企业规模的软件系统该如何设计呢？在开始写代码之前，我们需要选择一个合适的架构，这个架构将决定软件实施过程中的功能属性和质量属性。因此，了解软件设计中的不同架构模式对我们的软件设计会有较大的帮助。

# 什么是架构模式

根据维基百科：架构模式是针对特定软件架构场景常见问题的通用、可重用解决方案。架构模式类似于软件设计模式，但范围更广。本文将简要解释10种常见架构模式及其用法、优缺点。

* 分层模式（Layered pattern）
* 客户端-服务器模式（Client-server pattern）
* 主从模式（Master-slave pattern）
* 管道-过滤器模式（Pipe-filter pattern）
* 代理模式（Broker pattern）
* 点对点模式（Peer-to-peer pattern）
* 事件-总线模式（Event-bus pattern）
* 模型-视图-控制器模式（Model-view-controller pattern）
* 黑板模式（Blackboard pattern）
* 解释器模式（Interpreter pattern）

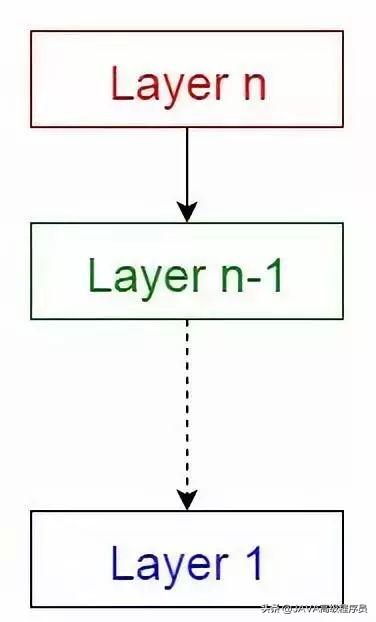
# 分层模式

此模式用于可分解为子任务的结构化程序，每个子任务都位于特定的抽象层级，每一层都为上一层提供服务。一般信息系统最常见的4个层次如下。

* 表示层(也称为UI层)
* 应用层(也称为服务层)
* 业务逻辑层(也称为领域层)
* 数据访问层(也称为持久层)

应用场景：

* 一般的桌面应用程序
* 电子商务web应用程序
* 一般的移动App



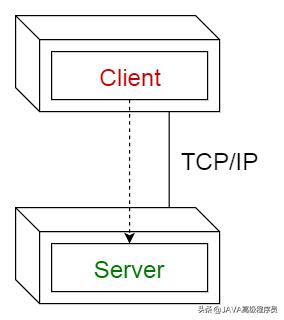
# 客户端-服务器模式

这种模式由两部分组成：服务器和多个客户端。服务器将向多个客户端提供服务。客户端从服务器请求服务，服务器向这些客户端提供相关服务。此外，服务器继续侦听客户端请求。

应用场景：

电子邮件、文档共享和银行等在线应用程序。

基于IPC的应用程序



# 主从模式

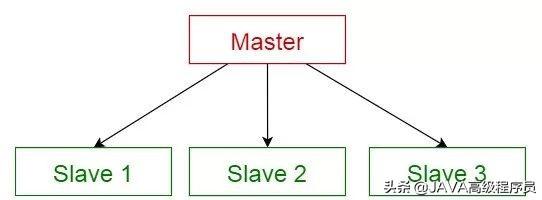
这种模式由两部分组成：主节点和从节点。主节点将工作分配给相同的从节点，并根据从节点返回的结果计算最终结果。

应用场景：

在数据库复制中，主数据库被视为权威源数据库，从数据库与之同步。

通过总线连接到计算机系统(主驱动器和从驱动器)的外围设备。

进程内的多线程应用。



# 管道-过滤器模式

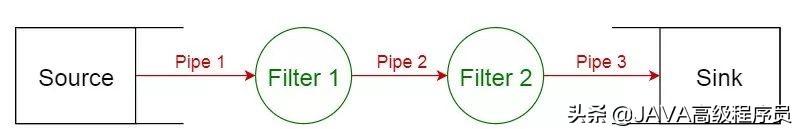
这种模式可用于构造生成和处理数据流的系统。每个处理步骤都包含一个过滤器组件。要处理的数据通过管道传递。这些管道可用于缓冲或同步目的。

应用场景：

编译器。连续过滤器执行词法分析、词法解析、语义分析和代码生成。

生物信息学的工作流

工具链式的应用程序



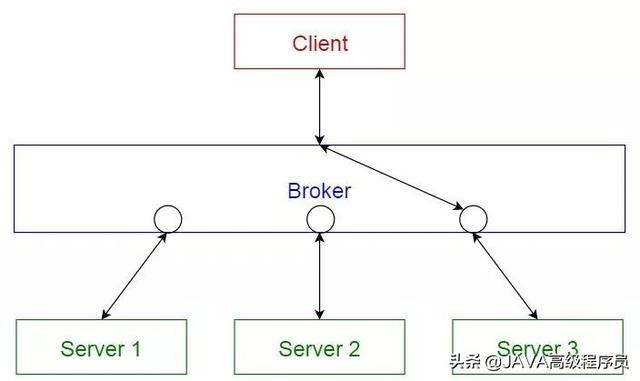
# 代理模式

这种模式通过解耦组件来构造分布式系统。这些组件可以通过远程服务调用彼此交互。代理组件负责协调组件之间的通信。服务器向代理发布功能(服务和特征)。客户端向代理请求服务，然后代理将客户端重定向到合适的服务。需要注意broker，agent，proxy以及delegate的区别。

应用场景：

消息代理软件，例如：Apache ActiveMQ、Apache Kafka、RabbitMQ和JBoss消息传递。

网络传输中的代理软件。



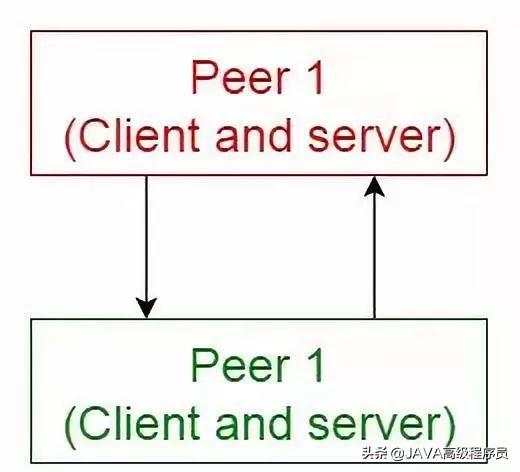
# P2P模式

在这种模式中，每个组件都称为对等节点。对等节点既可以作为客户机(从其他对等节点请求服务)，也可以作为服务器(向其他对等节点提供服务)。对等节点可以充当单个客户机或服务器，也可以同时充当客户机和服务器，并且可以随着时间变化动态地更改角色。

使用场景：

文件共享网络，例如Gnutella和G2等。

多媒体协议，如P2PTV和PDTP。



P2P模式

# 事件-总线模式

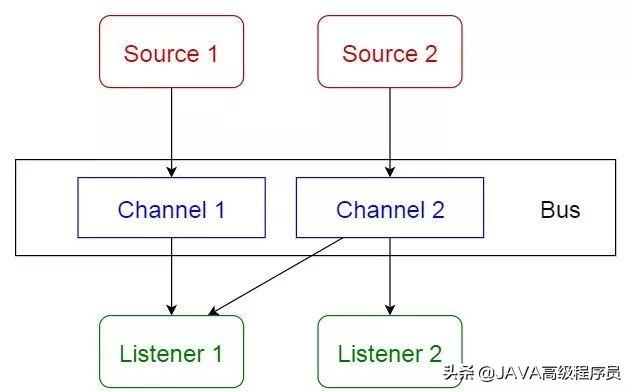
这种模式也被称为订阅发布模式，主要处理事件，有4个主要组件：事件源、事件监听者、通道和事件总线。事件源将消息发布到事件总线上的特定通道，监听者订阅特定的通道。消息发布到监听者之前订阅的通道，监听者将收到消息的通知。

使用场景：

安卓开发

通知服务

注册中心



事件-总线模式

# 模型-视图-控制器模式

这种模式，也称为MVC模式，将一个交互应用程序分为三个部分：

模型-包含核心功能和数据

视图——向用户显示信息(可以定义多个视图)

控制器——处理来自用户的输入

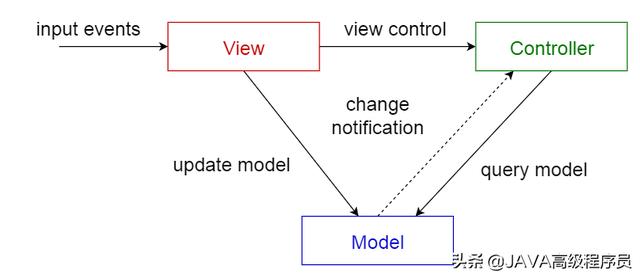
这样做是为了将信息的内部表示、信息呈现给用户的方式、接受用户输入的方式分离开来。这种模式解耦组件并允许有效的代码重用。

应用场景：

一般的web应用程序架构

Django和Rails等Web框架

一般的GUI 应用程序



# 黑板模式

这种模式对于没有确定解决方案策略的问题非常有用。黑板图案由三个主要部分组成：

黑板：一个结构化的全局内存，包含来自解决方案空间的对象

知识源：具有自己表示形式的专门化模块

控制组件：选择、配置和执行模块

所有的组件都可以到达黑板。组件可以生成添加到黑板上的新数据对象。组件在黑板上查找特定类型的数据，并通过与现有的知识源进行模式匹配找到这些数据。

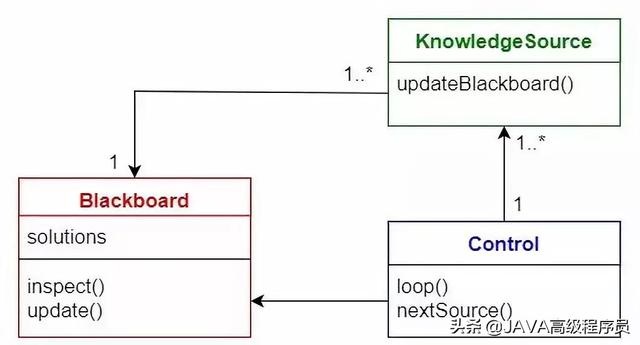
应用场景：

语音识别

车辆识别及追踪

蛋白质结构识别

声纳信号的解释



黑板模式

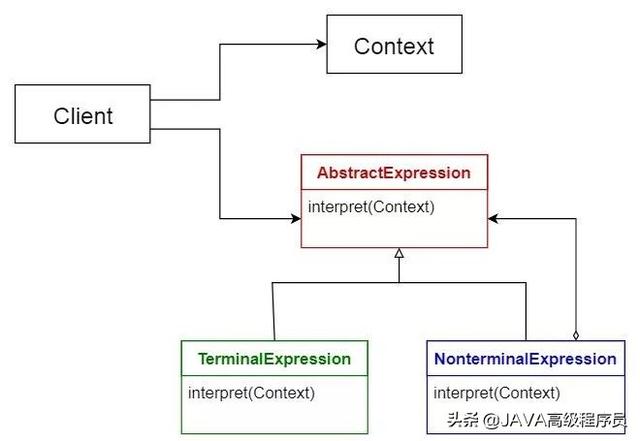
# 解释器模式

这种模式用于设计一个解释专用语言编写的程序组件。它主要指定如何评估每一行程序，即用特定语言编写的句子或表达式。其基本思想是语言的每个符号都有一个类。

应用场景：

数据库查询语言，如SQL。

用于描述通信协议的语言。



解释器模式

下面的表格总结了每种架构模式的优缺点。

