1.4　设计模式的编目

抽象工厂模式（Abstract Factory Pattern / 抽象工厂 / 工厂族模式）：

> 提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们的具体类型；在嵌入式C中可用于**多种通信接口族的封装**，例如 `comm_factory_create()` 根据配置创建出一组匹配的驱动：UART、SPI、I2C 的收发与初始化函数集合，使上层通信模块在切换总线类型时不需修改业务逻辑，只替换工厂即可。

适配器模式（Adapter Pattern / 适配器 / 转接器模式）： 适配薄层 -移植常用

> 将一个类的接口转换成客户端期望的另一种接口，使原本不兼容的模块可以协同工作；在嵌入式 C 中常用于**统一不同外设驱动的接口格式**，例如将一个只能用 `HAL_UART_Transmit()` 的旧驱动包装成通用的 `comm_send()` 接口，使其能被系统的通信框架直接调用。

桥接模式（Bridge Pattern / 桥接模式 / 桥梁模式）： LVGL的打点函数

> 将抽象部分与其实现部分分离，使它们可以独立变化；在嵌入式 C 中常用于**分离设备逻辑与硬件驱动实现**，例如一个“显示控制模块”可独立于具体的显示接口（SPI、RGB、HDMI）而存在，只需在运行时“桥接”不同驱动实现。

建造者模式（Builder Pattern / 生成器模式 / 构造者模式）： TCP通信 -分步骤 或者 事件状态机

> 将一个复杂对象的构建过程与它的表示分离，使同样的构建步骤可以创建不同的对象；在嵌入式 C 中常用于**分阶段初始化复杂外设或配置结构体**，例如通过分步设置参数后统一生成一个完整的通信配置或任务对象。
>
> 初始化网卡硬件（MAC）
>
> 初始化物理层（PHY）
>
> 配置 IP 地址、子网、网关

> 初始化 TCP/IP 协议栈
>
> 注册回调、启动任务

责任链模式（Chain of Responsibility Pattern / 责任链 / 职责链模式）： 串行分层执行

> 将请求沿着处理者链传递，直到有对象处理它，从而避免请求发送者与接收者耦合；在嵌入式 C 中常用于**事件或消息处理流程的分发**，例如一个按键事件或通信消息依次通过不同处理模块（滤波器、解码器、业务处理）处理，直到某个模块处理完毕。

命令模式（Command Pattern / 命令模式 / 指令模式）： 消息处理队列- 指令对象化

> 将一个请求封装成对象，从而让你可用不同的请求、队列或日志参数化请求，以及支持撤销操作；在嵌入式 C 中常用于**统一控制硬件操作或任务指令**，例如按键操作、外设控制、定时任务调度都可以封装成命令对象，由上层统一调用或排队执行。

组合模式（Composite Pattern / 组合模式 / 部件-整体模式）： 文件系统

> 将对象组合成树形结构以表示"部分-整体"的层次结构，使客户端可以统一对待单个对象和组合对象；在嵌入式 C 中常用于**菜单、UI控件、文件系统或任务集合管理**，例如 LVGL 的容器控件可以包含按钮、标签、滑块，操作统一通过父容器处理。

装饰者模式（Decorator Pattern / 装饰模式 / 装饰器模式）： ulog打印

> 动态地给对象添加额外功能，而不改变其原有结构或接口；在嵌入式 C 中常用于**给驱动或任务增加功能**，例如在串口发送函数外层增加日志记录、校验、加密等功能，而不修改原始驱动代码。

外观模式（Facade Pattern / 外观模式 / 门面模式）： 和linux驱动设计原则有冲突； 适合多人协作，实现功能逻辑块

> 为子系统提供一个统一的高层接口，使子系统更易使用，降低依赖复杂性；在嵌入式 C 中常用于**封装多个外设或驱动操作成统一接口**，例如对一个完整通信模块（UART + DMA + CRC + 事件回调）提供一个简单的 `comm_init()`、`comm_send()` 接口，让上层业务不关心内部复杂细节。

工厂方法模式（Factory Method Pattern / 工厂方法 / 虚拟构造器模式）： 这个模式和抽象工厂模式区别？那个更灵活

> 定义一个创建对象的接口，让子类决定实例化哪一个类，从而使一个类的实例化延迟到子类；在嵌入式 C 中常用于**根据具体硬件或外设类型生成相应驱动实例**，例如不同型号的传感器或通信接口由不同的工厂函数创建，但上层调用统一接口获取对象。

Abstract Factory：上层只管调用统一接口，内部逻辑决定具体硬件；Factory Method：上层逻辑决定使用哪一个具体硬件对象。

享元模式（Flyweight Pattern / 享元模式 / 轻量级模式）： 通信的接收缓冲区； 指令队列

> 通过共享尽量多的相同对象来减少内存使用，把对象的状态分为**可共享内部状态**和**不可共享外部状态**；在嵌入式 C 中常用于**大量相似对象的管理**，例如 GUI 中重复的控件样式、字体、图标，或者通信协议中大量重复配置的数据结构。

解释器模式（Interpreter Pattern / 解释器模式）： 自己写的字符对比 二进制对比 ；大一点 python解释器 lua解释器

> 给定一种语言，定义它的文法表示，并定义一个解释器来解释句子；在嵌入式 C 中常用于**协议解析、命令解析或脚本解释**，例如解析简单通信协议、AT 命令或自定义脚本指令。

迭代器模式（Iterator Pattern / 迭代器模式 / 游标模式）： 一组IO中 ，哪些为高？ 一堆数据，几种状态 ，返回其中某种状态的 数据序号

> 提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露该对象的内部表示；在嵌入式 C 中常用于**遍历数组、链表或任务队列**，例如遍历消息队列、传感器数据列表或 GUI 控件集合。

FlashDB KV数据库

中介者模式（Mediator Pattern / 中介者模式 / 调停者模式）： 消息总线 ；共享内存 openAMP

> 用一个中介对象来封装对象之间的交互，使各对象不需要直接引用彼此，从而降低耦合；在嵌入式 C 中常用于**模块间消息或事件通信**，例如不同驱动或任务通过中介者发送消息，而不直接调用对方接口。

备忘录模式（Memento Pattern / 备忘录模式 / 快照模式）： ？？？

> 在不暴露对象实现细节的情况下，捕获并保存对象的内部状态，以便将来恢复；在嵌入式 C 中常用于**配置快照、状态保存或回滚机制**，例如保存外设寄存器配置、任务状态或通信会话状态，方便恢复。

RTOS 的线程上下文切换就是 Memento 模式：保存任务的完整 CPU 状态快照，当切换回来时恢复，保证任务无缝执行”。

backtrace

观察者模式（Observer Pattern / 观察者模式 / 发布-订阅模式）： 一对多，单触发多执行； 消息订阅

> 定义对象间的一种一对多依赖，当一个对象状态发生变化时，所有依赖它的对象都会得到通知并自动更新；在嵌入式 C 中常用于**事件驱动、消息通知或状态同步**，例如传感器状态变化通知多个任务、驱动事件广播、GUI 控件更新。

原型模式（Prototype Pattern / 原型模式 / 克隆模式）： 创建相似任务或配置对象 ；UI里面常见

> 通过复制现有对象来创建新对象，而不是通过构造函数创建；在嵌入式 C 中常用于**快速生成相似配置对象或任务实例**，例如重复初始化配置、创建相似通信帧或 GUI 控件实例。

代理模式（Proxy Pattern / 代理模式 / 代理类模式）：非直接读写寄存器 通过结构体对象 函数调用间接访问

> 为另一个对象提供一个**代理或占位符**以控制对它的访问；在嵌入式 C 中常用于**外设访问控制、延迟初始化或安全访问**，例如对外设寄存器操作做统一接口、懒加载硬件资源或通过中间层控制访问。

单例模式（Singleton Pattern / 单例模式 / 全局唯一模式）： 全局变量 全局对象 ？？？ 多线程 得加锁

> 保证一个类只有一个实例，并提供全局访问点；在嵌入式 C 中常用于**全局硬件资源管理、系统配置或全局状态对象**，例如系统日志模块、全局通信接口或外设管理器。

状态模式（State Pattern / 状态模式 / 状态机模式）：按钮的 点击 双击 长按 这种状态切换

> 允许对象在内部状态改变时改变它的行为，看起来像修改了对象的类；在嵌入式 C 中常用于**事件驱动的状态机、任务或外设状态管理**，例如设备工作模式切换、通信协议状态机、按键处理。

策略模式（Strategy Pattern / 策略模式 / 算法封装模式）： 定义统一的数据输入输出接口

> 定义一系列算法，将每个算法封装起来，并使它们可以互换；在嵌入式 C 中常用于**不同功能或算法的可替换实现**，例如不同滤波器、数据压缩算法或通信协议处理方式。

模板方法模式（Template Method Pattern / 模板方法模式 / 固定流程模式）：流程可复用，步骤可定制

> 在父类中定义一个算法的骨架，而将某些步骤延迟到子类实现；在嵌入式 C 中常用于**固定流程但可定制步骤的任务或外设操作**，例如初始化外设、通信序列或任务执行流程。

协议转换模块中，固定收→解析→发的流程属于 Template Method，不同协议的解析和转发逻辑是可替换步骤

访问者模式（Visitor Pattern / 访问者模式 / 元素操作分离模式）： 对不同的对象 附加统一的新操作 ；RTT 外设使用计数

> 将作用于某对象结构的操作封装成独立的访问者，使得可以在不改变对象结构的前提下增加新操作；在嵌入式 C 中常用于**不同数据类型或协议帧的统一处理**，例如对多种外设数据包或消息结构做统一解析、统计或日志处理。

1.5 组织编目

1. 创建型模 式与对象的创建有关；结构型模式处理类或对象的组合；行为型模式对类或对象怎样交互和怎样分配职责 进行描述。
2. 类模式处理类和子类之间的关系，这些关 系通过继承建立，是静态的，在编译时便确定下来了。对象模式处理对象间的关系，这些关系在运行时是 可以变化的，更具动态性。

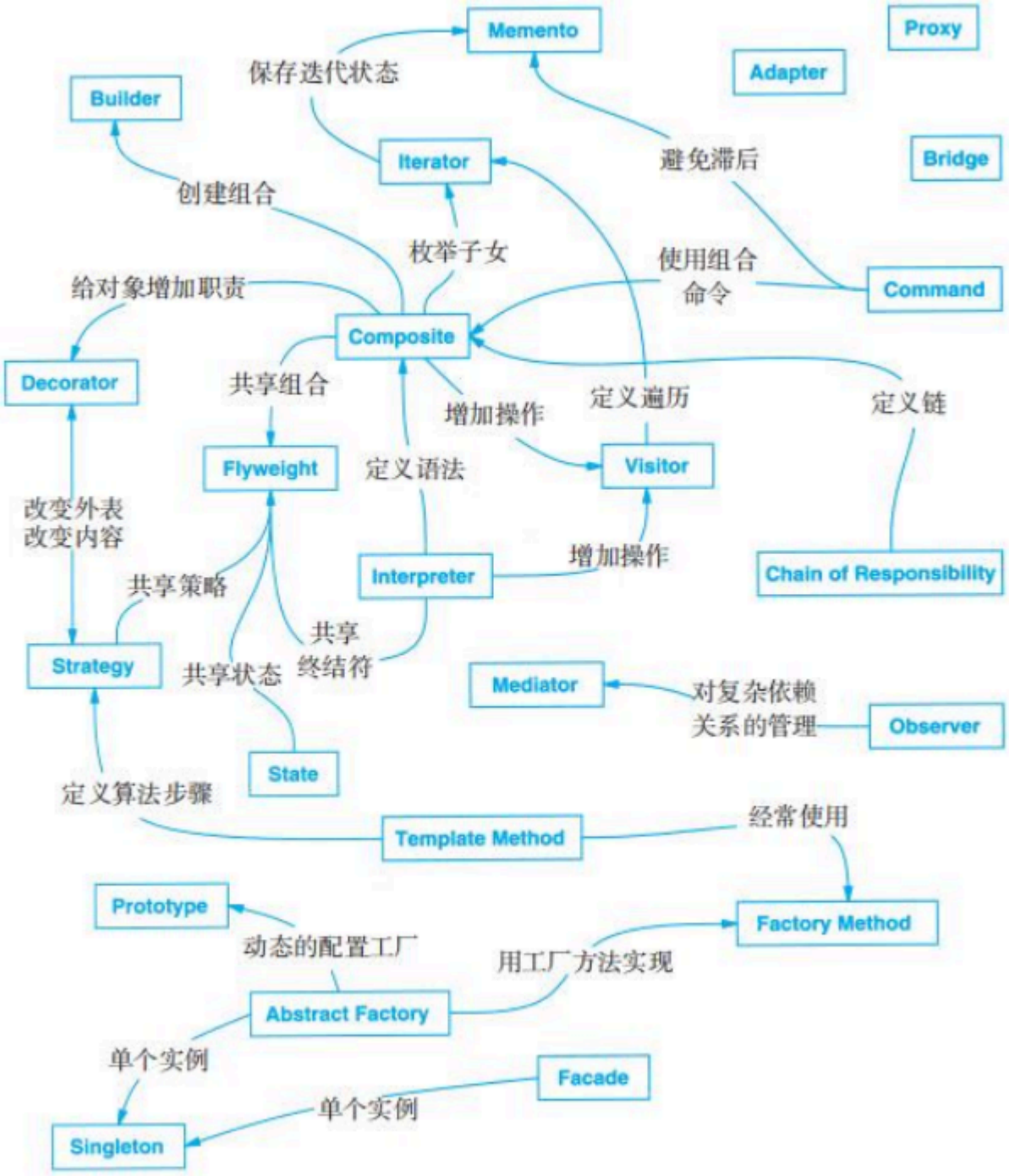
		目　的		
		创　建　型	结　构　型	行　为　型
范围	类	Factory Method(3.3)	Adapter(类)(4.1)	Interpreter(5.3) Template Method(5.10)
	对象	Abstract Factory(3.1) Builder(3.2) Prototype(3.4) Singleton(3.5)	Adapter(对象)(4.1) Bridge(4.2) Composite(4.3) Decorator(4.4) Facade(4.5) Flyweight(4.6) Proxy(4.7)	Chain of Responsibility(5.1) Command(5.2) Iterator(5.4) Mediator(5.5) Memento(5.6) Observer(5.7) State(5.8) Strategy(5.9) Visitor(5.10)

创建型类模式将对象的部分创建工作延迟到子类，而创建型对象模式则将它延迟到另一个对象中。

结构型类模式使用继承机制来组合类，而结构型对象模式则描述了对象的组装方式。

行为型类模式使用继承 描述算法和控制流，而行为型对象模式则描述了一组对象怎样协作完成单个对象所无法完成的任务。

根据模式的“相关模式”部分所描述的它们怎样互相引用来组织设计模式。


Memento
Proxy
保存迭代状态
Adapter
Builder
Iterator
避免滞后
Bridge
创建组合
枚举子女
使用组合
命令
给对象增加职责
Command
Composite
Decorator
共享组合
定义遍历
定义链
增加操作
Flyweight
定义语法
Visitor
改变外表
改变内容
Interpreter
增加操作
Chain of Responsibility
共享策略
共享
终结符
Strategy
共享状态
Mediator
对复杂依赖
关系的管理
Observer
State
定义算法步骤
Template Method
经常使用
Prototype
Factory Method
动态的配置工厂
用工厂方法实现
Abstract Factory
单个实例
Facade
单个实例
Singleton