软件设计启示

Talk About Crud, Pattern, OOADP,DDD

李力-2020-10-14

软件开发隐喻

- 软件开发隐喻：开车，建造软件，系统生长，焦油坑，软件工程，软件工艺，码农，工匠，软件蓝领，两顶帽子，平时用的最多的隐喻是挖坑，填坑，码农，多瘤程序员，写字楼的农民工……
- 社会现象隐喻：养鱼，开车，海王
- 个人观点：种树
- 个人最喜欢：两顶帽子
- 中国传统造物思想：
  - 熟能生巧,巧能生妙,妙能生绝,绝能生神
  - 天人合一，道器合一，师法自然，格物致用，见朴抱素
  - 建筑，瓷器

启示的目的是什么？

- 软件设计目的：减少复杂性
- 首要技术目标与追求：消除偶然复杂性，控制本质复杂性
  - 消非控根
- 新角度认识Saas代码

大纲

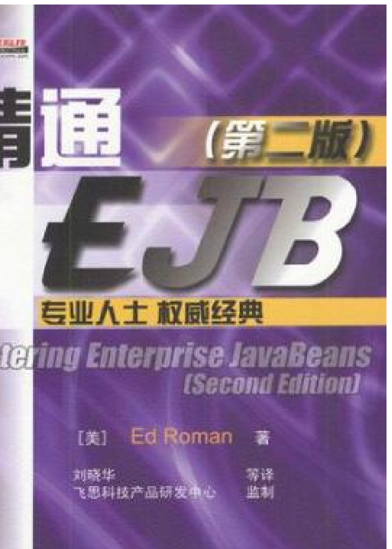
- 1 CRUD
  - 命名
  - MVC标准过程
  - 修改代码
- 2 Pattern
  - 依赖管理
  - 资源管理
  - 设计模式
  - GRASP
- 3 OOADP
  - 重构
  - 建模
  - RedisClientTemplate扩展
- 4 DDD
  - 统一语言
  - 充提限界上下文
  - 开放平台的边界
- 5 设计综合趣谈
  - Java集合灵魂
  - API Design About LoadingCache，ThreadLocal，Unix IO
  - Executor Framework
  - AOP Alliance
- 6 软件设计启发录

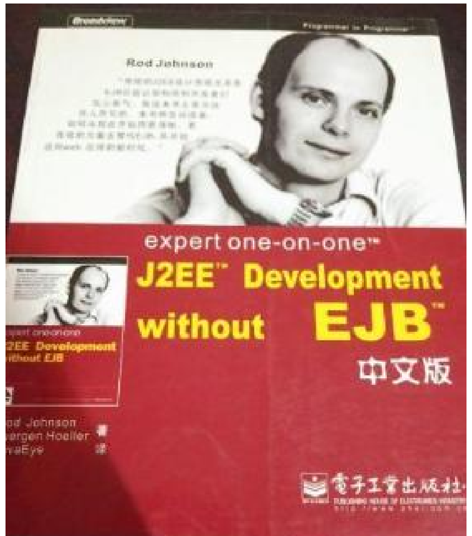
1. CRUD

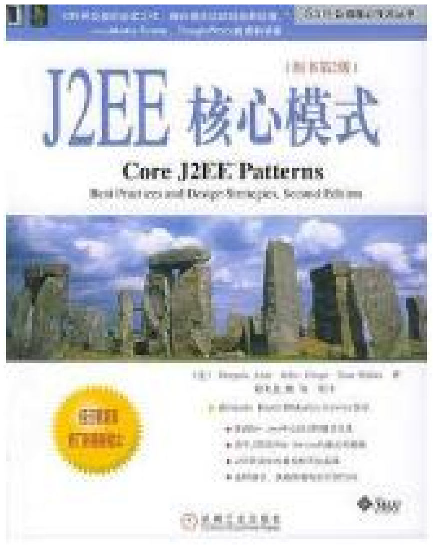
应用开发演进之路

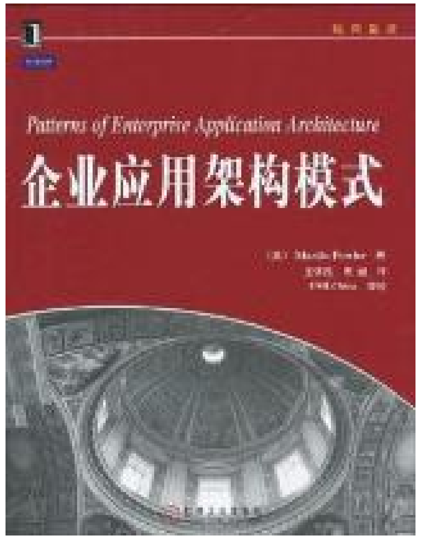
- 1 架构层
- 单体->SOA(Dubbo)-> 微服务->ServerLess->Faas

- 2 设计层









CRUD

- 思想：数据流组成了系统 CRUD的三层境界
- 1 命名的想象
  - RedisClientTemplate和RedisClientUtil有什么区别？
  - 写个工具类？慎重！ ThreadFactoryUtil，FixedThreadBoundedQueueFactoryUtils.newFixedThreadQueuePoolPrintLog ，RedisDumpParser，Executors，工具类越多，代表系统职责分配却差劲。
  - ***Service，***Util，***EnumType -》 GatewayInvoker，HttpEncryptDecryptEngine，MemoryCacheManager，EnumStatus，FunctionHub
  - Set/Get Lombok -> moveLeft , drawX , clearStatus 充提，场外交易校验在哪里做？（三种思路的差异）
  - 隐喻文化
- 2 方法签名的艺术
- 3 伟大的注释：优化设计，维护性，职责划分，java的代码文档，javadoc
- 4 TDD：为什么测试很重要？解耦，扩展性，代码质量保证
- 5 代码生成：MybatisPlus真的好吗？对系统坏处在哪里？
- 6 Web MVC演进之路(产生贫血对象和集中式控制对系统伤害最大，退化成面向过程数据流编程)

CRUD场景1-命名

- 目标：好读，很难读错，好写
- 命名也许是API可用性中最重要的一个因素。
- 起名字花费写代码1/5时间
- 名词，动词，形容词，副词，可变与不可变
- 对称的世界
- NDD

- 描述起名字：
  - 琅琅上口
  - 绕梁三日
  - 言必有中
  - 家喻户晓
  - 众所周知
  - 词不达意
  - 言过其实
  - 一词多意
  - 多词一意

- Nouns for classes
  - `BigInteger`, `PriorityQueue`
- Nouns or adjectives for interfaces
  - `Collection`, `Comparable`
- Nouns, linking verbs or prepositions for non-mutative methods
  - `size`, `isEmpty`, `plus`
- Action verbs for mutative methods
  - `put`, `add`, `clear`
- If you follow these, they quickly become second nature

TwoHardThings

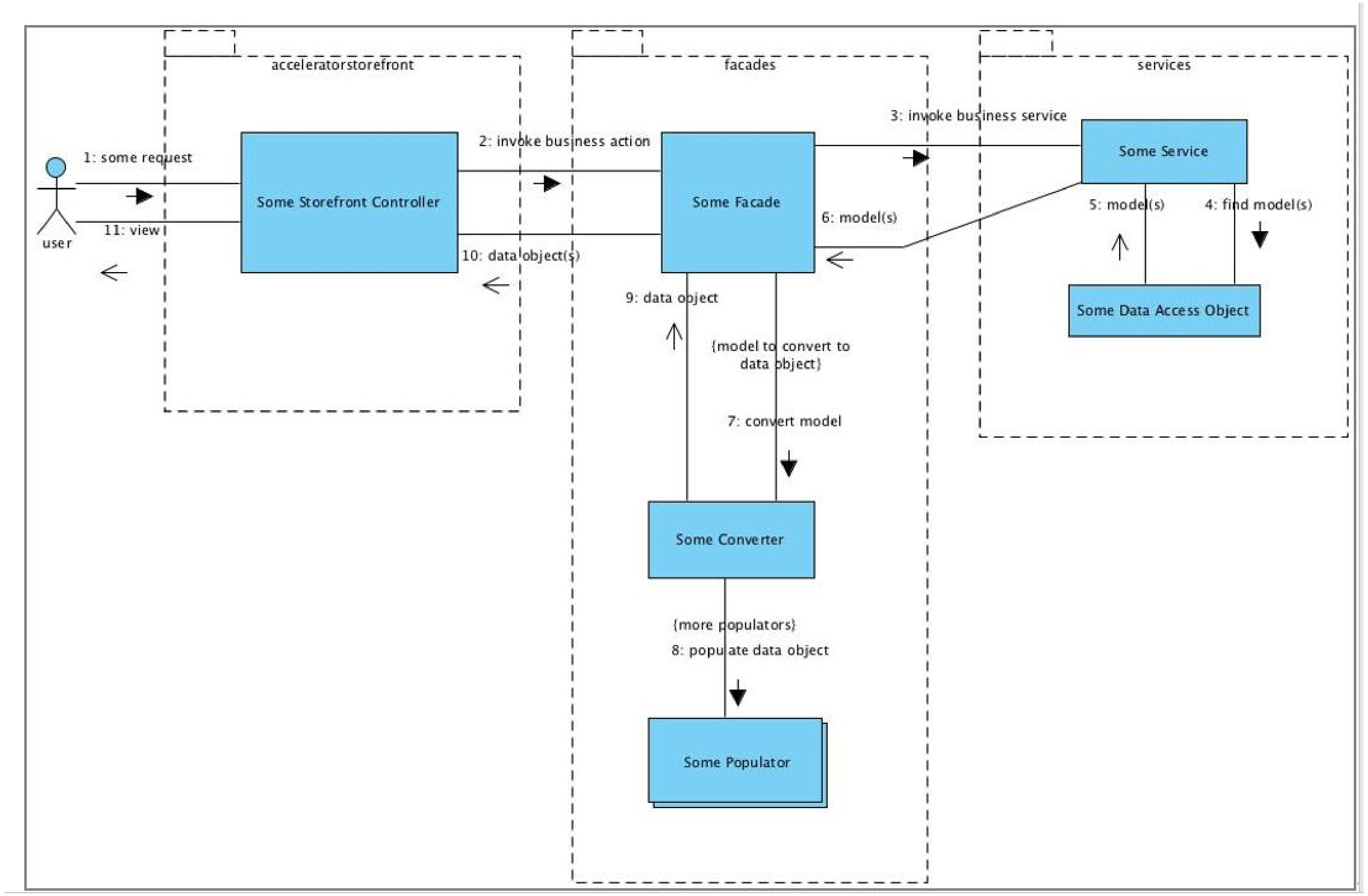
Martin Fowler

14 July 2009

> *There are only two hard things in Computer Science: cache invalidation and naming things.*
>
> *-- Phil Karlton*

CRUD场景2 MVC过程

- servlet+html
- servlet+jsp
- servlet+dao
- Model1
- Model2
- ssm框架



CRUD场景3-如何改代码

- 修改代码的技术与过程
  - 如何修改PublicInfo大泥球?
  - 如何改一行代码?
- Refactor Domain Model
- Refactor Design
- Refactor Code（Implement）

2. Pattern

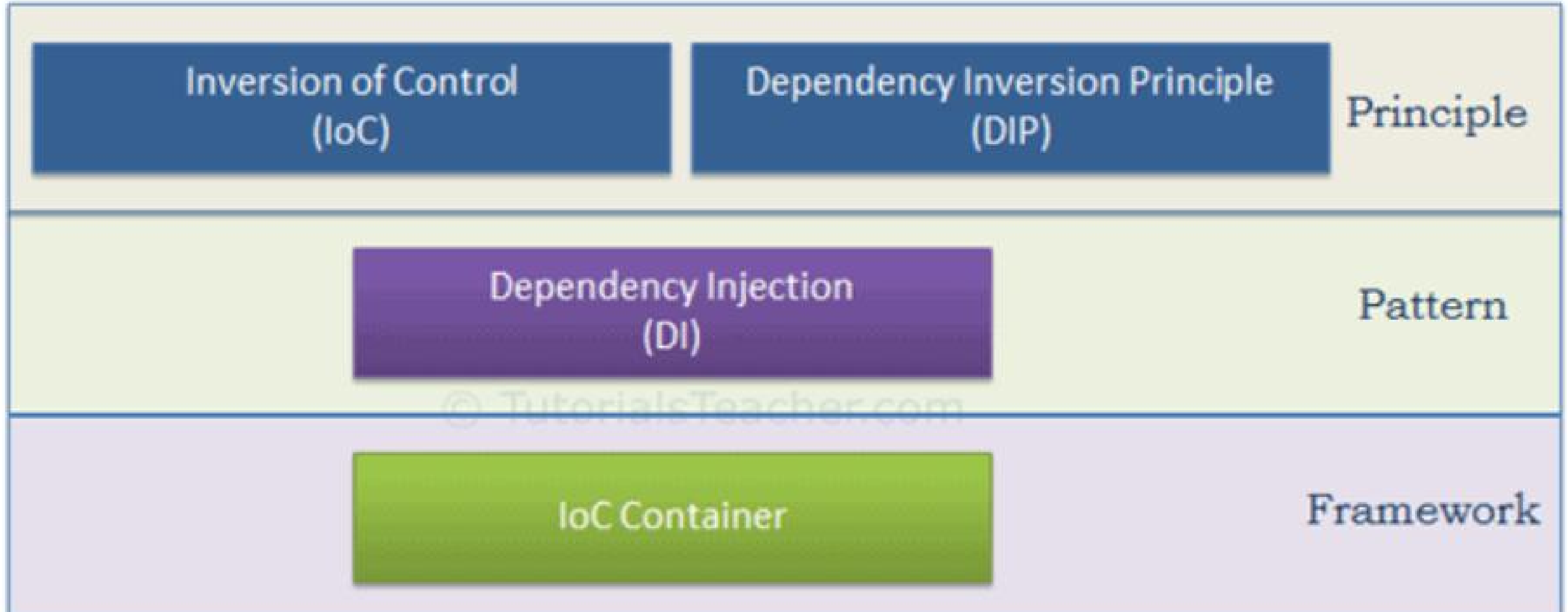
Pattern

- 思想：复用
- GRASP-9
- SOLID
- POSA
- POEAA
- J2EE Pattern
- OOD:GOF23

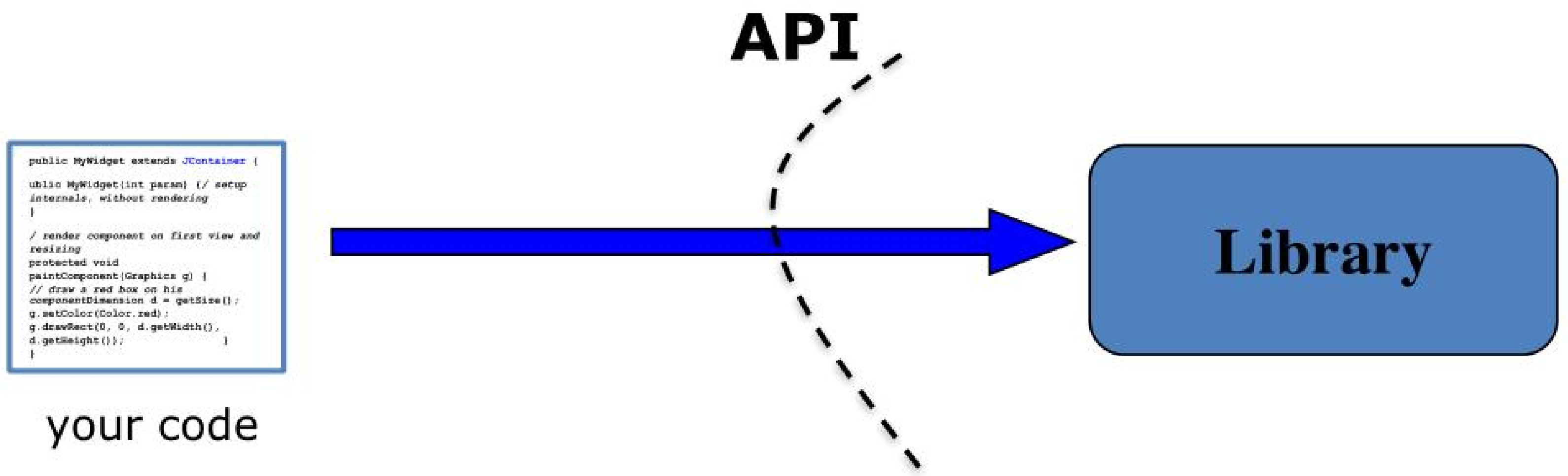
Pattern场景1-依赖管理

DI， DIP， IOC， 依赖注入问题(项目遇到最多， 写代码最值得注意的地方)， 循环依赖详解， 为什么没有关联注入？注入父类？

- 如何开发基础类库？cache， mq， service register/discover
- 神奇的魔法：Spring一直在调用你的代码 -》 Faas 容器的通用设计 -》 容器类设计



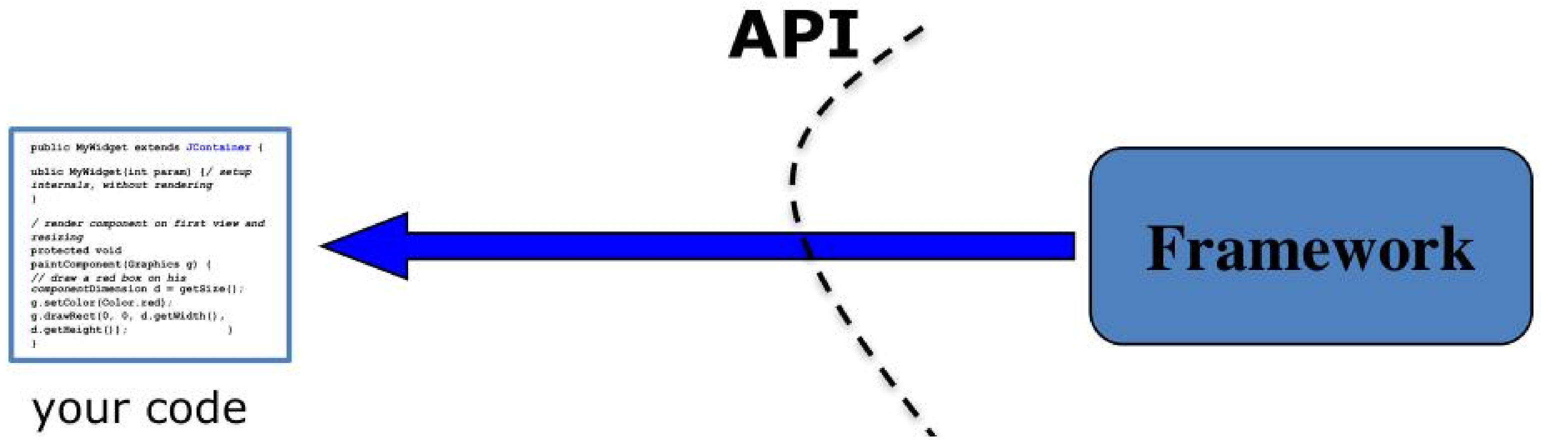
到底是谁管理谁？



```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());                    }
}
```



```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());                    }
}
```

Pattern场景2-资源管理POSA3

- Resource：计算机，web，Spring，组件，堆
- Resource Pattern
  - 资源获取：LookUp，Lazy Acquisition，Eager Acquisition，Partial Acquisition
  - 资源生命周期：Caching，Pooling，Coordinator，Resource Lifecycle Manager
  - 资源释放：Leasing，Evictor
- Resource Manager？Container

Pattern场景3-设计模式

- GOF 23
  - 为什么你想不到模式呢？1 缺少抽象，没有建模思想 2 对23种设计模式不熟悉
  - 为什么你想到了却不会用呢？ 1 想错了 2 不理解模式解决的问题 3 代码功底不好
  - Fianace：加密机创建是单例，调用加密机和hicoin是façade，并不是封一层，对比私有化代码

  - 项目例子：com.chainup.common.result.CommonResult VS @Builder

Pattern场景4-GRASP

- Spring IOC拒绝创建者。 订单和产品谁创建谁？ Spring使得对对象思考少了很多。但是当你实际写超过5个类以上交互，就会体会设计的价值了。
- PV(Pure fabrication)是根本原则，评估开发者和架构师的成熟度。Information Hidden.
- 源码阅读指南(1个类，5个类，50个类，500个类，2000个类)

通用职责分配软件模式或原则（GRASP）

模式/原则	描　　述
信息专家（Information Expert）	对象设计和职责分配的基本原则是什么？ 将职责分配给信息专家，信息专家是指具有履行职责所需信息的类
创建者（Creator）	谁来创建？（注意，工厂（Factory）是常见的替代方案。） 当以下条件之一为真时，将创建类A实例的职责分配给类B： 1）B容纳A；2）B聚集A；3）B具有A的初始化数据； 4）B记录A；5）B紧密地使用A
控制器（Controller）	在UI层之上第一个接受和协调（“控制”）系统操作的对象是哪个？ 将职责分配给代表以下选择之一的对象： 1）代表整个“系统”、“根对象”、运行软件的设备，或者是主要子系统（这些都是外观（facade）控制器的变体） 2）代表发生该系统操作的用例场景（用例或会话控制器）
低耦合（Low Coupling）（评价性）	如何减少变化所产生的影响？ 职责的分配要使（不必要）的耦合保持为低。使用这一原则来评价候选方案
高内聚（High Cohesion）（评价性）	如何保持对象有重点、可理解和可管理，同时还有支持低耦合的作用？ 职责的分配要保持高内聚。使用这一原则来评价候选方案
多态（Polymorphism）	当行为基于类型变化时，谁对此负责？ 当相关候选者或行为基于类型（类）而变化时，使用多态操作将行为职责分配给行为所变化的类型
纯虚构（Pure Fabrication）	当你走投无路但又不想破坏高内聚和低耦合时，将职责分配给谁？ 将一组高度内聚的职责分配给人为的或便利性的“行为”类，该类并不代表问题领域概念，而是虚构的事物，以此来支持高内聚、低耦合和复用性
间接性（Indirection）	如何分配职责以避免直接耦合？ 将职责分配给中介对象，由该对象来协调其他构件或服务，以避免其直接耦合
防止变异（Protected Variations）	如何给对象、子系统和系统分配职责，以使这些元素中的变化或不稳定性不会对其他元素产生不良影响？ 确定预计变化或不稳定之处；为其创建稳定“接口”以分配职责

3. OOADP

OOADP

思想核心：分类思想

- 1　抽象
- 2　封装
- 3　模块化
- 4　层次化

- OO
- 封装，继承，多态，组合

- OOA：从问题域词汇表确定类(关键抽象)，创建协作结构(机制)，多组对象一起工作，满足需求，
- **OOD：RDD，GRASP，GOF**
- OOP: 类和对象，接口，抽象类，方法，消息

OOA

- 经典方法
- 行为分析
- 领域分析
- 用例分析
- CRC
- 非正式描述
- 结构化分析

OOAD场景1-重构

- 重构哪些?
- @Refactor Project
- https://github.com/xiaozhiliaoo/refactor
- <dependency>
-     <groupId>com.github.xiaozhiliaoo</groupId>
-     <artifactId>refactor</artifactId>
-     <version>0.0.2</version>
- </dependency>

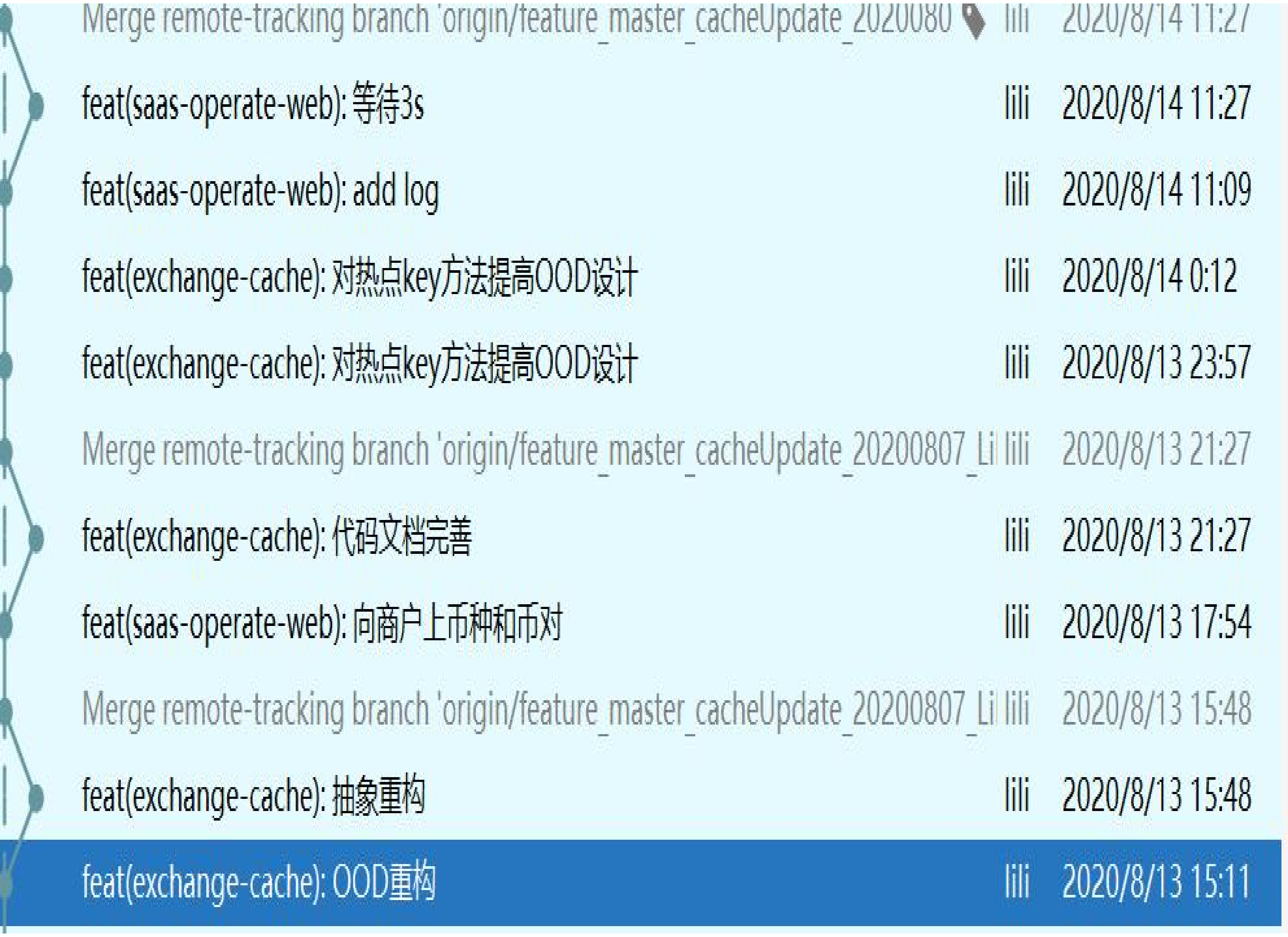
- 重复抽象： 场外订单的pc+app+open+open-api-server format 四份代码
- 大泥球： PublicInfo

OOAD场景2-建模

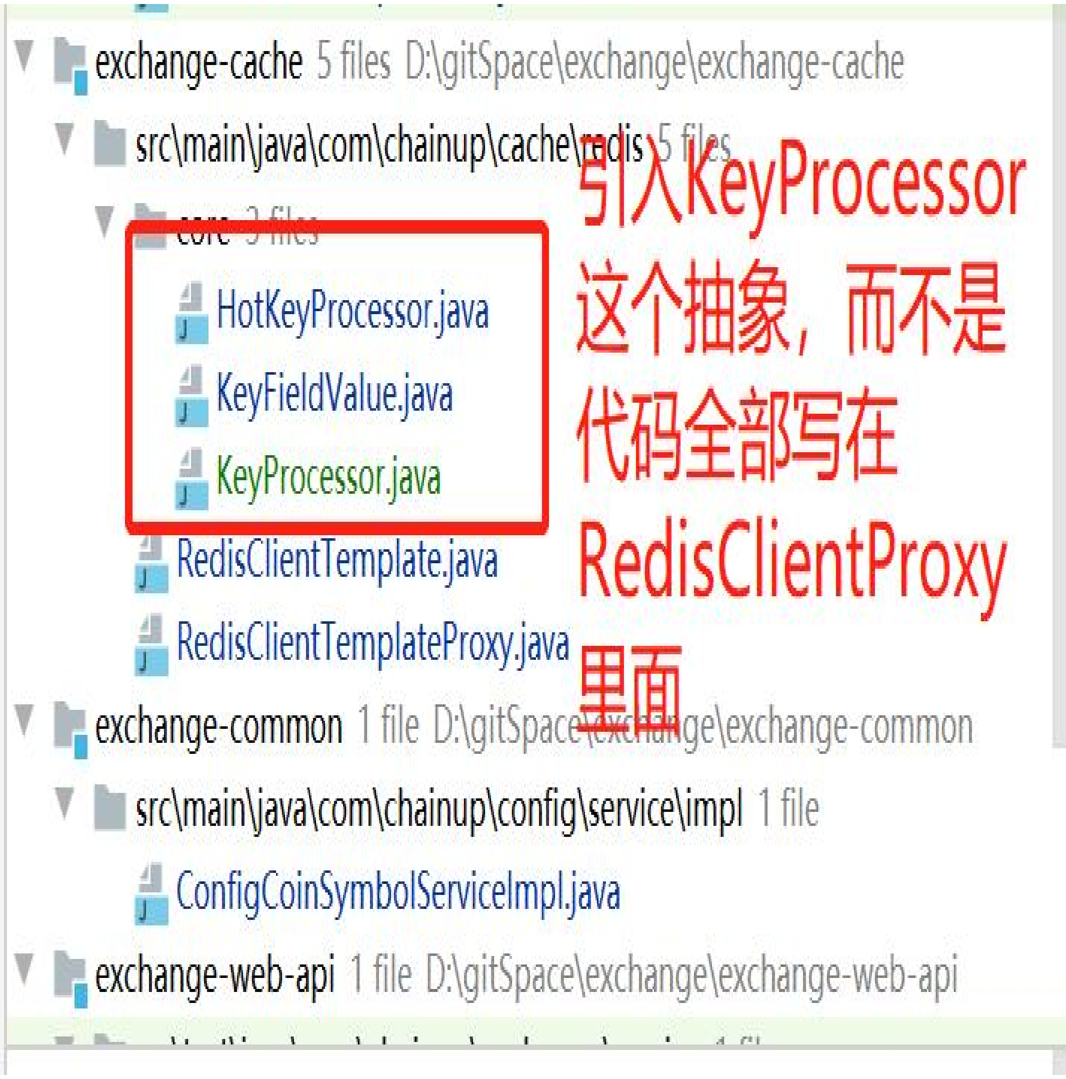
- 书和书架关系是什么?
- 房间，调温器。
- 把大象放进冰箱有几步?
- 棉花糖是糖，不是棉花。
- 番茄是水果，还是蔬菜。
- 钱(Money)是BigDecimal吗?
- 石头，剪刀，布怎么建模？并且实现呢？Double Dispatch

OOAD场景3-Redis的扩展

- RedisClientTemplate历史，设计剖析和缺陷。
  - 从 RedisClientTemplate -> RedisClientTemplate+RedisClientTemplateProxy-> RedisClientTemplate+RedisClientTemplateProxy+KeyProcessor->??? 依旧很差的设计，缺少机制的设计Proxy和Processor的交互没设计好，拦截器模式是否更好？
- 实现思考：
  - OOA：CRC卡，用例，用户故事
  - OOD：
  - 1 确定关键抽象
    - 1.1 细化关键抽象
    - 1.2 命名关键抽象
  - 2 识别机制
    - 2,1 机制即模式，设计的灵魂
  - 3 评估抽象
    - 耦合，内聚，充分，完整，基础
- OOP: LoadingCache….If else …
- 框架设计精髓-Junit4



Merge remote-tracking branch 'origin/feature_master_cacheUpdate_2020080	lili	2020/8/14 11:27
feat(saas-operate-web): 等待3s	lili	2020/8/14 11:27
feat(saas-operate-web): add log	lili	2020/8/14 11:09
feat(exchange-cache): 对热点key方法提高OOD设计	lili	2020/8/14 0:12
feat(exchange-cache): 对热点key方法提高OOD设计	lili	2020/8/13 23:57
Merge remote-tracking branch 'origin/feature_master_cacheUpdate_20200807_Li	lili	2020/8/13 21:27
feat(exchange-cache): 代码文档完善	lili	2020/8/13 21:27
feat(saas-operate-web): 向商户上币种和币对	lili	2020/8/13 17:54
Merge remote-tracking branch 'origin/feature_master_cacheUpdate_20200807_Li	lili	2020/8/13 15:48
feat(exchange-cache): 抽象重构	lili	2020/8/13 15:48
feat(exchange-cache): OOD重构	lili	2020/8/13 15:11



```
▼ exchange-cache 5 files D:\gitSpace\exchange\exchange-cache
  ▼ src\main\java\com\chainup\cache\redis 5 files
    ▼ core 3 files
        HotKeyProcessor.java
        KeyFieldValue.java
        KeyProcessor.java
      RedisClientTemplate.java
      RedisClientTemplateProxy.java
▼ exchange-common 1 file D:\gitSpace\exchange\exchange-common
  ▼ src\main\java\com\chainup\config\service\impl 1 file
      ConfigCoinSymbolServiceImpl.java
▼ exchange-web-api 1 file D:\gitSpace\exchange\exchange-web-api
```

引入KeyProcessor这个抽象，而不是代码全部写在RedisClientProxy里面

4. DDD

DDD

- 1 思想
- 2 战略设计
- 3 战术设计
- 4 Saas场景

1 思想

- 领域成为最重要的关注点
- 复杂软件控制之道
- 新银弹？？？

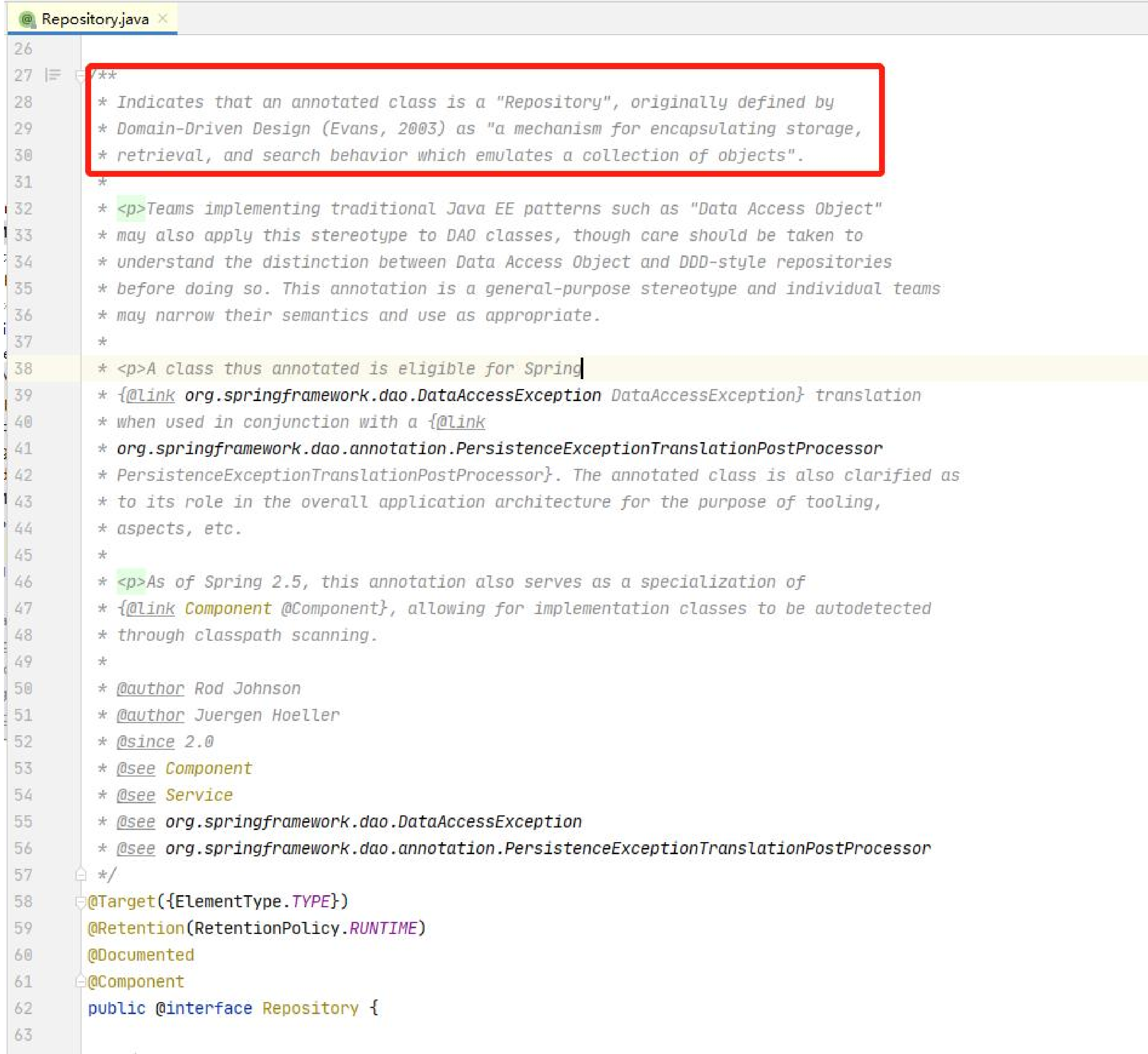
2 战略设计

- 限界上下文
- 通用语言
- 上下文映射

3 战术设计

- 聚合
- 实体
- 值对象
- CQRS
- 事件溯源

- @Repository
- @Configurable


Repository.java

```java
/**
 * Indicates that an annotated class is a "Repository", originally defined by
 * Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage,
 * retrieval, and search behavior which emulates a collection of objects".
 *
 * <p>Teams implementing traditional Java EE patterns such as "Data Access Object"
 * may also apply this stereotype to DAO classes, though care should be taken to
 * understand the distinction between Data Access Object and DDD-style repositories
 * before doing so. This annotation is a general-purpose stereotype and individual teams
 * may narrow their semantics and use as appropriate.
 *
 * <p>A class thus annotated is eligible for Spring
 * {@link org.springframework.dao.DataAccessException DataAccessException} translation
 * when used in conjunction with a {@link
 * org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor
 * PersistenceExceptionTranslationPostProcessor}. The annotated class is also clarified as
 * to its role in the overall application architecture for the purpose of tooling,
 * aspects, etc.
 *
 * <p>As of Spring 2.5, this annotation also serves as a specialization of
 * {@link Component @Component}, allowing for implementation classes to be autodetected
 * through classpath scanning.
 *
 * @author Rod Johnson
 * @author Juergen Hoeller
 * @since 2.0
 * @see Component
 * @see Service
 * @see org.springframework.dao.DataAccessException
 * @see org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor
 */
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Repository {
```

4 框架

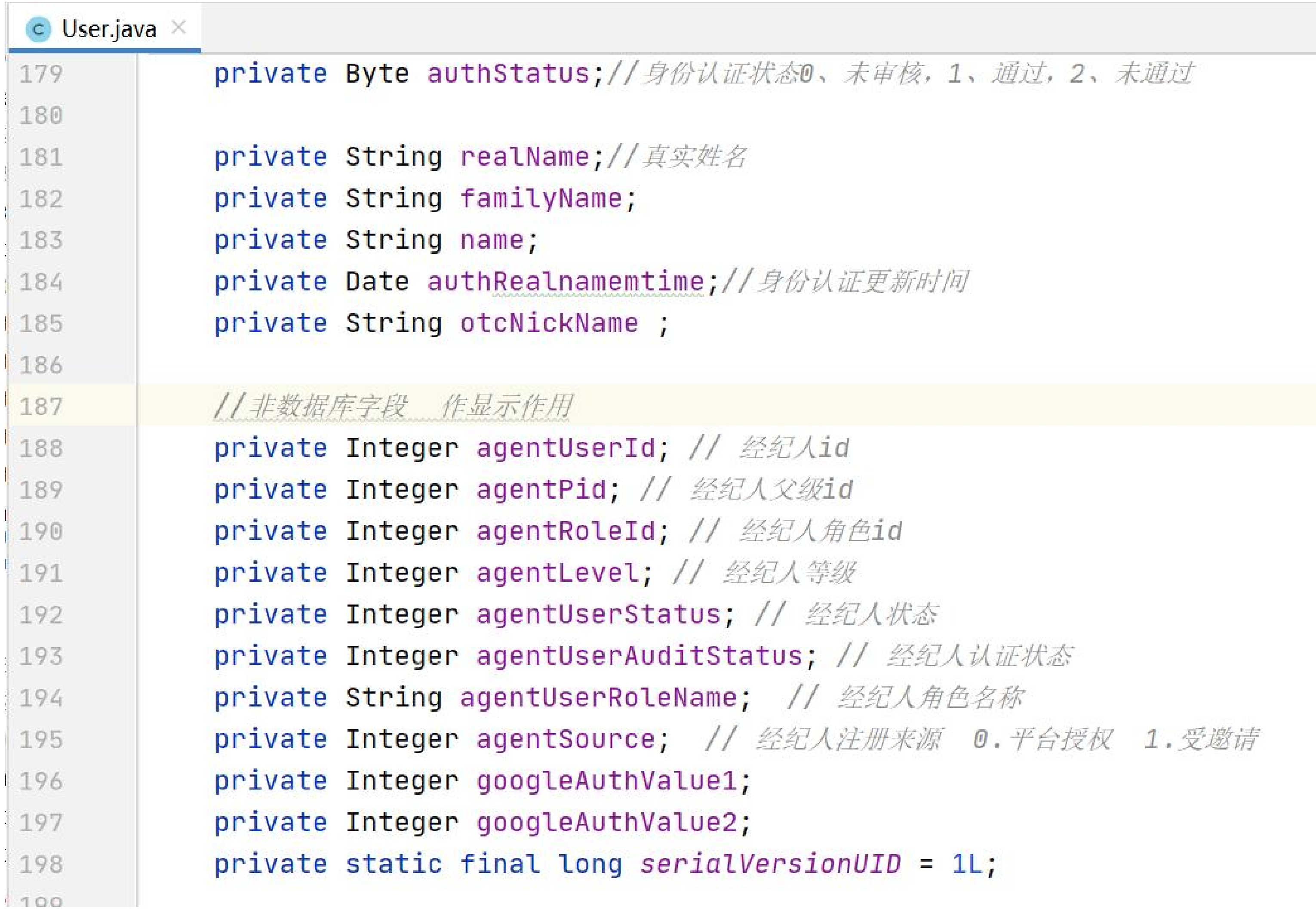
- Alibaba-COLA (https://github.com/alibaba/COLA)
- Jdonframework (https://github.com/banq/jdonframework)

DDD场景

- 1 统一语言：PublicInfo，Base，风控，充提，币种币对，用户，管理员，Waas
- 2 账户Account里面操作充值，转账
- 3 统一场外订单列表展示 != formatOrderList (缺乏想象力，建模水平差，非常重要业务场景却叫格式化订单？)

缺少领域的概念

- 领域和数据库对象混合



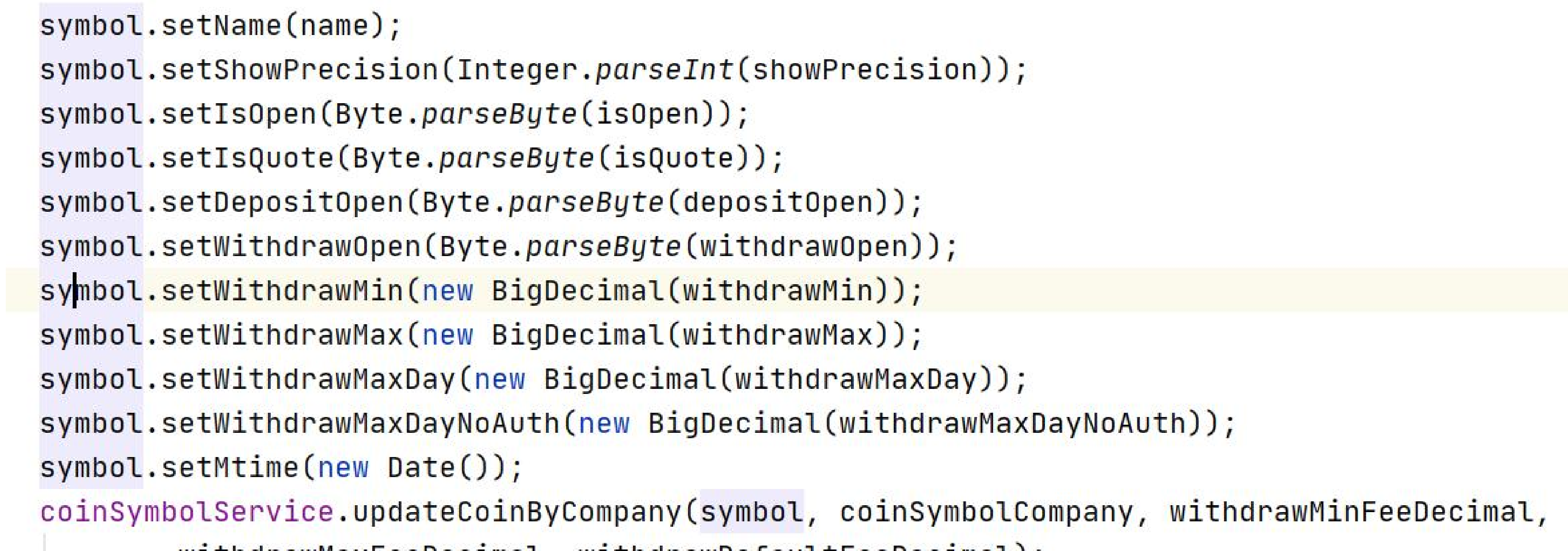
```java
    private Byte authStatus;//身份认证状态0、未审核，1、通过，2、未通过

    private String realName;//真实姓名
    private String familyName;
    private String name;
    private Date authRealnamemtime;//身份认证更新时间
    private String otcNickName ;

    //非数据库字段  作显示作用
    private Integer agentUserId; // 经纪人id
    private Integer agentPid; // 经纪人父级id
    private Integer agentRoleId; // 经纪人角色id
    private Integer agentLevel; // 经纪人等级
    private Integer agentUserStatus; // 经纪人状态
    private Integer agentUserAuditStatus; // 经纪人认证状态
    private String agentUserRoleName;  // 经纪人角色名称
    private Integer agentSource;  // 经纪人注册来源  0.平台授权  1.受邀请
    private Integer googleAuthValue1;
    private Integer googleAuthValue2;
    private static final long serialVersionUID = 1L;
```

DDD场景-充提，上币

- 事务脚本(从Mysql到**中间件**)-》限界上下文+领域对象
- 可能是商户修改,新加币种，也可能总后台，也可以是一个查询?
- 没有任何业务含义-》每次修改从头看代码，**浪费时间**-》丰富业务模型



```
symbol.setName(name);
symbol.setShowPrecision(Integer.parseInt(showPrecision));
symbol.setIsOpen(Byte.parseByte(isOpen));
symbol.setIsQuote(Byte.parseByte(isQuote));
symbol.setDepositOpen(Byte.parseByte(depositOpen));
symbol.setWithdrawOpen(Byte.parseByte(withdrawOpen));
symbol.setWithdrawMin(new BigDecimal(withdrawMin));
symbol.setWithdrawMax(new BigDecimal(withdrawMax));
symbol.setWithdrawMaxDay(new BigDecimal(withdrawMaxDay));
symbol.setWithdrawMaxDayNoAuth(new BigDecimal(withdrawMaxDayNoAuth));
symbol.setMtime(new Date());
coinSymbolService.updateCoinByCompany(symbol, coinSymbolCompany, withdrawMinFeeDecimal,
        withdrawMaxFeeDecimal, withdrawDefaultFeeDecimal);
```

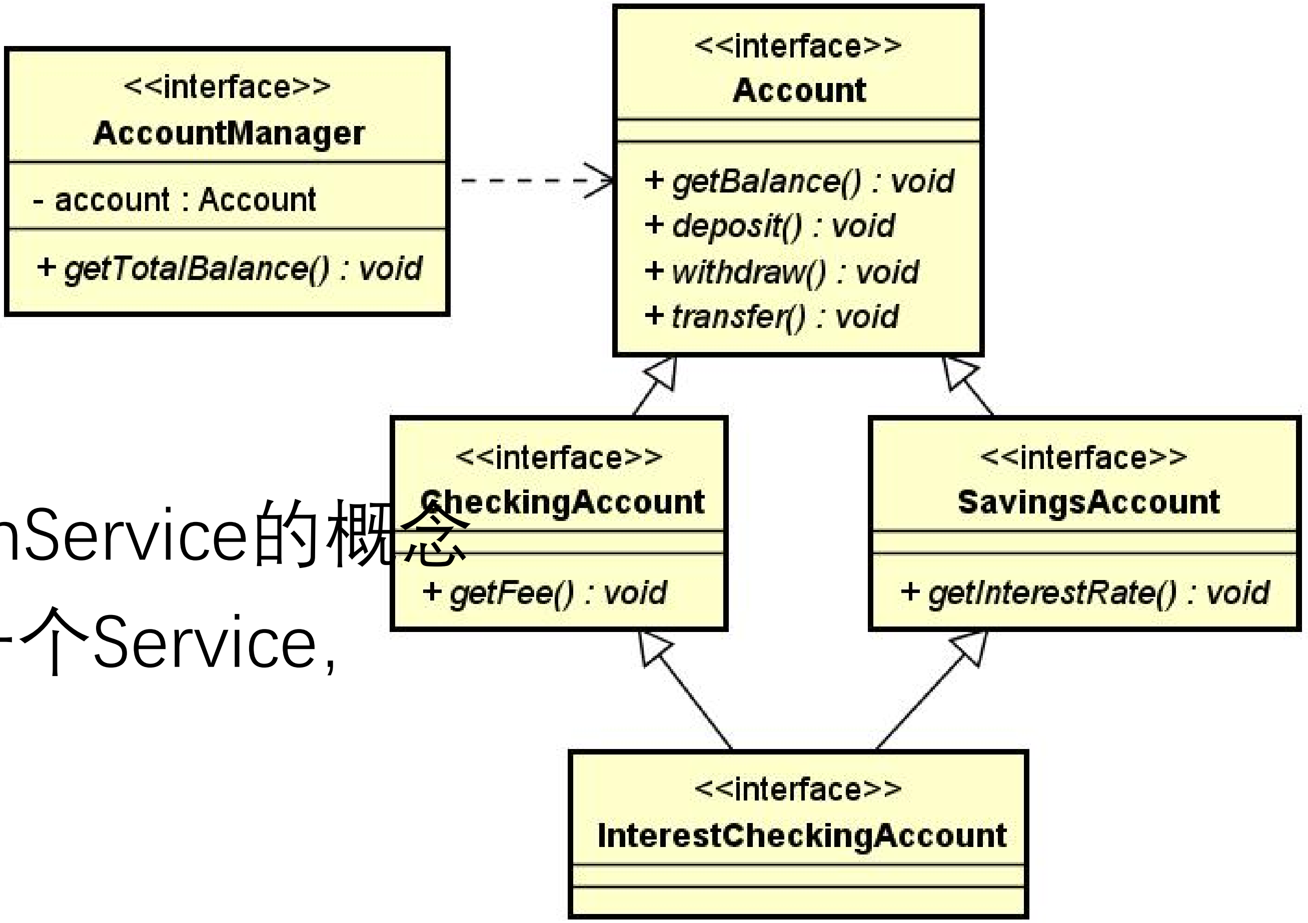
DDD场景-充提建模

- 账户， 流水， 充提
- 如果不是事务脚本呢？
- Account聚合DespositService

和WithdrawService， TransactionService的概念

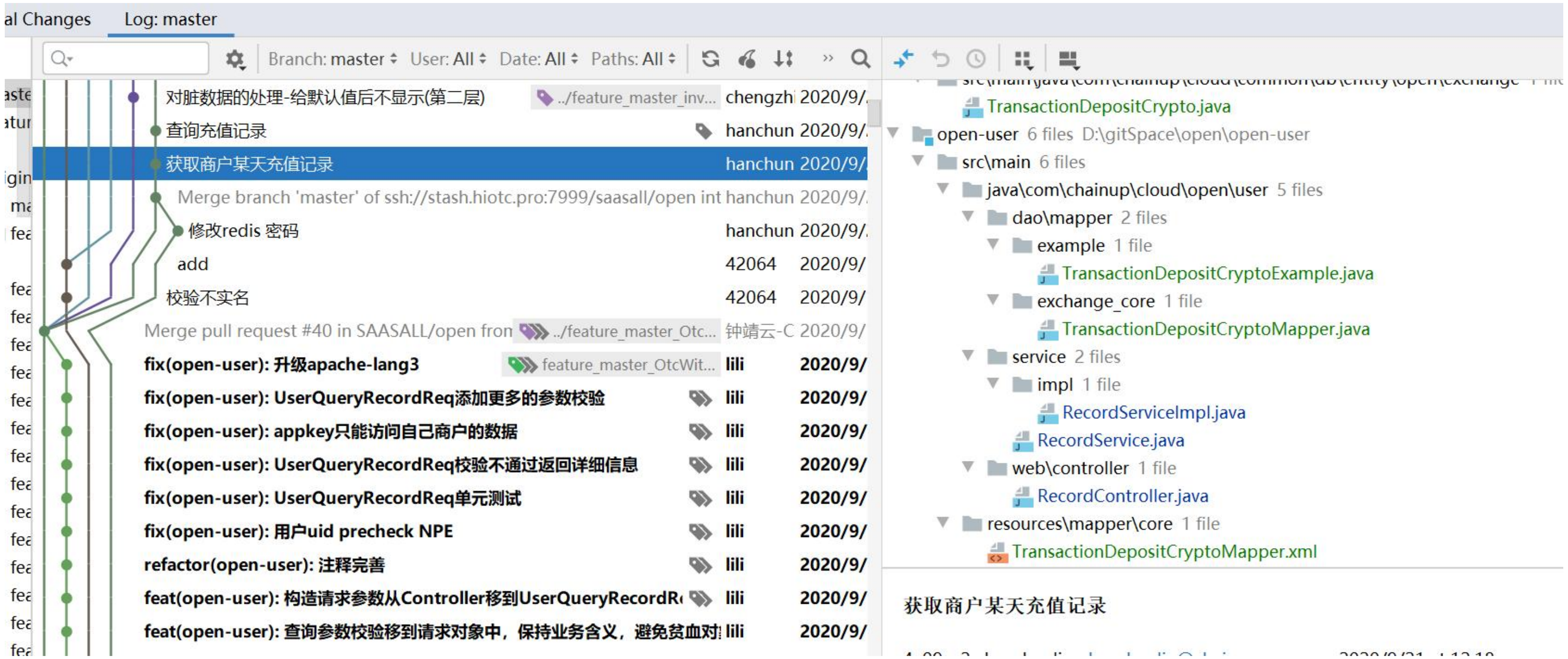
代码生成会使得你每个表建立一个Service，

账户系统统一入口。



DDD-场景

- 开放平台参数校验，自己检查自己



al Changes　Log: master

Branch: master　User: All　Date: All　Paths: All

```
对脏数据的处理-给默认值后不显示(第二层)        ../feature_master_inv...   chengzhi 2020/9/.
查询充值记录                                                         hanchun  2020/9/.
获取商户某天充值记录                                                 hanchun  2020/9/.
Merge branch 'master' of ssh://stash.hiotc.pro:7999/saasall/open int hanchun  2020/9/.
修改redis 密码                                                       hanchun  2020/9/.
add                                                                  42064    2020/9/
校验不实名                                                           42064    2020/9/
Merge pull request #40 in SAASALL/open fron ../feature_master_Otc... 钟靖云-C 2020/9/
fix(open-user): 升级apache-lang3                feature_master_OtcWit... lili     2020/9/
fix(open-user): UserQueryRecordReq添加更多的参数校验                 lili     2020/9/
fix(open-user): appkey只能访问自己商户的数据                         lili     2020/9/
fix(open-user): UserQueryRecordReq校验不通过返回详细信息             lili     2020/9/
fix(open-user): UserQueryRecordReq单元测试                           lili     2020/9/
fix(open-user): 用户uid precheck NPE                                 lili     2020/9/
refactor(open-user): 注释完善                                        lili     2020/9/
feat(open-user): 构造请求参数从Controller移到UserQueryRecordR        lili     2020/9/
feat(open-user): 查询参数校验移到请求对象中，保持业务含义，避免贫血对 lili     2020/9/
```

```
      TransactionDepositCrypto.java
▼ open-user  6 files  D:\gitSpace\open\open-user
  ▼ src\main  6 files
    ▼ java\com\chainup\cloud\open\user  5 files
      ▼ dao\mapper  2 files
        ▼ example  1 file
            TransactionDepositCryptoExample.java
        ▼ exchange_core  1 file
            TransactionDepositCryptoMapper.java
      ▼ service  2 files
        ▼ impl  1 file
            RecordServiceImpl.java
          RecordService.java
      ▼ web\controller  1 file
          RecordController.java
    ▼ resources\mapper\core  1 file
        TransactionDepositCryptoMapper.xml
```

获取商户某天充值记录

5. 设计综合趣谈

设计的杂谈

- 1 写代码重在设计
- 2 设计保持一致性
- 3 一致性带来品味
- 4 同时重视原理和**设计**
    - 两道经典面试题
        - HashMap原理是什么?
        - 线程状态有哪些?

Java集合类API-接口是灵魂

特点：超级复杂，非常灵活，类库核心，扩展容易，技术体现

解决：接口定义类型，抽象类实现骨架，具体类实现功能

经典实例：Apache collections 扩展，Guava collections扩展，Apache Common Event扩展

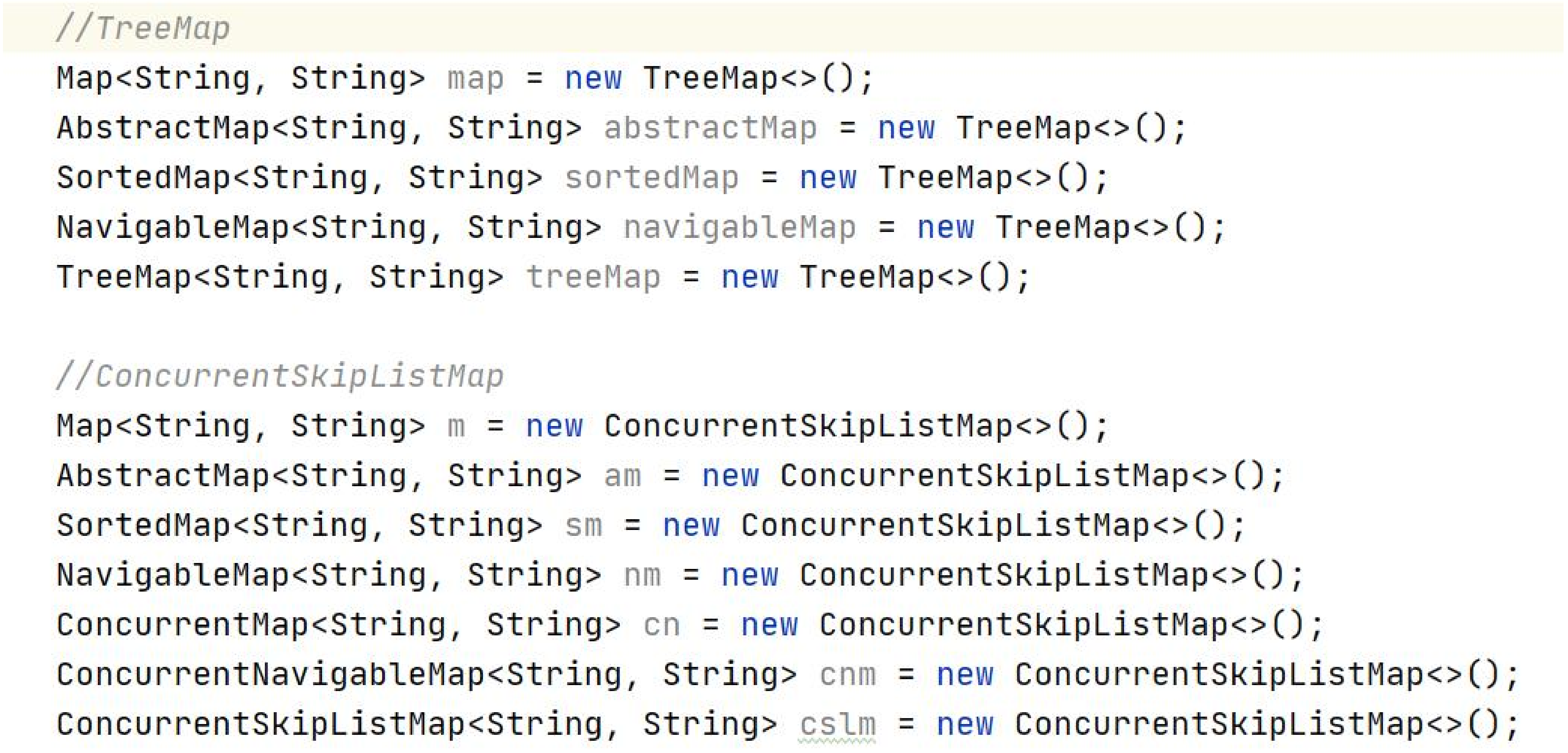
设计目标：Easily extensibility, reuse，change，Small and simple，powerful，

架构：Core Collection Interface，General-Purpose Implementations，
Wrapper Implementations， Abstract Implementations，Algorithms

实现设计目标：Iterator，Decorator，Template，Adapter，Strategy，Marker interface

思考：如何创建一个TreeMap呢？

TreeMap和ConcurrentSkipListMap



```java
//TreeMap
Map<String, String> map = new TreeMap<>();
AbstractMap<String, String> abstractMap = new TreeMap<>();
SortedMap<String, String> sortedMap = new TreeMap<>();
NavigableMap<String, String> navigableMap = new TreeMap<>();
TreeMap<String, String> treeMap = new TreeMap<>();

//ConcurrentSkipListMap
Map<String, String> m = new ConcurrentSkipListMap<>();
AbstractMap<String, String> am = new ConcurrentSkipListMap<>();
SortedMap<String, String> sm = new ConcurrentSkipListMap<>();
NavigableMap<String, String> nm = new ConcurrentSkipListMap<>();
ConcurrentMap<String, String> cn = new ConcurrentSkipListMap<>();
ConcurrentNavigableMap<String, String> cnm = new ConcurrentSkipListMap<>();
ConcurrentSkipListMap<String, String> cslm = new ConcurrentSkipListMap<>();
```

Java集合类API-扩展性

- 如何给集合基本操作添加额外行为？（增加时候通知，删除时候日志，计数……）
  - AOP(Paper Recommend: Design Pattern Implementation in Java and AspectJ)
  - 结构型
  - 行为型

API Design

- ThreadLocal

```
// Broken - inappropriate use of String as capability.
// Keys constitute a shared global namespace.
public class ThreadLocal {
    private ThreadLocal() { } // Non-instantiable

    // Sets current thread's value for named variable.
    public static void set(String key, Object value);

    // Returns current thread's value for named variable.
    public static Object get(String key);
}
```

```
public class ThreadLocal {
    private ThreadLocal() { } // Noninstantiable

    public static class Key { Key() { } }

    // Generates a unique, unforgeable key
    public static Key getKey() { return new Key(); }

    public static void set(Key key, Object value);
    public static Object get(Key key);
}
```

- Works, but requires boilerplate code to use

```
static ThreadLocal.Key serialNumberKey = ThreadLocal.getKey();
ThreadLocal.set(serialNumberKey, nextSerialNumber());
System.out.println(ThreadLocal.get(serialNumberKey));
```

```
public class ThreadLocal {
    public ThreadLocal() { }
    public void set(Object value);
    public Object get();
}
```

```
public class ThreadLocal<T> {
    public ThreadLocal() { }
    public void set(T value);
    public T get();
}
```

- Removes clutter from API and client code

```
static ThreadLocal serialNumber = new ThreadLocal();
serialNumber.set(nextSerialNumber());
System.out.println(serialNumber.get());
```

API Design

- GuavaCache
  - 三层认识：Cache不是Map ->Cache是Map -> Cache不是Map
  - LinkedHashMap-> ConcurrentLinkedHashMap -> ReferenceMap -> ReferenceCache -> MapMaker -> CacheBuilder+LoadingCache
- Unix file I/O
  - PublicService#getPublicInfo(String langType, String hostUrl) 演进之路
    - Improve : getPublicInfo(Language language, String hostUrl) -》 Thick API

A Thick Interface

- **Unix file I/O:**

```
int open(const char* path, int flags, mode_t permissions) );
int close(int fd);
ssize_t read(int fd, void* buffer, size_t count);
ssize_t write(int fd, const void* buffer, size_t count);
off_t lseek(int fd, off_t offset, int referencePosition;
```

- **Hidden below the interface:**
  - On-disk representation, disk block allocation
  - Directory management, path lookup
  - Permission management
  - Disk scheduling
  - Block caching
  - Device independence

Executor,ExecutorService,Runnable机制

- *Thread, Runnable，Callable，FutureTask，到ForkJoinTask*
  - *Task而不是Thread*
- Executor, ExecutorService， TaskExecutor(Spring)
- AbstractExecutorService
- ThreadPoolExecutor, ForkJoinPool

```
public interface Executor
```

An object that executes submitted `Runnable` tasks. This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc. An `Executor` is normally used instead of explicitly creating threads. For example, rather than invoking `new Thread(new(RunnableTask())).start()` for each of a set of tasks, you might use:

```
public interface ExecutorService
extends Executor
```

An `Executor` that provides methods to manage termination and methods that can produce a `Future` for tracking progress of one or more asynchronous tasks.

```
ublic abstract class AbstractExecutorService
xtends Object
mplements ExecutorService
```

rovides default implementations of `ExecutorService` execution methods. This class implements the `submit`, `invokeAny` and `invokeAll` methods using a `RunnableFuture` returned by `newTaskFor`, which defaults to he `FutureTask` class provided in this package. For example, the implementation of `submit(Runnable)` creates an associated `RunnableFuture` that is executed and returned. Subclasses may override the ewTaskFor methods to return `RunnableFuture` implementations other than `FutureTask`.

```
public class ForkJoinPool
extends AbstractExecutorService
```

An `ExecutorService` for running `ForkJoinTask`s. A `ForkJoinPool` provides the entry point for submissio

A `ForkJoinPool` differs from other kinds of `ExecutorService` mainly by virtue of employing *work-stealin* active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when n submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, F

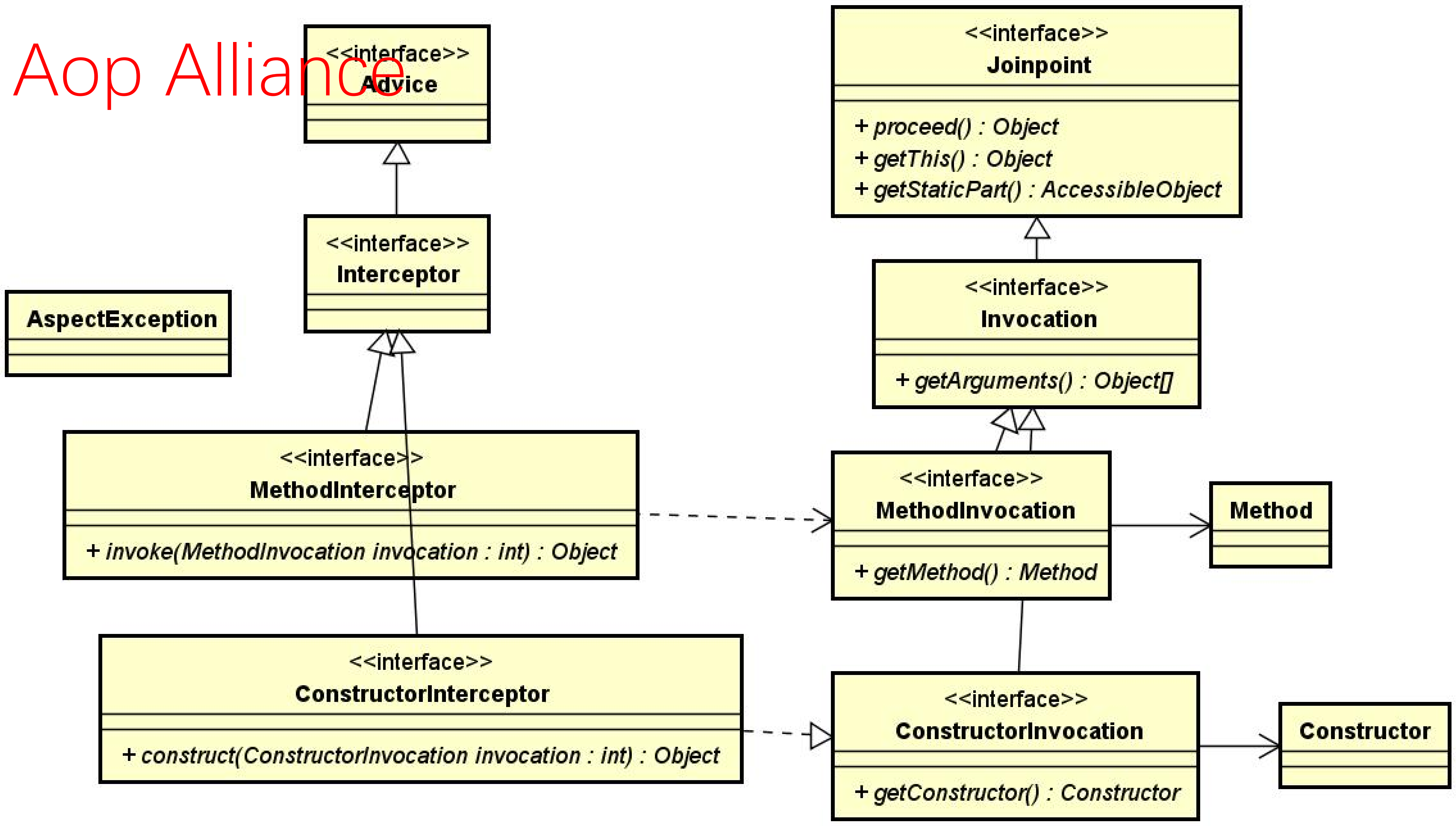
```
public class ThreadPoolExecutor
extends AbstractExecutorService
```

An `ExecutorService` that executes each submitted task using one of possibly several pooled threads, normally configured using `Executors` factory methods.

Thread pools address two different problems: they usually provide improved performance when executing large numbers of asynchronous tasks, due to reduc means of bounding and managing the resources, including threads, consumed when executing a collection of tasks. Each `ThreadPoolExecutor` also maintains tasks.

To be useful across a wide range of contexts, this class provides many adjustable parameters and extensibility hooks. However, programmers are urged to use

Aop Alliance



6. 软件设计启示录

启示录简介

- 1 Joshua Bloch： Bumper-Sticker API Design
- 2 OOD经验原则总结
- 3 GRASP-RDD
- 4 A checklist for design reviews
- 5 John Ousterhout : 软件设计哲学原则
- 6 Bruce Eckel： On Java 8编程指南
- 7 设计模式checklist

参考书籍-启示来源

- 1 面向对象分析与设计第三版-Grady Booch （OOAD）
- 2 软件设计哲学（Software Design）
- 3 领域驱动设计，实现领域驱动设计，领域驱动设计精髓（Domain，Bussiness Modeling）
- 4 面向对象启思录（OOD，Design）
- 5 重构 （Coding）
- 6 Effective Java3（Excellent Book！！！）
- 7 实现模式（Coding）
- 8 UML模式与应用第三版-Craig Larman（Analysis&Design）

Thank you!

- xiaozhiliaoo@gmail
- https://twitter.com/xiaozhiliaoo
- https://github.com/xiaozhiliaoo
- https://github.com/xiaozhiliaoo/my-slides