# Knowledge Sharing

### OOP

#### 基本概念

|  |  |
| --- | --- |
| **Abstraction** | 隐藏细节，暴露功能。(面向接口编程)  Hiding internal details and showing functionality |
| **Inheritance** | 重用基类的属性和方法。  create a class based on an existing one, reuse the fields and method of that class. |
| **Encapsulation** | 将类的属性设置成私有，通过类的公有方法访问这些属性。  Keep fields private in a class, then provide public methods to access those fields. |
| **Polymorphism** | 多种形态，包括overload 和 override. |

#### 类&对象&接口

* **类(Class)**是构建对象的模板或蓝图。
* **对象**的三个主要特征：
* 对象的行为(behavior), 即方法, 对象可以完成哪些操作。
* 对象的状态(state, 即属性或字段)）。
* 对象的标识(identity), 区分对象。(不是所有对象都有标识)
* **对象(object) 和 对象的类(class) 和 对象的类型（type）**

Person jack = new Employee(“Jack”); // Person是Employee的超类/父类/基类

1. new 操作创建了一个Employee类的对象，返回一个引用并赋值给Person类型的变量jack。
2. 对象与对象变量之间的关系，对象变量引用这个对象，但对象变量本身不是对象。
3. 对象变量存储在JVM内存模型的栈（stack）中, 而对象本身存储在JVM内存模型的堆(Heap)。
4. 对象的类(class)定义了对象(object)的内部状态和操作的实现。
5. 对象的类型(type) 决定了此对象能响应的请求的集合，也就是访问这个对象的接口。
6. 一个对象可以有许多类型，对象的类型可以是这个对象的类以及它的任意超类或接口。

* **识别类 (即建模modeling),** 如何从真实世界的问题描述中抽象出类

1. **名词建模法**，通常名词是类，动词是类的方法，形容词等描述性词汇是类的字段， 这种方式比较直观，和真实世界的概念直接反应。

比如：护士给病人打标准剂量的流感疫苗，可以识别出 Nurse, Patient, Vaccine 三个类，Nurse可以完成打疫苗的操作，也就是Nurse承担打疫苗的责任(Responsibility)。

aNurse.administrateFluVaccine(aPatient, aVaccine);

1. **动词建模法**， 找出动词是否产生有价值的过程数据。

比如：

1. 通过银行账号取钱或存钱，取钱和存钱都产生了交易记录(transaction), 所以要为交易建模。
2. 请假申请需要审批，审批产生记录，需要对 审批记录建模。
3. **也可以对描述性的词汇进行抽象建模。**

比如，策略模式中可以对类型码进行抽象，员工(Employee)的薪资策略和他们的岗位相关，可以对岗位类型进行抽象。

* **类与类之间的关系(从语义semantics上讲)**
* **依赖(“use-a”),** 订单与账户(Order & Account), 订单需要通过账户查看客户信用。
* **聚合(“has-a”),** 订单与订单项(Order & OrderItem)。
* **继承(“Is-a”) ,** 子类和父类的关系是 Is-a relationship, a safe door is a door

类与类之间的四种关系的定义：<https://blog.csdn.net/u012207345/article/details/72933691>

* 依赖(Dependency)关系是类与类之间的联接。依赖关系表示一个类依赖于另一个类的定义。例如，一个人(Person)可以买车(car)和房子(House)，Person类依赖于Car类和House类的定义，因为Person类引用了Car和House。与关联不同的是，Person类里并没有Car和House类型的属性，Car和House的实例是以参量的方式传入到buy()方法中去的。**一般而言，依赖关系在Java语言中体现为局域变量、方法的形参，或者对静态方法的调用。**
* 关联(Association）关系是类与类之间的联接，它使一个类知道另一个类的属性和方法。**关联可以是双向的，也可以是单向的。**在Java语言中，关联关系一般使用成员变量来实现。
* 聚合(Aggregation) 关系是关联关系的一种，是**强的关联关系。聚合是整体和个体之间的关系。**例如，汽车类与引擎类、轮胎类，以及其它的零件类之间的关系便整体和个体的关系。与关联关系一样，聚合关系也是通过实例变量实现的。但是关联关系所涉及的两个类是处在同一层次上的，而在聚合关系中，两个类是处在不平等层次上的，一个代表整体，另一个代表部分。
* 组合(Composition) 关系是关联关系的一种，是**比聚合关系强的关系。它要求普通的聚合关系中代表整体的对象负责代表部分对象的生命周期，组合关系是不能共享的。**代表整体的对象需要负责保持部分对象和存活，在一些情况下将负责代表部分的对象湮灭掉。代表整体的对象可以将代表部分的对象传递给另一个对象，由后者负责此对象的生命周期。换言之，代表部分的对象在每一个时刻只能与一个对象发生组合关系，由后者排他地负责生命周期。部分和整体的生命周期一样。
* **接口**
* 接口是行为/功能集合的抽象，定义了行为规范。
* 接口是一种契约(Contract) 或定义了通信的协议。
* 接口可以用于定义类型。
* 无任何抽象方法的接口可以看做标识。比如Spring Framework中Aware 接口。
* **抽象类 vs 接口**
* 语法(Syntax)的不同
* 语义(Semantics)的不同（或设计的不同）， 子类和超类是 **Is-a** 的关系；而接口是定义某种功能，实现某个接口的类，具有接口中定义的功能。
* 面向接口编程， 这里的接口指的是超类类型(父类或接口)。

#### 对象建模

将业务领域中的概念抽象成对象，避免设计出只有数据的贫血对象，对象设计的核心为：

* **角色与职责**

1. 行为的协作者
2. 信息专家模式
3. 单一职责原则

* **抽象**

1. 提炼行为特征

2. 封装变化

还要从重用(Reusable)、解耦(Decoupling)提高灵活性(Feasible)来应对变化等角度考虑设计。

#### 面向对象的设计原则

* 封装变化
* 多用组合，少用继承
* 针对接口编程，不针对实现编程
* 为交互对象之间的松耦合设计而努力
* 类应该对扩展开放，对修改闭合 – 开闭原则
* 依赖抽象，不要依赖具体类 – 依赖倒置原则
* 只和朋友交谈 – 迪米特法则
* 别找我，我会找你
* 类只有一个改变的理由 – 单一职责原则

**SOLID 原则 + 其他原则**

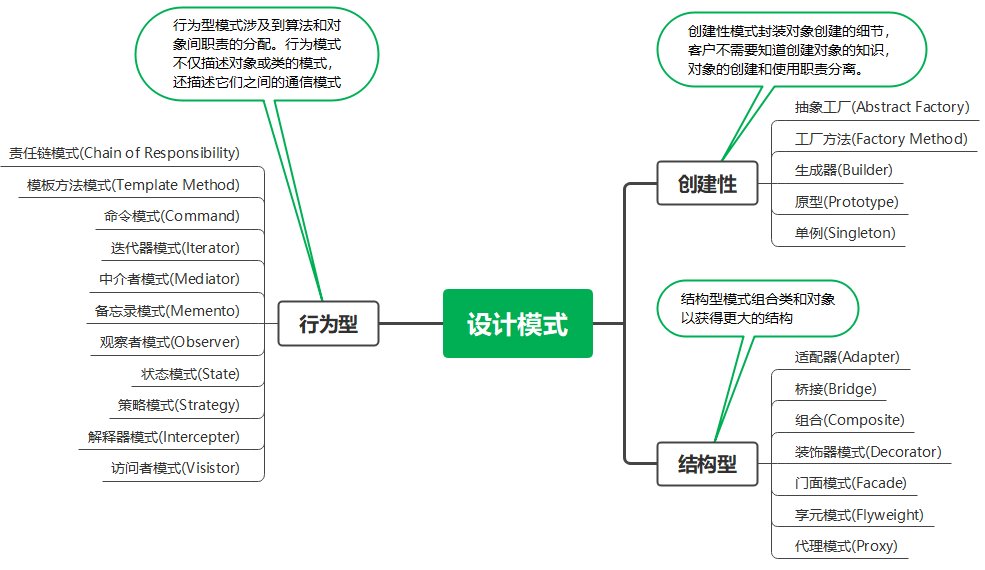
|  |  |
| --- | --- |
| **单一职责原则**  **(Single Responsibility)** | 一个类只有一个引起它变化的原因。 |
| **开发闭合原则(Open-Closed)** | 对扩展是开放的，对更改是封闭的。 |
| **Liskov替换原则**  **(Liskov Substitution)** | 子类型必须能够替换它的父类型，这是类继承设计原则，类继承的is-a关系是就行为而言的。违背此原则的例子：  1. 功能退化的子类违背里氏替换原则。2.子类共有的行为，没有被提炼到父类。3.长方形和正方形 |
| **接口隔离原则(Interface Segregation)** | 接口的设计原则，类的接口要是高内聚的。不要强迫客户依赖它们不用的方法。不要接口污染。 |
| **依赖倒置原则**  **(Dependency Inversion)** | 高层模块不依赖于底层模块，二者都依赖于抽象，面向接口编程  每个高层次都为它所需要的服务声明一个抽象接口，较低的层次实现这个接口，每个高层次通过接口依赖低层次。 |
| **迪米特法则 /最少知识原则** | 一个类对于其他类知道的越少越好，就是说一个对象应当对其他对象有尽可能少的了解。Only talk to your immediate friends，即只与“直接的朋友”通信。应用到面向对象的程序设计中时，可描述为 "类应该与其协作类进行交互但无需了解它们的内部结构"。一个对象或方法只能调用对象自身，传入的对象或内部创建的对象。 |
| **Tell, Don't Ask** | 最好是向对象发出执行某些操作或逻辑的命令，而不是查询其状态，然后采取某些行动。 |
| **KISS Keep it simple, stupid.** | 不要过度设计 |
| **YAGNI**  **You aren't gonna need it** | 不要过度设计 |
| **DRY Don't repeat yourself** | 避免重复。 |
| **信息专家模式** | 如果某个类拥有完成某个职责所需要的所有信息，那么这个职责就应该分配给这个类来实现。信息的持有者即为操作该信息的专家，优先考虑将信息相关的行为分配给这些信息的持有者。DDD聚合的设计就符合信息专门模式。 |
| **奥卡姆剃刀原理** | 原理称为“如无必要，勿增实体”，即“简单有效原理”。 |

#### 设计模式 – 可复用的面向对象设计

复用机制包括：

* 类继承
* 对象组合
* 泛型

面向对象系统中功能复用的两种技术是类继承和对象组合，优先使用对象组合，委托(delegation) 是对象组合的一种方式。



注：除了上面介绍的23种设计模式之外，还有其他模式，如规格模式(Specification)

Java & Spring中的模式：

|  |  |
| --- | --- |
| 装饰器模式 | FilterInputStream  Collections.SynchronizedXXX  Collections.UnmodifiableXXX |
| 观察者模式 | Observer  Observable |
| 迭代器模式 | Iterator |
| 原型模式 | Object#clone() 浅克隆，深克隆 |
| 责任链模式 | Spring 的拦截器 |
| 代理模式 | JDK的动态代理 Proxy和CGLIB代理, CGLIB用于Spring AOP，JDK中的proxy必须基于接口，CGLIB却没有这个限制。 |

### Java Foundation(v11)

#### Equal operator (==) & equals method

== 和 equals()方法的区别？

判断下面的代码中 s1, s2, s3 and s4 的相等性， == or equals？

|  |
| --- |
| String s1 = “abc”;  String s2 = “ab” + “c”;  String tmp = “ab”;  String s3 = tmp + “c”;  String s4 = new String(“abc); |

#### Object Class

Object是属于类的基类，它有哪些方法？

为什么Java规范要求定义一个类是重新hashCode()和equals()方法？

两个对象的hashCode相等，是否这两个对象相等？

两个对象相等，是否它们的hashCode相等

#### Generic

Generic有哪些制约条件？

#### Thread

线程的实现方式

线程的状态，一个启动的线程可否再启动？

线程池，如何创建线程池？

创建多线程的数量的基准？

Java多线程的内存模型

Servlet的工作模式是单实例多线程

#### Thread-safe

* 什么是线程安全？
* Java中线程安全的, 如类String, Integer, Vector, Hashtable, ConcurrentHashMap, StringBuffer, Java 8新增加的时间和日期的类等。
* 什么样的类是线程安全的？
* Stateless class 无状态的类
* Immutable class 不可变的类
* Synchronized 访问类的属性/字段的方法实现同步
* Lock 对访问类的属性/字段加锁
* 使用Java中线程安全的类作为类的字段

#### Collection APIs

1. List -> Vector, ArrayList, LinkedList
2. Set -> HashSet, TreeSet
3. Map -> HastMap, Hashtable
4. Concurrent collection – ConcurrentHashMap

问题

为什么HashSet中不能包括重复的对象？

#### IO APIs

1. Byte Stream
2. Char Stream

#### Exception

1. Checked exception
2. Unchecked exception

#### Lambda Expression

1. Lambda expression 代表一个函数，由arguments（或者no arguments），body, 或 Arrow Token组成。（函数式编程中函数式一等公民）

2. Lambda expression 提供了functional interface的一种实现方法，lambda expression 返回一个functional interface的一个实例。e.g.:

|  |
| --- |
| //变量类型必须是 functional interface  Runnable runnable = () -> System.out.println("hello, world"); |

3. 只有一个抽象方法的接口叫函数式接口（functional interface），lambda expression 只能为一个方法提供body, 所以functional interface 只能有一个抽象方法。

4. 因为functional interface中的抽象方法没有声明异常，所以lambda expression不能直接处理checked exception, 需要将checked exception转换成unchecked expression。网上有例子如何避免在lambda expression中使用try-catch block

5. Lambda expression可以使用本地变量，本地变量必须是final的，或事实不可改变的， 也就是初始化后不能再一次在lambda expression外赋值。

6. Lambda expression的好处，

1）减少代码量。

2）可以方便的处理大数据集。

3）可以方便的利用多核CPUs, 比如Stream API提供的并行操作。

#### Stream API

1. Collection用于存取数据，Stream用于并行或串行操作数据。
2. Stream operations 分为中间操作(intermediate operations)和终结操作(terminal operations); intermediate operations 如filer, map, flatMap, sort, distinct 等，用于转换一个stream到另外一个stream; 而终结操作如reduce, collect, sum, average, max, min, forEach, anyMatch, findAny 等计算出一个结果。
3. Stream oeprations 可以组成Stream 流水线， stream pipeline由 一个 source, 零个、1个或多个 中间操作 和一个终结操作组成，中间操作将不执行，直到终结操作被调用。
4. 一个Stream对象被调用后，不能再次被调用, 也就是stream 对象只能被消费一次。
5. Stream的数据源source可以来自 Collection, Array, IO channel 或者Stream 自身的创建者。

E.g.: Arrays.stream(new String[]{"a","b","c"});

IntStream.rangeClosed(1, 10);

1. 并行操作的stream, 例如: employees.parallelStream()。

#### 类加载机制

类加载机制 （javap -c \*.class 生产可读的jvm指令)

**步骤： 加载(load), 链接(link), 初始化(Initialize)**

* Load: 将二进制字节码装载到JVM中， 类的全限定名+ClassLoader完成加载
* Link: 校验， 初始化静态变量赋予默认值，解析类中的调用的接口、类 （NoSuchMethodError, NoSuchFileError), 包括： 校验，准备， 解析(可选）。
* Initialize: 执行类中的静态代码，构造器代码，静态属性的初始化。

触发初始化有: new , 反射调用类的方法， 子类调用初始化，JVM启动指定的初始化类。

加载和链接 将二进制字节码转换为 Class 对象；

初始化 是 在初次主动使用对象前执行， 给静态变量赋值和调用<clinit>()等

#### Class Loader 类加载顺序

* Bootstrap class loader
* Extension class loader
* System class loader
* Custom class loader

#### Memory Leak 内存泄漏

**内存泄漏**：程序未能释放不再使用的存储。

JVM在为新的对象分配空间时，内存不足报OutOfMemoryError异常

#### Garbage Collection 垃圾回收

需要了解垃圾回收机制

下面的代码执行完中，第一行创建的Employee对象可否被垃圾回收？

|  |
| --- |
| Employee jack = new Employee(“Jack”);  List<Employee> employees = new ArrayList();  Employees.add(jack);  jack = null; |

### RESTful API Design

OAS 3.0 <https://swagger.io/specification/v3/>

<https://swagger.io/specification/>

example:

<https://editor.swagger.io/?_ga=2.182741158.1248925185.1671853933-1005930888.1670039681>



<https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>

* HTTP Method：GET, POST, PUT, PATCH, DELETE
* HTTP status： 2XX, 4XX, 5XX
* URI design: e.g: /customers/1/orders
* JSON Response design
* RESTful Framework:
* Spring RESTful framework
* Jersey

### Spring Framework

Spring Framework提供企业级应用提供轻量级的解决方案。它包括的组件/特性有：

Spring Core (IoC, DJ, AOP …)

Spring MVC

Data Access

…

#### Spring IoC Container (IoC容器)

Spring IoC container 负责Spring Bean的创建、装配、初始化和管理Bean的整个生命周期。

好处是职责分离，开发者只需要关心Bean使用。

由Spring IoC container管理的类/对象称为Spring Bean。

BeanFactory 和 ApplicationContext可以代表Spring IoC容器，注意它们之间的区别。

问题:

如何获取已经注册的Bean?

#### Dependency Injection

依赖倒置原则，面向接口编程，对象之间的解耦(Decouple)。

@Autowired 注解

* 注入方式，优先考虑哪种方式？
  + 基于构造器的方式
  + 基于Setter方法的方式
* 通过类型type或名称name查找bean，缺省是通过类型。
* 实际注入是通过BeanPostProcessor执行的, **AutowiredAnnotationBeanPostProcessor, @Vaue注解也是如此。**
* **@Qualifier 注解，**如果同一个类型有多个Bean实例，可以通过此注解指定注入的Bean。
* @Primary注解

问题:

@Autowired @Value 在什么时候执行？

#### Spring Bean Life Cycle生命周期

Spring IoC container 负责Spring Bean的创建、装配、初始化和管理Bean的整个生命周期。

1. 创建对象（实例化）。
2. 依赖注入。
3. 初始化bean（Initialization）可以通过实现 InitializingBean接口，或者在初始化的方法上加入@PostConstruct注解.
4. 执行bean 后置处理器 （PostBeanProcesser）bean，它有两个方法:
   1. postProcessBeforeInitialization 在第3步之前执行
   2. postProcessAfterInitialization 在第3步之后执行
5. Spring bean在销毁前，可以通过Bean实现DisposableBean接口 或 在bean的一个方法上加上@PreDestroy注解 去释放一些资源。

#### Spring Bean Scope

1. Singleton scope bean在Spring IoC Container 启动的时候创建对象。默认Scope是 singleton, 如果加上@lazy注解，则在调用时创建对象。一个Spring IoC容器只有一个实例。
2. Prototype scope bean在调用的时候创建对象。一个Spring IoC容器可以有多个实例。
3. As a rule, you should use the prototype scope for all stateful beans and the singleton scope for stateless beans. 通常，你应该对有状态的bean（不是线程安全的有状态的Bean）使用prototype scope, 而对无状态的Bean（线程安全的Bean）使用Singlton scope.
4. Spring Singleton Bean 与Singleton设计模式

两者概念不同，

**Spring Singleton Bean** 是Spring IoC容器只为一个Bean创建一个实例，缓存起来，当客户端发起多个请求(多线程) 并发访问同一个Bean，Spring IoC容器不保证这个bean在多线程条件下是线程安全的。换句话说，Bean的类定义时需要考虑线程安全。Spring Singleton Bean 不一定是Singleton class。

**Singleton Design Pattern** 保证一个类仅有一个实例，并提供一个访问它的全局访问点。

#### Spring Properties

applicaton.properties/ application.yml – 默认Spring属性文件

可以通过属性spring.profiles.active为不同环境(dev,test, prod) 指定Spring 属性的配置文件。

如果某个属性在profile指定的属性文件中不存在，就使用在默认属性文件中的配置。

spring.profiles.active 也可以配置命令行。

#### Spring AOP

常用于Transaction Management, Logging, Security

问题:

如何使用AOP?

#### Spring Caching

Spring提供了Caching抽象，可以支持Redis or EhCache等cache存储。

* Caching注解
* Caching 管理器接口CacheManager
* Caching 配置

Spring Data Redis项目 提供了CacheManager的实现RedisCacheManager。

#### Spring Framework的回调(callback)接口

Spring framework 中定义了很多回调（callback） 接口，你可以定义一个类实现这些接口, 注册成bean后，Spring IoC会 自动侦测到这些bean，自动调用它们的方法， 这样大家可以做一些业务逻辑处理。

**常见的需要掌握的回调接口有：**

* BeanPostProcessor
* Aware接口的的子接口如 ApplicationContextAware, BeanFractoryAware
* ApplicationListener
* ApplicationRunner
* CommandLineRunner
* InitializingBean
* DisposableBean
* WebMvcConfigurer
* HandlerInterceptor

问题:

在Spring IoC container启动后就自动从数据库中获取数据，如何实现？

### Spring Boot

Spring Boot让你开发基于Spring Framework的应用更容易，因为：

* Auto-configuration
* Starter dependencies
* Embedded application server like Tomcat, Jetty

要知道Spring boot 2.0 & Spring boot 3.0区别。

问题1:

为什么往往用户自定义的Bean能替换自动配置的bean?

问题2:

如何自定义一个starter?

问题3:

常用的Spring Boot 注解

### Spring Cloud

Spring Cloud tools are used to build distributed systems.

### Microservice

1. Advantages and disadvantages
2. How to break a Monolith into Microservices 如何划分微服务？

<https://martinfowler.com/articles/break-monolith-into-microservices.html>

1. How to integrate microservices? RESTful API, MQ, RPC

### Test (Code Quality)

Ensure the code quality by code review and testing tools:

* Code review
* Junit：Junit is a testing platform on the JVM
* JMeter: performance test
* Selenium: Java web test
* Postman: Postman is an API platform for building and using APIs
* SolarQube: the leading tool for continuously inspecting the Code Quality and Security of your codebases and guiding development teams during Code Reviews.
* Mock: 测试工具有 Mockito、JMock、EasyMock等等，SpringBoot 目前默认的测试框架是 Mockito 框架。

### Data Security

#### JWT

#### Spring Cloud Gateway + JWT + Role-based authorization (RBAC)

Authentication: 认证

Authorization: 授权

#### OAuth2

#### SSL

#### Encrypt sensitive data

#### Prevent code Injection

#### SSO

### Database (RDS)

**SQL:** DDL, DML, DCL statements

**SQL Tuning**

### Deployment & Monitoring

#### CICD – Jenkins Pipeline

#### Zero downtime

* 蓝绿部署
* 灰度发布(A/B测试)
* 滚动发布

#### Managing and monitoring Spring Boot applications

#### APM – Application Performance Monitoring

### Cloud

#### Cloud providers:

AWS, GCP, Azure, Alibaba, etc.

#### Types of cloud services

IaaS, PaaS, SaaS

### 学习书籍推荐

**《重构》**

**《设计模式》**

**《Head First设计模式》**

**《重构与模式》**

《敏捷软件开发原则、模式与实践》

**《Effective Java中文版》**

《Java并发编程实战》

《阿里巴巴Java开发手册 第二册》

《Java 8实战》

《实现领域驱动设计》

**《解构领域驱动设计》**

《中台架构与实现 – 基于DDD和微服务》

《互联网时代的软件革命SaaS架构设计》

《数据仓库工具箱 第3版 维度建模权威指南》

《大数据之路大数据实践 阿里巴巴大数据实践》

**B站的免费视频， 例如 尚硅谷Spring/ Spring Boot/ Spring Cloud/ 大数据**

### 学习方法

* 费曼学习法
* 思维导图学习法