**Java知识点总结**

# Java基础

## 数据结构

## 多线程

### 线程安全问题产生的原因

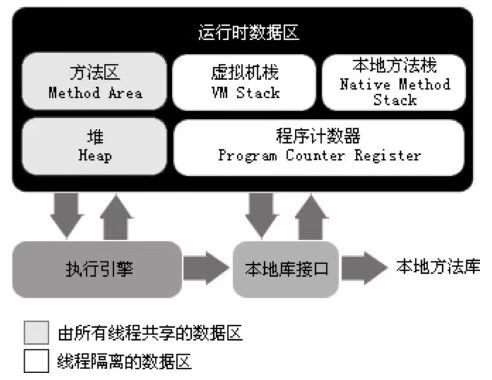
java的主内存与线程私有内存是线程安全问题产生的根本原因

## 设计模式

# Java虚拟机

## java内存模型

### 运行时内存



#### 程序计数器

是当前线程所执行的字节码的行号指示器。

#### 堆

是java虚拟机管理内存最大的一块，存放的是对象实例。几乎所有的对象实例都在这里分配内存

#### java虚拟机栈

描述的是java方法执行的内存模型。每个方法在执行的时候会创建一个栈帧，用于存储局部变量表、操作数栈、动态链接、方法出栈等信息。每一个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈出栈的过程。



##### 栈帧

每个方法执行，都会创建一个栈帧，伴随着方法从创建到执行完成。用于存储局部变量表，操作数栈，动态链接，方法出口等。

###### 局部变量表

主要用于存储方法参数和定义在方法体内的局部变量这些数据类型包括各类基本数据类型、对象引用（reference），以及returnAddressleixing。大小在编译时确定，运行期间不会更改局部变量表大小。

局部变量表中的变量只在当前方法调用中有效

#### 本地方法栈

本地方法栈与java虚拟机栈非常相似，java虚拟机栈为本地方法提供服务，本地方法栈为native方法提供服务。

##### 方法区

它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

###### 运行时常量池

运行时常量池是方法区的一部分。用于存放编译期生成的各种字面量和符号引用。相对与Class文件常量池的另外一个特征是具备动态性，指：运行期间也可以将新的常量放入池中。

#### 直接内存

避免了在Java堆和Native堆中来回复制数据。

# Dubbo

## Dubbo 原理



1. 服务端启动时会把所有接口注册到注册中心
2. 客户端启动时，订阅所需要的服务
3. 订阅内容变更时，会推送订阅的消息
4. 客户端启动时，会与服务端建立长连接，然后进行数据通信
5. 服务端，客户端启动后，后台会启动定时器，发送统计数据给monitor
6. 服务启动的时候，provider和consumer根据配置信息，连接到注册中心register，分别向注册中心注册和订阅服务
7. register根据服务订阅关系，返回provider信息到consumer，同时consumer会把provider信息缓存到本地。如果信息有变更，consumer会收到来自register的推送
8. consumer生成代理对象，同时根据负载均衡策略，选择一台provider，同时定时向monitor记录接口的调用次数和时间信息
9. 拿到代理对象之后，consumer通过代理对象发起接口调用
10. provider收到请求后对数据进行反序列化，然后通过代理调用具体的接口实现

## DubboSpi

### javaSpi

#### javaSpi的机制

是一种将服务接口与服务实现分离以达到解耦、大大提升了程序可扩展性的机制。

#### 实现javaSpi

1. 定义一个接口
2. 在META-INF/services目录下，创建该接口的同名文件
3. 该文件的内容就是接口的具体实现类的全类名（可以是多个）
4. 通过ServiceLoader.load()加载实现类

#### JavaSpi的缺点

1. 不能按需加载,全部加载,浪费资源；
2. 获取某个实现类的方式不够灵活，只能通过遍历；
3. 多个并发多线程使用 ServiceLoader 类的实例是不安全的；
4. 加载不到实现类时抛出并不是真正原因的异常，错误很难定位；

### DubboSpi的增强

1. 对 Dubbo 进行扩展，不需要改动 Dubbo 的源码；
2. 延迟加载，可以一次只加载自己想要加载的扩展实现。
3. 增加了对扩展点 IOC 和 AOP 的支持，一个扩展点可以直接 setter 注入其它扩展点
4. Dubbo 的扩展机制能很好的支持第三方 IoC 容器，默认支持 Spring Bean。

### 实现dubboSpi

1. Spi文件存储路径在META-INF\dubbo\internal 目录下 文件名为接口的全路径名 接口的包名+接口名；
2. 每个spi文件里面的格式定义为： 扩展名=具体的类名，例如 dubbo=com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtoco