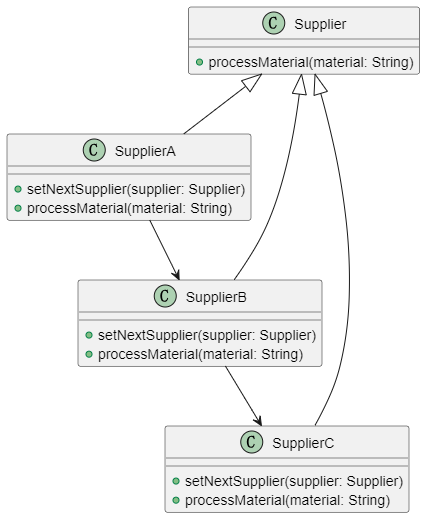
3.1 责任链模式

3.1.1 实现描述

在这个设计模式中，我们使用责任链模式来处理原材料的购买请求。每个具体供应商处理特定类型的原材料，如果不能处理，则将请求传递给下一个供应商。

3.1.2 类图等各种UML图，以及流程图

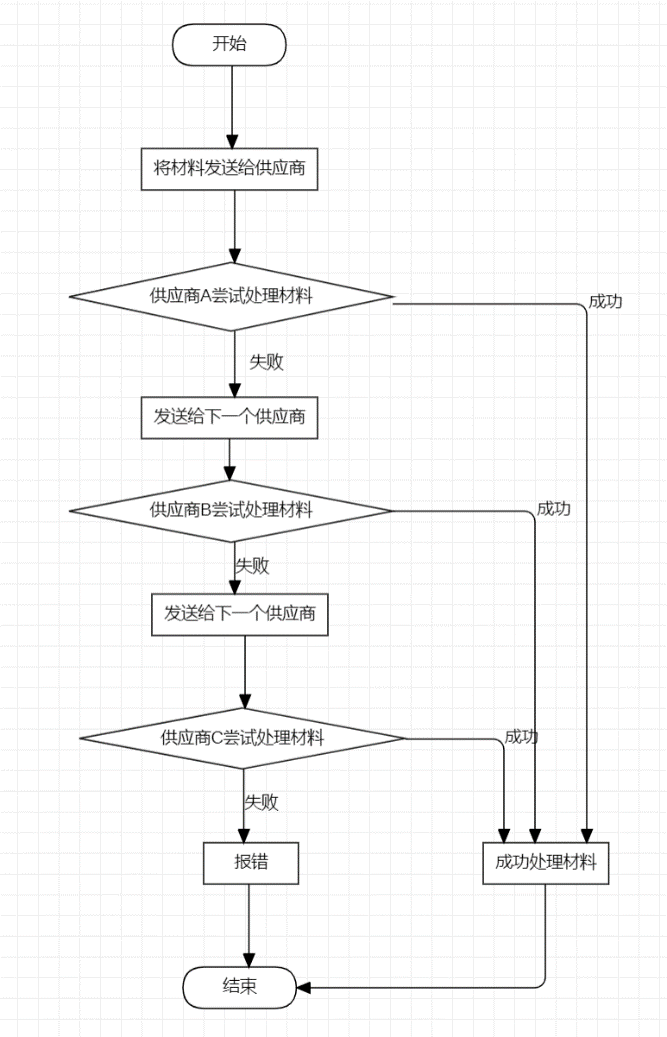
类图



这个类图描述了一个供应商（Supplier）的简单系统，其中包含了三个不同的供应商类：SupplierA、SupplierB 和 SupplierC。这些类共享一个共同的接口或方法 `processMaterial`，它接受一个字符串类型的参数 `material`。这个方法的目的可能是处理或加工某种材料。

每个供应商类都有一个方法 `setNextSupplier`，它接受一个 `Supplier` 类型的参数。这个方法可能用于设置当前供应商的下一个供应商，以便在当前供应商无法处理材料时，将任务传递给下一个供应商。

流程图



这个流程图描述了一个涉及材料处理的供应链流程。流程始于开始，随后进入步骤“将材料发送给供应商”。此后，流程分为两个主要分支：

如果供应商A成功处理材料，流程将直接跳到“成功处理材料”并结束。

如果供应商A处理材料失败，流程会继续，将材料发送给下一个供应商B。

对于供应商B，流程同样分为两个分支：

如果供应商B成功处理材料，流程将跳到“成功处理材料”并结束。

如果供应商B处理材料失败，流程会继续，将材料发送给下一个供应商C。

对于供应商C，流程再次分为两个分支：

如果供应商C成功处理材料，流程会跳到“成功处理材料”并结束。

如果供应商C处理材料失败，流程将报错并结束。

整个流程的目标是确保材料被成功处理。如果连续三个供应商都无法成功处理材料，流程将以报错结束，确保流程的可靠性和成功处理的达成。

3.1.3 代价分析

责任链模式在某些情况下可能会带来一些代价或负面影响。

3.1.3.1 责任链模式可能会导致性能损失，因为请求必须沿着链传递，直到找到合适的处理者。如果责任链很长，或者每个处理者都需要执行复杂的操作，可能会影响系统的响应时间。

3.1.3.2 责任链模式可能会使代码变得复杂和难以理解。由于责任链中的每个处理者都需要知道下一个处理者是谁，可能导致耦合度增加，代码变得难以维护和理解。

3.1.3.3 责任链模式可能导致请求被多个处理者处理，这可能会引入潜在的安全问题。如果没有正确配置责任链，可能会导致一些处理者未经授权地处理某些请求。

3.1.3.4 责任链模式可能不适用于所有的情况。如果系统的处理流程是固定的，责任链模式可能会显得过于复杂，不如直接使用其他简单的设计模式或方法。