聊聊十种常见的软件架构模式

Vijini 架构精进之路 2021-04-13 08:15

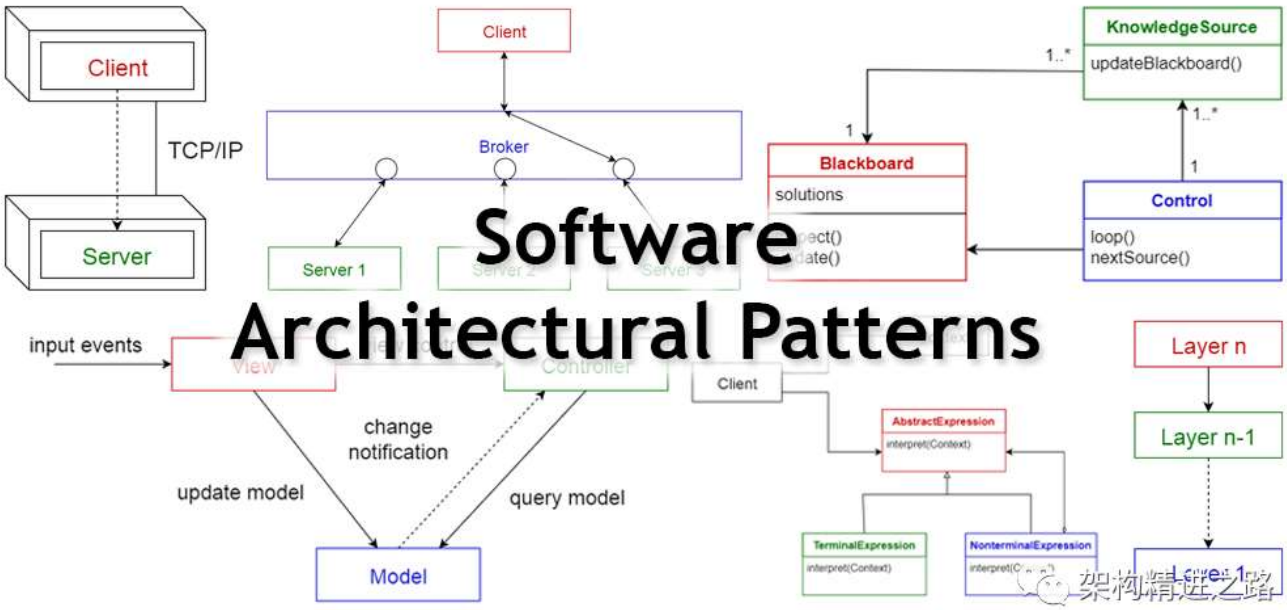
收录于合集

#系统架构应用汇总

31个

有没有想过要设计多大的企业规模系统?

在主要的软件开发开始之前，我们必须选择一个合适的体系结构，它将为我们提供所需的功能和质量属性。因此，在将它们应用到我们的设计之前，我们应该了解不同的体系结构。



什么是架构模式?

根据维基百科中的定义:

> 架构模式是一个通用的、可重用的解决方案，用于在给定上下文中的软件体系结构中经常出现的问题。架构模式与软件设计模式类似，但具有更广泛的范围。

在本文中，将简要地解释以下10种常见的体系架构模式，以及它们的用法、优缺点。

1. 分层模式
2. 客户端-服务器模式
3. 主从设备模式
4. 管道-过滤器模式
5. 代理模式
6. 点对点模式
7. 事件总线模式
8. 模型-视图-控制器模式
9. 黑板模式
10. 解释器模式

一. 分层模式

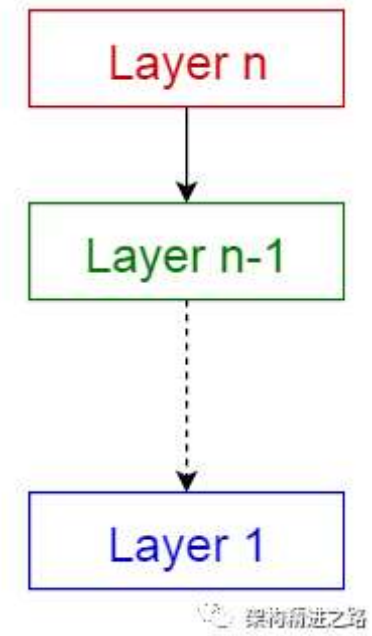
这种模式也称为多层体系架构模式。它可以用来构造可以分解为子任务组的程序，每个子任务都处于一个特定的抽象级别。每个层都为下一个提供更高层次服务。

一般信息系统中最常见的是如下所列的4层。

- 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(也称为持久化层)

使用场景：

- 一般的桌面应用程序
- 电子商务Web应用程序

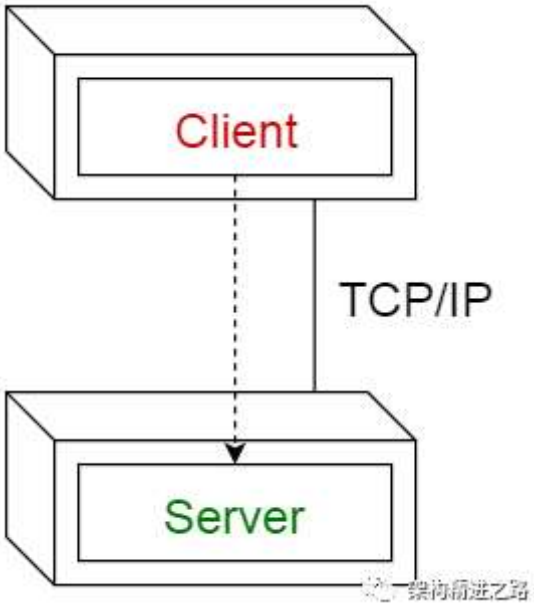


二. 客户端-服务器模式

这种模式由两部分组成：一个服务器和多个客户端。服务器组件将为多个客户端组件提供服务。客户端从服务器请求服务，服务器为这些客户端提供相关服务。此外，服务器持续侦听客户机请求。

使用场景：

- 电子邮件，文件共享和银行等在线应用程序

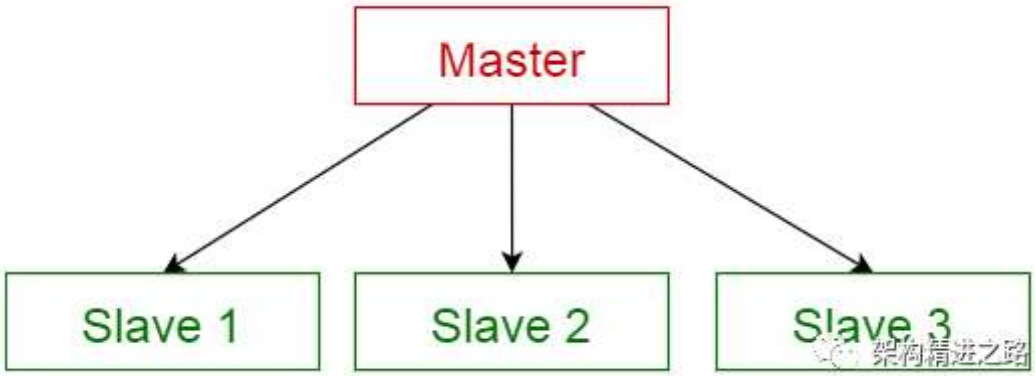


三. 主从设备模式

这种模式由两方组成;主设备和从设备。主设备组件在相同的从设备组件中分配工作，并计算最终结果，这些结果是由从设备返回的结果。

使用场景：

- 在数据库复制中，主数据库被认为是权威的来源，并且要与之同步
- 在计算机系统中与总线连接的外围设备(主和从驱动器)

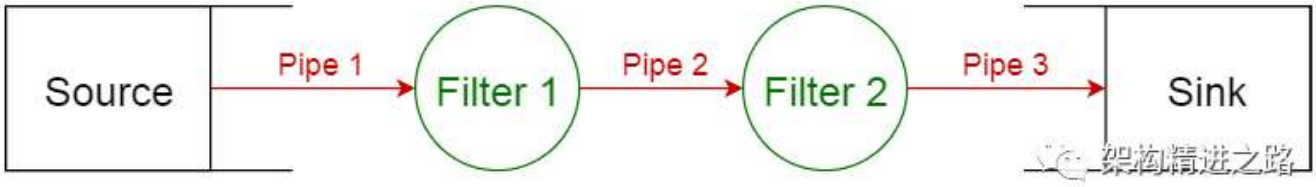


四. 管道-过滤器模式

此模式可用于构造生成和处理数据流的系统。每个处理步骤都封装在一个过滤器组件内。要处理的数据是通过管道传递的。这些管道可以用于缓冲或用于同步。

使用场景：

- 编译器。连续的过滤器执行词法分析、解析、语义分析和代码生成
- 生物信息学的工作流



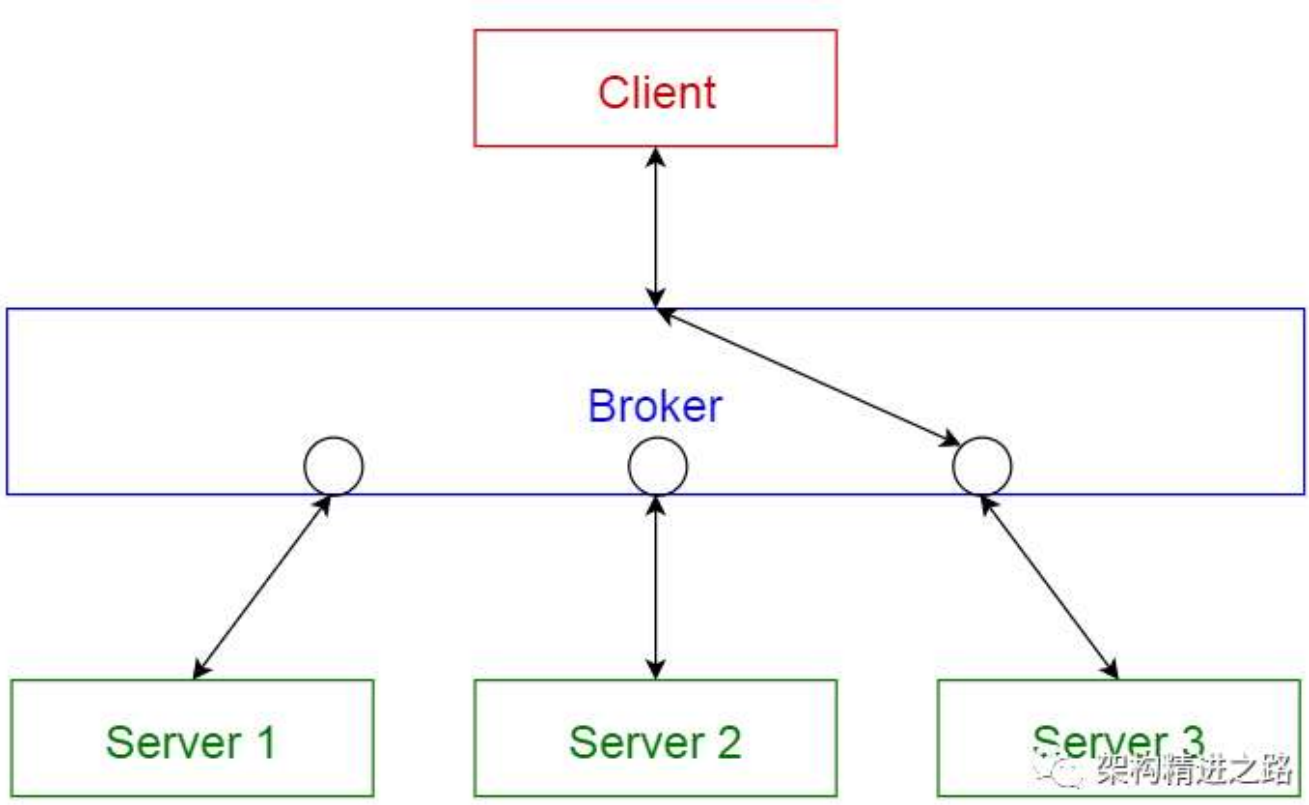
五. 代理模式

此模式用于构造具有解耦组件的分布式系统。这些组件可以通过远程服务调用彼此交互。代理组件负责组件之间的通信协调。

服务器将其功能(服务和特征)发布给代理。客户端从代理请求服务，然后代理将客户端重定向到其注册中心的适当服务。

使用场景：

- 消息代理软件，如Apache ActiveMQ，Apache Kafka，RabbitMQ和JBoss Messaging

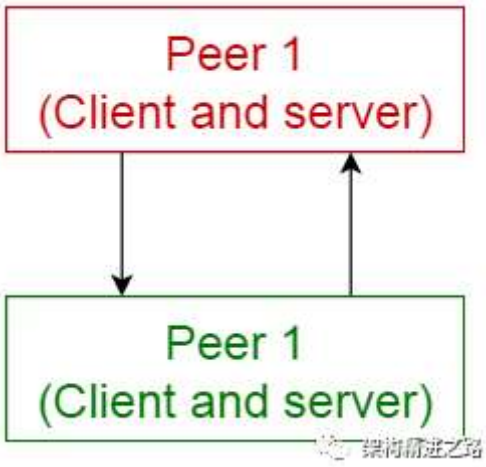


六. 点对点模式

在这种模式中，单个组件被称为对等点。对等点可以作为客户端，从其他对等点请求服务，作为服务器，为其他对等点提供服务。对等点可以充当客户端或服务器或两者的角色，并且可以随时间动态地更改其角色。

使用场景：

- 像Gnutella和G2这样的文件共享网络
- 多媒体协议，如P2PTV和PDTP
- 像Spotify这样的专有多媒体应用程序

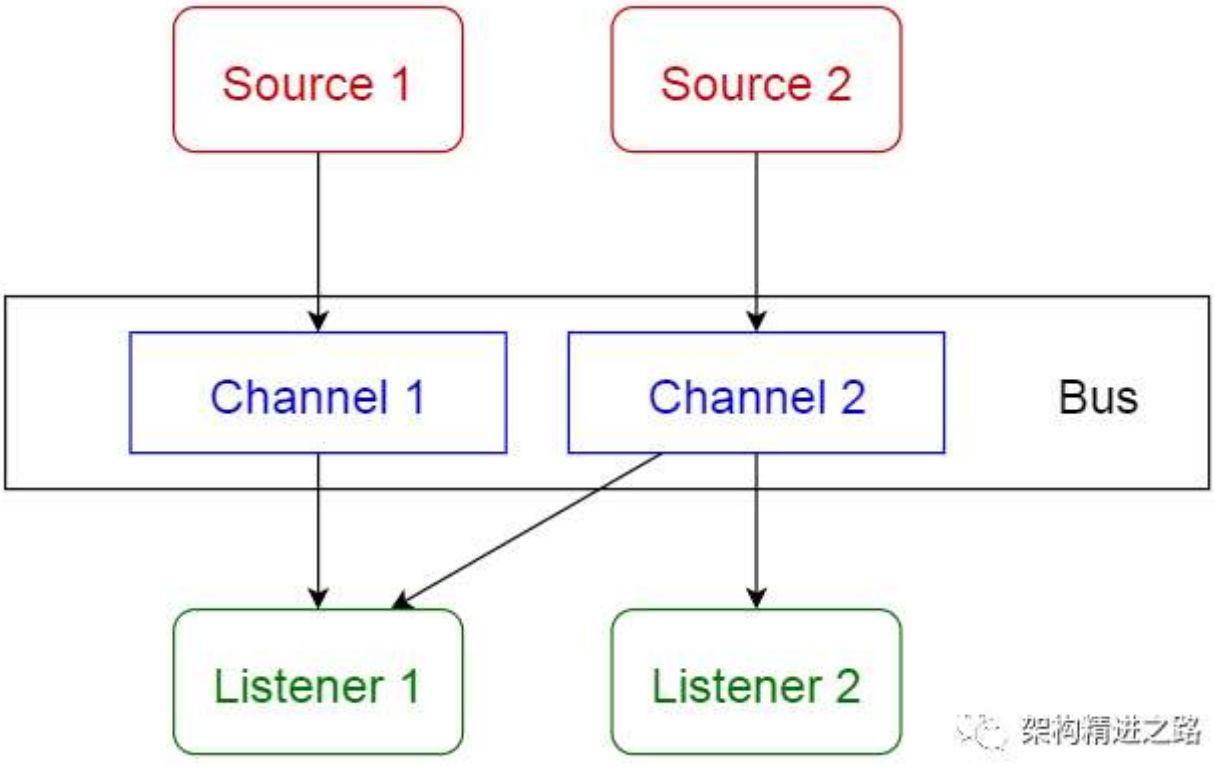


七. 事件总线模式

这种模式主要是处理事件，包括4个主要组件：事件源、事件监听器、通道和事件总线。消息源将消息发布到事件总线上的特定通道上。侦听器订阅特定的通道。侦听器会被通知消息，这些消息被发布到它们之前订阅的一个通道上。

使用场景：

- 安卓开发
- 通知服务



八. 模型-视图-控制器模式

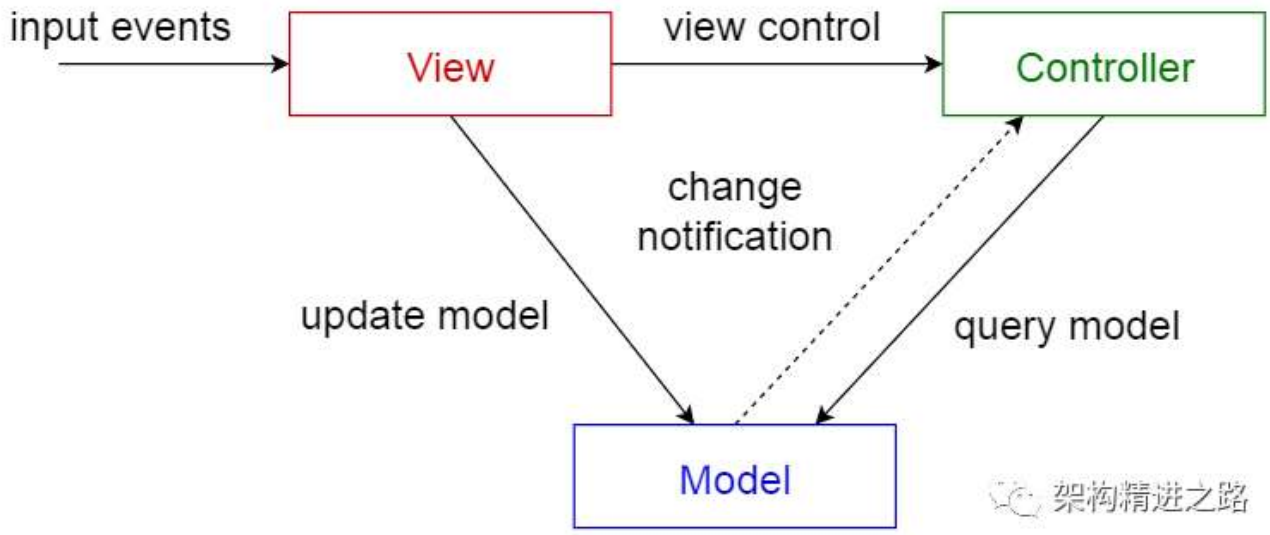
这种模式，也称为MVC模式，把一个交互式应用程序划分为3个部分，

- 模型：包含核心功能和数据
- 视图：将信息显示给用户(可以定义多个视图)
- 控制器：处理用户输入的信息

这样做是为了将信息的内部表示与信息的呈现方式分离开来，并接受用户的请求。它分离了组件，并允许有效的代码重用。

使用场景：

- 在主要编程语言中互联网应用程序的体系架构
- 像Django和Rails这样的Web框架



九. 黑板模式

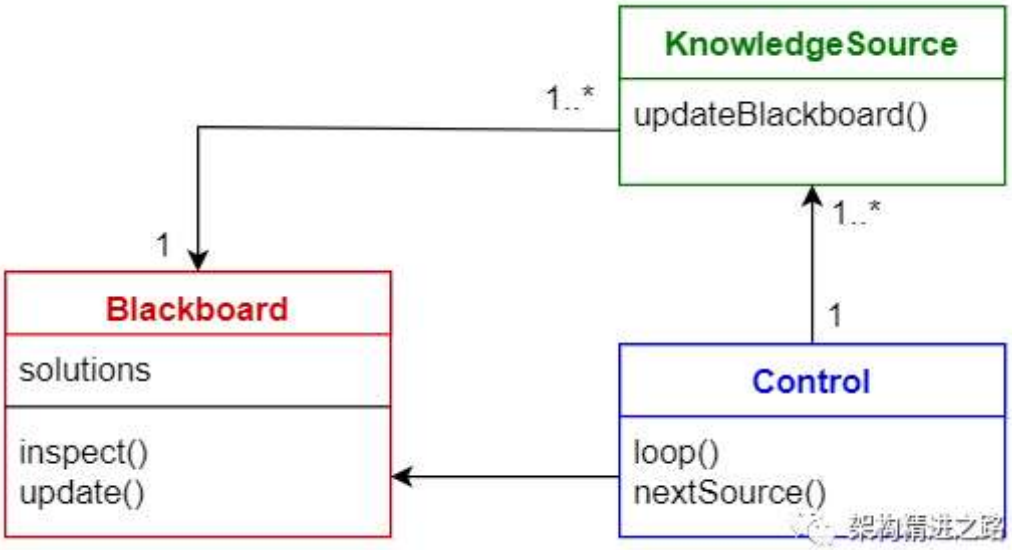
这种模式对于没有确定解决方案策略的问题是有用的。黑板模式由3个主要组成部分组成。

- 黑板——包含来自解决方案空间的对象的结构化全局内存
- 知识源——专门的模块和它们自己的表示
- 控制组件——选择、配置和执行模块

所有的组件都可以访问黑板。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，并通过与现有知识源的模式匹配来查找这些数据。

使用场景：

- 语音识别
- 车辆识别和跟踪
- 蛋白质结构识别
- 声纳信号的解释

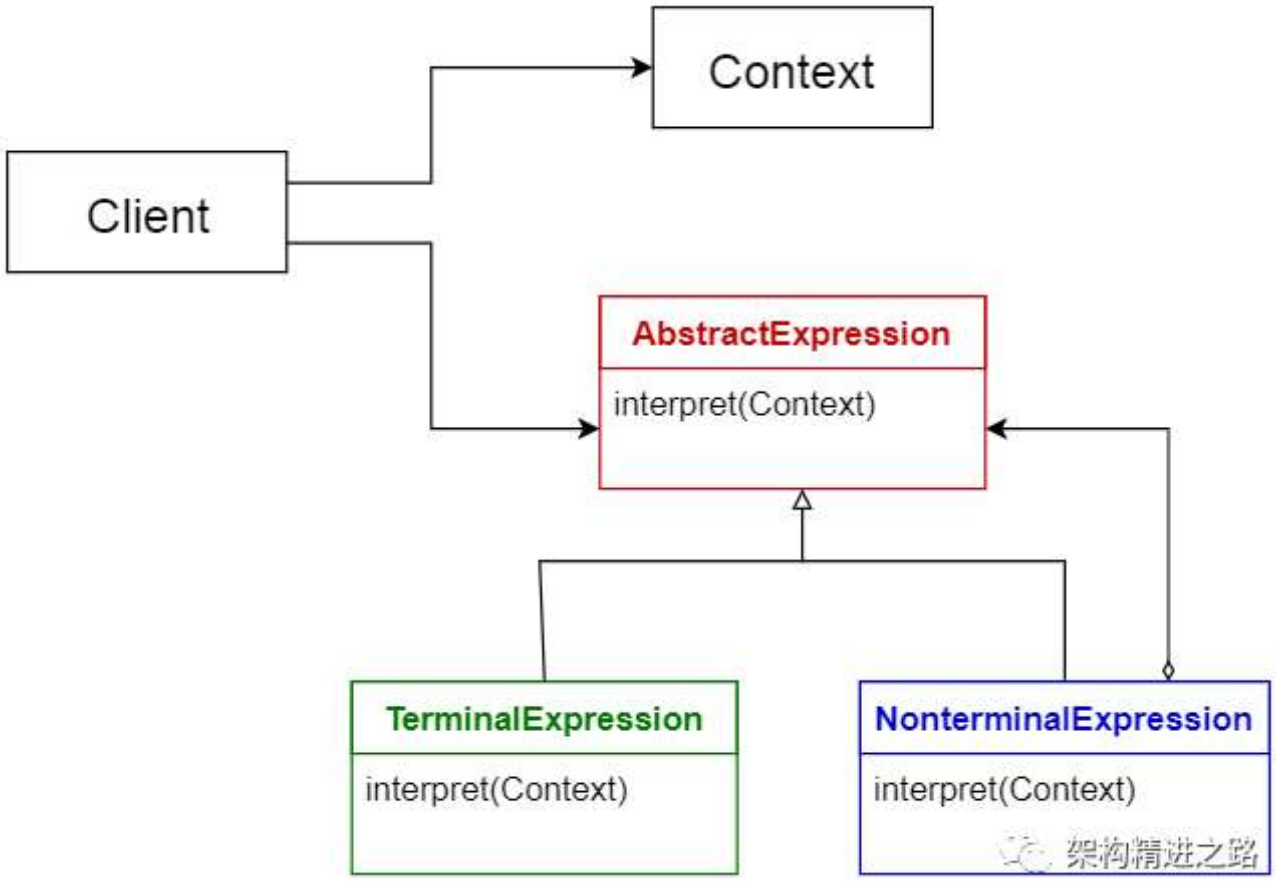


十. 解释器模式

这个模式用于设计一个解释用专用语言编写的程序的组件。它主要指定如何评估程序的行数，即以特定的语言编写的句子或表达式。其基本思想是为每种语言的符号都有一个分类。

使用场景：

- 数据库查询语言，比如SQL
- 用于描述通信协议的语言



体系架构模式的比较

下面给出的表格总结了每种体系架构模式的优缺点。

名称	优点	缺点
分层模式	一个较低的层可以被不同的层所使用。层使标准化更容易，因为我们可以清楚地定义级别。可以在层内进行更改，而不会影响其他层。	不是普遍适用的。在某些情况下，某些层可能会被跳过。
客户端-服务器模式	很好地建立一组服务，用户可以请求他们的服务。	请求通常在服务器上的单独线程中处理。由于不同的客户端具有不同的表示，进程间通信会导致额外开销。

名称	优点	缺点
主从设备模式	准确性——将服务的执行委托给不同的从设备，具有不同的实现。	从设备是孤立的：没有共享的状态。主-从通信中的延迟可能是一个问题，例如在实时系统中。这种模式只能应用于可以分解的问题。
管道-过滤器模式	展示并发处理。当输入和输出由流组成时，过滤器在接收数据时开始计算。轻松添加过滤器，系统可以轻松扩展。过滤器可重复使用。可以通过重新组合一组给定的过滤器来构建不同的管道。	效率受到最慢的过滤过程的限制。从一个过滤器移动到另一个过滤器时的数据转换开销。
代理模式	允许动态更改、添加、删除和重新定位对象，这使开发人员的发布变得透明。	要求对服务描述进行标准化。
点对点模式	支持分散式计算。对任何给定节点的故障处理具有强大的健壮性。在资源和计算能力方面具有很高的可扩展性。	服务质量没有保证，因为节点是自愿合作的。安全是很难得到保证的。性能取决于节点的数量。
事件总线模式	新的发布者、订阅者和连接可以很容易地添加。对高度分布式的应用程序有效。	可伸缩性可能是一个问题，因为所有消息都是通过同一事件总线进行的。
模型-视图-控制器模式	可以轻松地拥有同一个模型的多个视图，这些视图可以在运行时连接和断开。	增加复杂性。可能导致许多不必要的用户操作更新。
黑板模式	很容易添加新的应用程序。扩展数据空间的结构很简单。	修改数据空间的结构非常困难，因为所有应用程序都受到了影响。可能需要同步和访问控制。

名称	优点	缺点
解释器模式	高度动态的行为是可行的。对终端用户编程性提供好处。提高灵活性，因为替换一个解释程序很容易。	由于解释语言通常比编译后的语言慢，因此性能可能是一个问题。