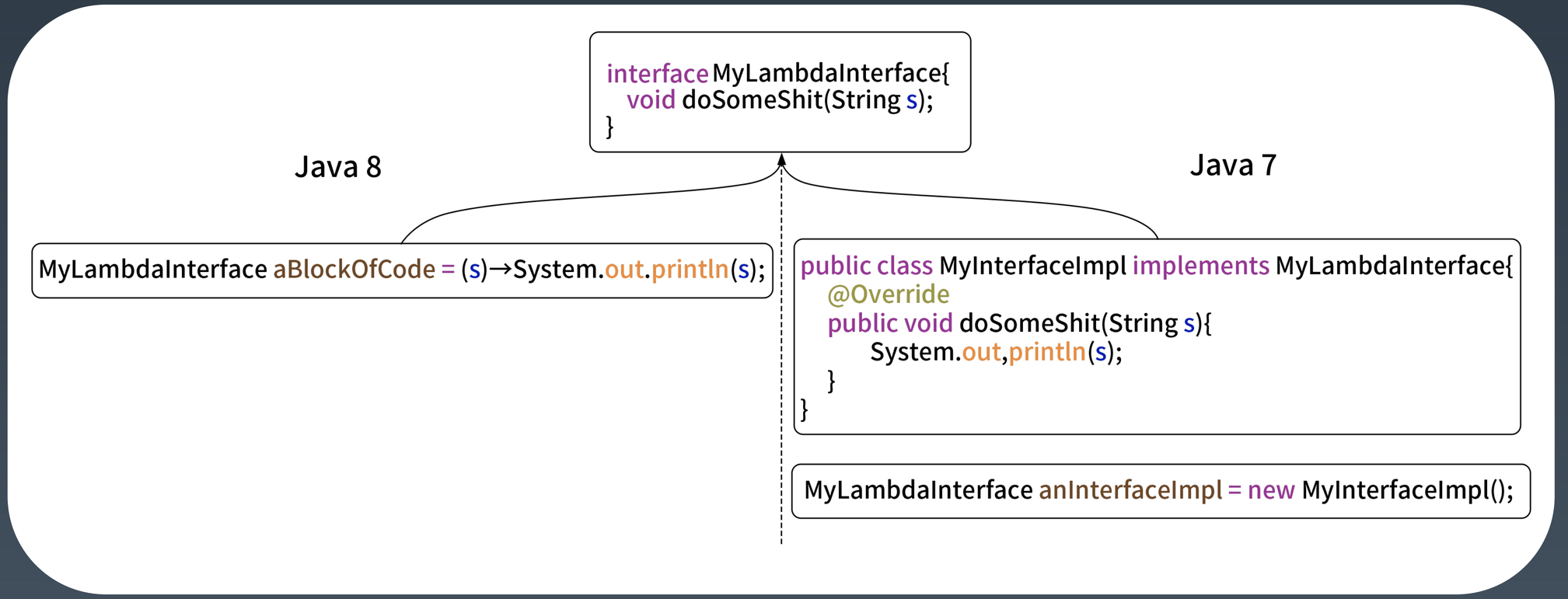
(parameters) ->{ statements; }

Java Lambda表达式

```
// 1. 不需要参数,返回值为 5
() -> 5

// 2. 接收一个参数(数字类型),返回其2倍的值
x -> 2 * x

// 3. 接受2个参数(数字),并返回他们的差值
(x, y) -> x - y

// 4. 接收2个int型整数,返回他们的和
(int x, int y) -> x + y

// 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)
(String s) -> System.out.print(s)
```

Java Lambda 表达式

代码实例



Java Lambda 表达式

极客大学

代码实例

```
public static void enact (MyLamadaInterface myLambda, String s){
    myLambda.doSomeShit(s);
}
```

Java 8

直接把Lambda表达式传给enact()

```
enact(s →System.out.println(s), “Hello World!”);
```

Java 7

需要先定义一个class实现接口，再把class instance传给enact()

```
public class MyInterfaceImpl implements MyLambdaInterface{
   @Override
   public void doSomeShit(String s){
        System.out,println(s);
   }
}
```

```
MyLambdaInterface anInterfaceImpl = new MyInterfaceImpl();
```

```
enact(anInterfaceImpl, “Hello World!”);
```

深入 Java8 函数式

@FunctionalInterface

Predicate<T> 有参数、条件判断

Function<T, R> 有参数、有返回值

Consumer<T> 无返回值

Supplier<T>　无参数、有返回值

能否进一步简化：方法引用

Java Lambda表达式

代码实例

2.Java8 Stream*

再聊聊 Java 集合与泛型

什么是泛型？

伪泛型，擦除法

运行期怎么判断有泛型？

Lambda 里用泛型

多个泛型约束条件

泛型也是为了简化编程

什么是流

Stream（流）是一个来自数据源的元素队列并支持聚合操作

- 元素：特定类型的对象，形成一个队列。 Java 中的 Stream 并不会存储元素，而是按需计算。
- 数据源：流的来源。可以是集合，数组，I/O channel，产生器 generator 等。
- 聚合操作：类似 SQL 语句一样的操作，比如 filter, map, reduce, find, match, sorted 等。
- 和以前的 Collection 操作不同， Stream 操作还有两个基础的特征：
- Pipelining：中间操作都会返回流对象本身。这样多个操作可以串联成一个管道，如同流式风格（fluent style）。这样做可以对操作进行优化，比如延迟执行（laziness）和短路（short-circuiting）。
- 内部迭代：以前对集合遍历都是通过 Iterator 或者 For-Each 的方式，显式的在集合外部进行迭代，这叫做外部迭代。Stream 提供了内部迭代的方式，通过访问者模式(Visitor)实现。

创建有Stream有哪些方法

Stream 操作

中间操作：

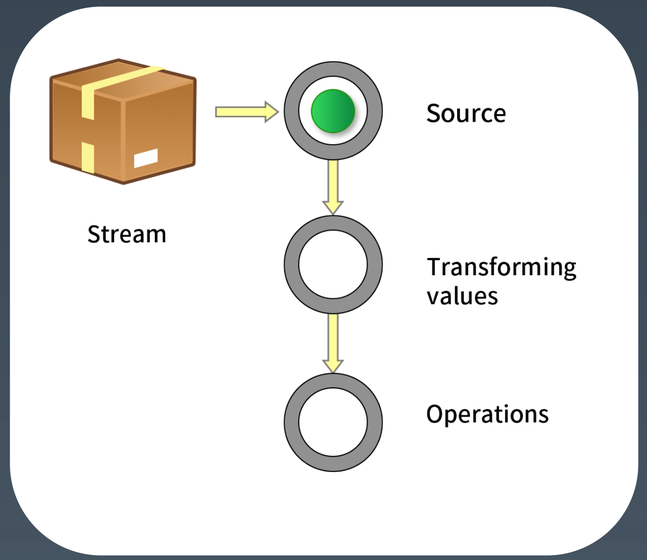
1、选择与过滤

filter(Predicate p) 接收 Lambda ， 从流中排除某些元素。

distinct() 筛选，通过流所生成元素的 hashCode() 和 equals() 去除重复元素。

limit(long maxSize) 截断流，使其元素不超过给定数量。

skip(long n) 跳过元素，返回一个扔掉了前 n 个元素的流。若流中元素不足 n 个，则返回一个空流。



Stream 操作

中间操作：

2、映射

map(Function f) 接收 Lambda ， 将元素转换成其他形式或提取信息；接收一个函数作为参数，该函数会被应用到每个元素上，并将其映射成一个新的元素。

mapToDouble(ToDoubleFunction f) 接收一个函数作为参数，该函数会被应用到每个元素上，产生一个新的 DoubleStream。

mapToInt(ToIntFunction f) 接收一个函数作为参数，该函数会被应用到每个元素上，产生一个新的 IntStream。

mapToLong(ToLongFunction f) 接收一个函数作为参数，该函数会被应用到每个元素上，产生一个新的 LongStream。

flatMap(Function f) 接收一个函数作为参数，将流中的每个值都换成另一个流，然后把所有流连接成一个流。

Stream操作

中间操作：

3、排序

sorted() 产生一个新流，其中按自然顺序排序

sorted(Comparator comp) 产生一个新流，其中按比较器顺序排序

Stream 操作

终止操作：

1.查找与匹配

allMatch——检查是否匹配所有元素
anyMatch——检查是否至少匹配一个元素
noneMatch——检查是否没有匹配的元素
findFirst——返回第一个元素
findAny——返回当前流中的任意元素
count——返回流中元素的总个数
max——返回流中最大值
min——返回流中最小值

2.归约 reduce, 需要初始值（类比 Map-Reduce）

3.收集 collect

4.迭代 forEach

toList List<T> 把流中元素收集到 List
toSet Set<T> 把流中元素收集到 Set
toCollection Collection<T> 把流中元素收集到创建的集合
count 计算流中元素的个数
summaryStatistics 统计最大最小平均值

Steam 操作示例

Stream 代码

Stream 大大简化了集合编程

极客大学

3.Lombok

Lombok 是什么

Lombok 是什么?

Lombok 是基于 jsr269 实现的一个非常神奇的 java 类库，会利用注解自动生成 java Bean 中烦人的 get、set 方法及有参无参构造函数，还能自动生成 logger、ToString、HashCode、Builder 等 java 特色的函数或是符合设计模式的方法，能够让你 java Bean 更简洁，更美观。

基于字节码增强，编译期处理。

可以配置开发工具 IDE 或 Mave 能使用。

编译期增强跟前面讲的字节码工具异同点?

Lombok 示例

@Setter @Getter

@Data

@XXXConstructor

@Builder

@ToString

@Slf4j

泛型也是为了简化编程

4.Guava

什么是 Guava

Guava 是什么?

Guava 是一种基于开源的 Java 库，其中包含谷歌正在由他们很多项目使用的很多核心库。这个库是为了方便编码，并减少编码错误。这个库提供用于集合，缓存，支持原语，并发性，常见注解，字符串处理，I/O 和验证的实用方法。

Guava 的好处

标准化 - Guava 库是由谷歌托管。

高效 - 可靠，快速和有效的扩展 JAVA 标准库。

优化 -Guava 库经过高度的优化。

JDK8 里的一些新特性源于 Guava。

什么是 Guava

集合[Collections]

Guava 对 JDK 集合的扩展，这是 Guava 最成熟和为人所知的部分

1 不可变集合：用不变的集合进行防御性编程和性能提升。

2 新集合类型：multisets, multimaps, tables, bidirectional maps 等

3 强大的集合工具类：提供 java.util.Collections 中没有的集合工具

4 扩展工具类：让实现和扩展集合类变得更容易，比如创建 Collection 的装饰器，或实现迭代器

什么是 Guava

缓存[Caches]

本地缓存实现，支持多种缓存过期策略

```
LoadingCache<Key, Graph> graphs = CacheBuilder.newBuilder()
        .maximumSize(1000)
        .expireAfterWrite(10, TimeUnit.MINUTES)
        .removalListener(MY_LISTENER)
        .build(
            new CacheLoader<Key, Graph>() {
                public Graph load(Key key) throws AnyException {
                    return createExpensiveGraph(key);
                }
        });
```

并发[Concurrency]

ListenableFuture：完成后触发回调的 Future

```
ListeningExecutorService service =
MoreExecutors.listeningDecorator(Executors.newFixedThreadPool(10));
ListenableFuture explosion = service.submit(new Callable() {
  public Explosion call() {
    return pushBigRedButton();
  }
});
Futures.addCallback(explosion, new FutureCallback() {
  // we want this handler to run immediately after we push the big red button!
  public void onSuccess(Explosion explosion) {
    walkAwayFrom(explosion);
  }
  public void onFailure(Throwable thrown) {
    battleArchNemesis(); // escaped the explosion!
  }
});
```

什么是 Guava

字符串处理[Strings]

非常有用的字符串工具，包括分割、连接、填充等操作

事件总线[EventBus]

发布-订阅模式的组件通信，进程内模块间解耦

反射[Reflection]

Guava 的 Java 反射机制工具类

JDK:

```
Foo  foo = (Foo) Proxy.newProxyInstance(
Foo. class.getClassLoader(),
new Class<?>[] {Foo. class},
invocationHandler);
```

Guava:

```
Foo  foo = Reflection.newProxy(Foo. class, invocationHanler);
```

极客大学

5.设计原则*

面向对象设计原则 SOLID

S.O.L.I.D 是面向对象设计和编程(OOD&OOP)中几个重要编码原则(Programming Priciple)的首字母缩写。

1. SRP：The Single Responsibility Principle 单一责任原则
2. OCP：The Open Closed Principle 开放封闭原则
3. LSP：The Liskov Substitution Principle 里氏替换原则
4. ISP：The Interface Segregation Principle 接口分离原则
5. DIP：The Dependency Inversion Principle 依赖倒置原则

最小知识原则/KISS，高内聚低耦合

编码规范、checkstyle

为什么需要编码规范？

常见的编码规范：

1、Google 编码规范：https://google.github.io/styleguide/javaguide.html

2、Alibaba 编码规范：https://github.com/alibaba/p3c

3、VIP 规范：https://vipshop.github.io/vjtools/#/standard/

其他规范：

架构设计规范，技术调研规范，数据库规范等等。

极客大学

6.设计模式*

GoF 23 设计模式

GoF 23 个经典设计模式（面向接口编程），

本质是一类特定场景下通用解决经验。

创建型

1. Factory Method（工厂方法）
2. Abstract Factory（抽象工厂）
3. Builder（建造者）
4. Prototype（原型）
5. Singleton（单例）

结构型

6. Adapter（适配器）
7. Bridge（桥接）
8. Composite（组合）
9. Decorator（装饰）
10. Facade（外观）
11. Flyweight（享元）
12. Proxy（代理）

行为型

13. Interpreter（解释器）
14. Template Method（模板方法）
15. Chain of Responsibility（责任链）
16. Command（命令）
17. Iterator（迭代器）
18. Mediator（中介者）
19. Memento（备忘录）
20. Observer（观察者）
21. State（状态）
22. Strategy（策略）
23. Visitor（访问者）

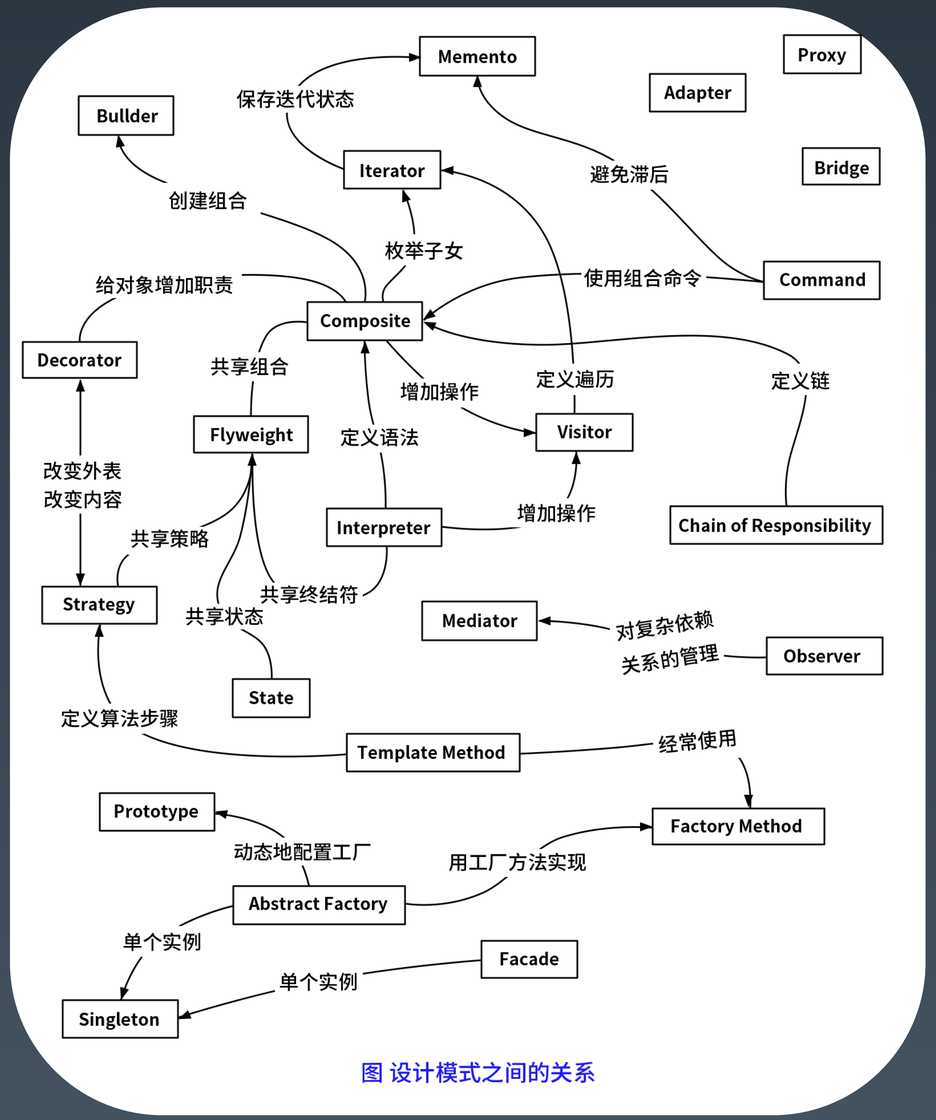


图 设计模式之间的关系

设计模式与反模式

模式的3个层次：解决方案层面（架构模式），组件层面（框架模式），代码层面（GoF 设计模式）

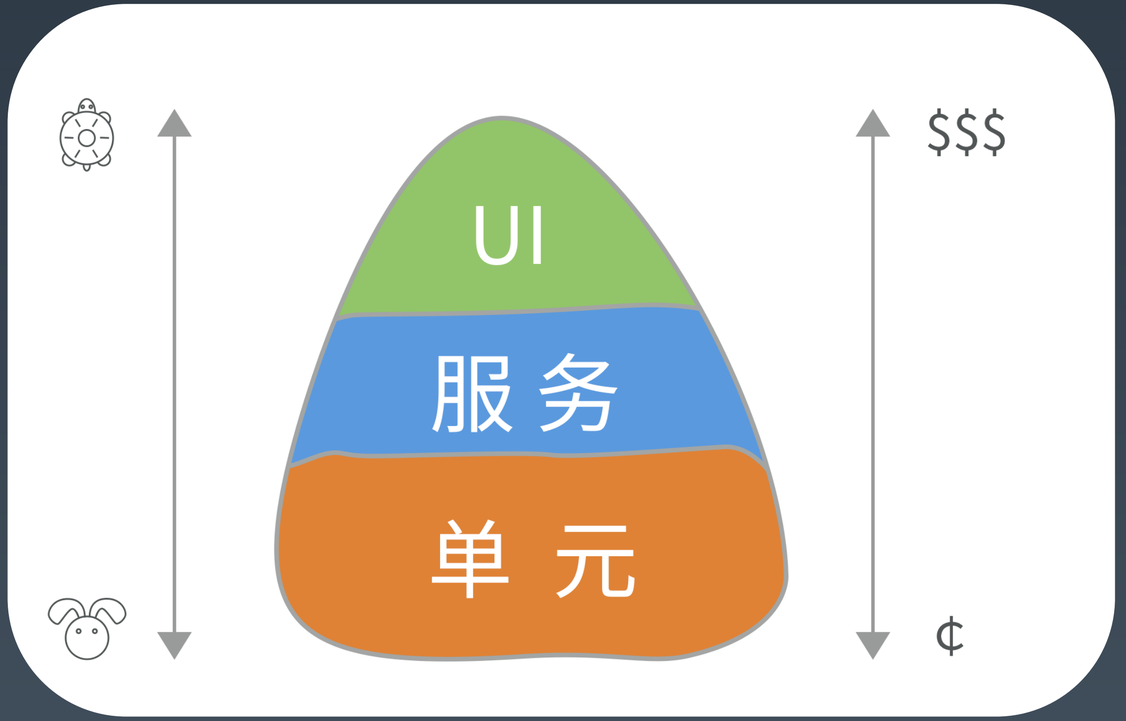
其他模式：集成模式，事务模式，IO 模式/Context 模式，甚至状态机 FSM，规则引擎 RE，workflow 都是模式。

反模式：死用模式，都是反模式。

极客大学

7.单元测试*

什么是单元测试



什么是单元测试

单元测试与新飞机的质量

首先不可避免要回答的一个问题是，“为何要做单元测试？”，我个人的回答是：“这是保证——你写的代码是你想要的结果——的最有效办法！”，当然如果你有更好的办法，请不吝赐教。

没有完备的单元测试的代码所构成的一个系统，就像组装一架飞机，各个配件没有分别经过严格检验，只在最后组装好后，再通过试飞来检验飞机是否正常一样。

尽管软件开发可以“开着飞机换引 擎”，但万一引发了线上事故，影响了绩效，减少了发量，这样的成本还是太高了。所以优秀的工程师总会想尽一切办法保证自己的出品没有质量问题，而单元测试就是一个有力的武器，可以大幅降低大家上线时的紧张指数。

发现缺陷越提前，修复成本越小

如何做单元测试

JUnit -> TestCase, TestSuite, Runner

SpringTest

Mock 技术

- Mockito
- easyMock

如何做单元测试

1. 单元测试方法应该每个方法是一个 case，断言充分，提示明确
2. 单测要覆盖所有的 corner case
3. 充分使用 mock（一切皆可 mock）
4. 如果发现不好测试，则说明业务代码设计存在问题，可以反向优化代码
5. 批量测试用例使用参数化单元测试
6. 注意测试是单线程执行
7. 合理使用 before, after, setup 准备环境
8. 合理使用通用测试基类
9. 配合 checkstyle，coverage 等工具
10. 制定单元测试覆盖率基线

单元测试的常见陷阱与经验

1. 尽量不要访问外部数据库等外部资源
2. 如果必须用数据库考虑用嵌入式 DB+ 事务自动回滚
3. 防止静态变量污染导致测试无效
4. 小心测试方法的顺序导致的不同环境测试失败
5. 单元测试总时间特别长的问题

极客大学

9.总结回顾与作业实践

第 11 课总结回顾

极客大学

Java8 Lambda/Stream

Lombok/Guava

设计原则与设计模式

单元测试与编程经验

第11课作业实践

1、（选做）尝试使用 Lambda/Stream/Guava 优化之前作业的代码。

2、（选做）尝试使用 Lambda/Stream/Guava 优化工作中编码的代码。

3、（选做）根据课上提供的材料，系统性学习一遍设计模式，并在工作学习中思考如何用设计模式解决问题。

4、（选做）根据课上提供的材料，深入了解 Google 和 Alibaba 编码规范，并根据这些规范，检查自己写代码是否符合规范，有什么可以改进的。

THANKS!