**Java知识整理**

1. **面向对象的思想**
   1. **基本概念**
      1. **封装**

将客观事物封装成抽象的类，这些类可以把自己的数据与方法让可信的类或对象操作，对不可信的类进行信息隐藏。

* + 1. **继承**

使用现有类的所有的功能，在无需重新编写原来的类的情况下对这些功能进行拓展。Java中只支持类的单继承性。一个类只能有一个直接的父类。

* + 1. **多态**

一个类的实例的相同方法在不同情况下有不同的表现形式。最常见的多态就是将子类传入父类参数中，运行时调用父类方法时通过传入的子类决定具体的内部结构或行为。（子类B继承父类A，声明使用父类A而创建使用子类B）

* 实现多态的三个条件

继承

重写（子类对象重写父类的方法）

父类引用指向子类对象

* 多态在父子类中的方法存在的三种情况：

1. 父类中有而子类中没有的方法，使用多态时会直接调用父类的方法。
2. 父类中有而子类中对父类的方法进行了重写，这时候调用的时子类的方法。
3. 父类中没有而子类中有的方法，这时调用会出错。
   1. **设计原则**
      1. **单一职责原则**

类的功能要单一。

* + 1. **里氏替换原则**

子类可以替换父类，出现在父类出现的任何地方（多态）。

* + 1. **依赖倒转原则**

高层次模块不应该依赖低层次的模块，他们应该依赖于抽象，抽象不应该依赖于具体实现，具体实现应该依赖于抽象。

* + 1. **迪米特法则（最少知道）**

一个对象应该对其他对象尽可能少的了解。

* + 1. **聚合复用原则**

对象委托优于继承，在一个新的对象中尽量通过持有其他对象来整合功能，避免使用继承。

* + 1. **开放封闭原则**

对拓展开放，对修改封闭。

* + 1. **接口隔离原则**

采用多个于特定客户类有关的接口比采用一个通用的接口好。接口功能要单一。

1. **Java重点基础知识**
   1. **java常用类、接口**

java.lang.Object

所有java类的基类。包含的方法：

Object clone()：克隆一个与该对象相同的类的对象。

java.lang.Thread

java.lang.String

java.lang.StringBuilder/StringBuffer

java.lang.System

java.lang.Arrays

java.util.Arrays

java.util.Scanner

* 1. **java IO**
     1. **BIO体系**
     2. **BIO分类**
     3. **BIO基本用法**
     4. **NIO基本用法**
     5. **开源IO工具**
  2. **Java多线程**
     1. **一些概念**
     2. **线程基本用法**
     3. **同步**
     4. **死锁**
     5. **synchronized与volatile**
     6. **锁**
     7. **JUC**
  3. **Java网络编程**
     1. **一些概念**
     2. **Socket基本用法**
     3. **爬虫原理**
     4. **Netty**
  4. **Java集合**
     1. **泛型**
     2. **集合体系结构**
     3. **基本用法**
     4. **java.util.concurrent包下的容器**
  5. **Java异常**
     1. **异常体系结构**

异常：程序运行过程中出现的错误。

Java中所有的异常都继承自抽象类java.lang.Throwable。

子类Error是虚拟机内部错误，如内存溢出错误（OutOfMemoryError）、线程终止错误（ThreadDeath）。这种错误程序无法处理，一般由虚拟机终止进程。

子类Exception是程序可以处理的异常。这种异常分为运行时异常（RuntimeException）和非运行时异常（又称检查异常CheckedException）。运行时异常可以不进行捕获，也可以进行捕获，非运行异常必须进行捕获处理。

常见异常：

ArithmeticException 算数异常

ArrayStoreException         试图将错误类型的对象存储到一个对象数组时抛出的异常

NullPointerException      空指针异常

NumberFormatException 数值强转异常

IndexOutOfBoundsException  数组下标越界异常

IllegalArgumentException   抛出的异常表明向方法传递了一个不合法或不正确的参数



* + 1. **异常处理**

java中的异常处理有两种方式：

1. 使用try{}catch(Exception e){}finally{}语法对异常进行捕获

程序首先进入try语句执行，如果没有异常则执行return之前会先执行finally中的语句（如果finally中存在return则不会再执行try中的return），

1. 在方法签名上使用throws将异常抛出，由调用者处理。

throw与throws的区别

throw 用于在方法体中抛出异常

throws 用在方法签名上，用来标识该方法可能抛出异常。

* + 1. **异常处理原则**
* 避免使用异常处理代替错误处理
* 处理异常不可代替简单测试
* 不要进行小粒度的异常处理，影响程序可读性
* 异常往往在高层处理，由调用者处理，层层外抛
  1. **Java反射**
     1. **反射的作用**
     2. **反射的基本用法**
     3. **自定义注解**
     4. **JDK与CGLIB动态代理**
     5. **Java动态编译**
  2. **JVM**
     1. **JVM内存结构**
     2. **Java类加载机制**
     3. **垃圾回收**
     4. **JVM监控工具**
     5. **JVM优化**
  3. **其他**
     1. **关键字**
     2. **JDK8新特性**
     3. **JDK9新特性**

1. **设计模式**
   1. **总览**
   2. **单例模式**
2. **协议**
   1. **TCP**
   2. **UDP**
   3. **HTTP**
   4. **RPC**
3. **框架**
   1. **mybaits**
   2. **spring**
   3. **spring mvc**
   4. **spring boot**
   5. **spring cloud**
   6. **dubbo**
   7. **k8s**
4. **微服务架构**
   1. **概念**
   2. **微服务基本组件**
   3. **Spring cloud netflix**
   4. **Spring cloud**
5. **持续集成持续交付**
   1. **一些概念**

管道：将源代码转换为可以发布产品的多个不同的任务（task）和作业（job）通常串联成为一个软件“管道”，一个自动化流程成功后会启动管道中的下一个流程。管道的工作之一就是快速处理变更。

快速失败（fail fast）：快速失败值的是在管道流程中尽快发现问题并快速通知用户的方式，这样可以及时修正问题并提交代码以便使管道再次运行。

* 1. **持续集成（continuous interagtion，CI）**

持续集成是在源代码变更后自动检测、拉取、构建和进行单元测试的过程。

持续集成基本思想：让一个自动化程序监测一个或多个源代码仓库是否又变更，当变更推送到仓库时，它会监测到变更、下载副本、构建并进行单元测试。持续集成中会使用jenkins这样的工具。

* 1. **持续交付（continuous Derlivery， CD）**

持续交付通常是指整个流程链，他自动检测源代码变更并通过构建、测试、打包和相关操作运行以生成可部署的版本，基本上不需要任何人为干预。

* 1. **maven**
  2. **jekins**
  3. **docker**