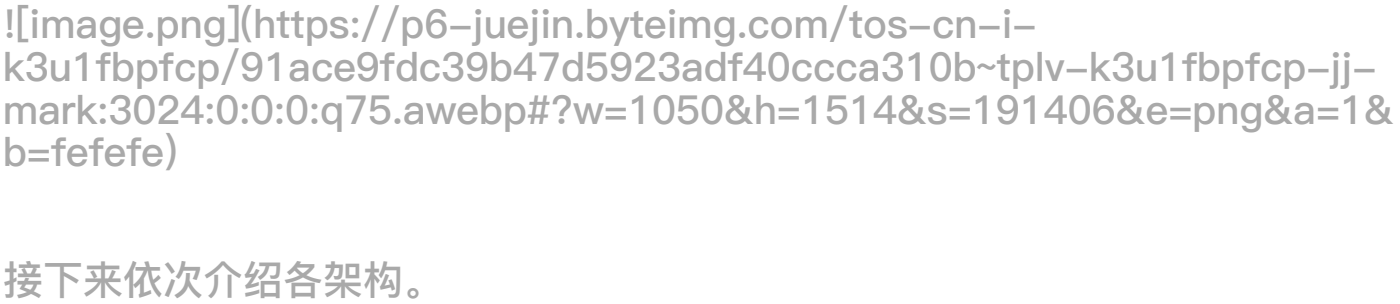
# DDD中常提到的应用架构总结（六边形、洋葱、整洁、清晰）

### 一、引言

最近在学习DDD应用**架构设计**时，接触到了不同的应用架构设计概念，如**六边形架构、洋葱架构、整洁架构、清晰架构...**，起初是一头雾水，在过程中也算对此有了些理解，故在本文中对这些架构进行了简单的介绍和总结。

这些架构随时间的演进可参见下图：



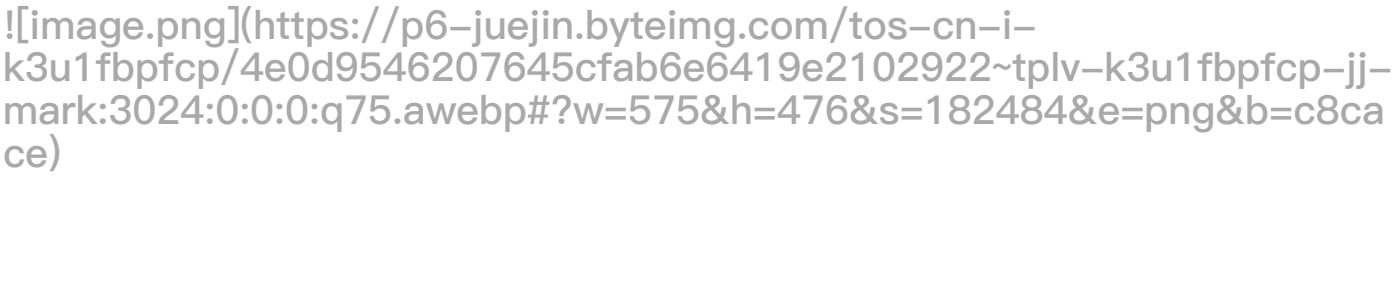

接下来依次介绍各架构。

### 二、EIC(Entity-Interface-Control) & EBI（Entity-Boundary-Interactor）

1992年由Ivar Jacobson提出**EIC(Entity-Interface-Control)** 架构，

之后Robert C. Martin在一次关于整洁架构**Clean Architecture**的演讲中将其调整为**EBI（Entity-Boundary-Interactor）**，即：

* **Entity => Entity**: 维持不变
* **Interface => Boundary**: 避免和编程语言中的interface相混淆
* **Control => Interactor**: 避免和MVC Controller相混淆



> **注：** 参考上图EBI架构，图片中的矩形边数或边界
!![image.png](https://p9-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/ca2b4f1267fb426db6cbcc45f2f2ec6d~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=41&h=44&s=482&e=png&b=ffffff)数量为4，
>
> 若将其修改为6边型，即可演进为2005年（13年后）提出的六边形架构（见下文），
>
> 即可以将其视为六边形架构（端口、适配器架构）思想的前身。

**EBI**对象类型说明如下：

* **Entity**: 实体对象，包含数据、行为（避免贫血模型anaemic entities）
* **Boundary(Interface)**: 系统与外界交互的边界，即对应bounday object，如controller实现（主动）、持久化实现（被动）、消息通知（被动）等。
* **Interactor(Control)**: 所有不适合放在Entity和Boundary的行为都需要放在Interactor对象中，等价于DDD中的：
+ **Application Services**（who orchestrate(编排) use cases) – DDD架构中的应用服务层
+ **Domain Services** (who contain Domain behaviour but are not entities) – DDD架构中的领域服务层

类比MVC(Model View Controller)模式，

* 模型层Model即表示整个后端back-end，包括services、entities等
* 表现层Presentation即包含View层、Controller层
* View层、Controller层即为Boundary，
* Entity即为实体数据及其关联的行为操作（充血模型）
* Interactor（Application Services, Domain Services）即为表现层（View、Controller）和Entity的连接，
* 将MVC和EBI模式组合起来类似：View-Controller-Interactor-Entity

### 三、端口和适配器架构Ports & Adapters Architecture（又称六边形架构Hexagonal Architecture）

2005年，Alistair Cockburn提出了端口和适配器架构Ports & Adapters Architecture(又称六边形架构Hexagonal Architecture)。

![分层架构（上下） -> 左右 -> 多边 -> 主被动适配器](https://p3-

juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/a653c3eb6ab14729bb58911ed6ed50d5~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=2095&h=1412&s=331294&e=png&b=f9f3f1)

端口和适配器架构中的相关概念：

* **工具Tools：** 应用application使用的工具Tools(WebSerevr, Cli, DB, SearchEngine, MQ)
* **端口Port：** 在application core层定义的工具交互规范（即定义 工具Tools如何使用application core的规范 或 工具Tools如何被application core使用的规范），对应编程语言的interface定义，且端口定义应该符合应用层逻辑（不可简单模仿或者直接使用底层工具的API）
* **适配器Adapter：** 连接工具Tools(WebSerevr, Cli, DB, SearchEngine, MQ)和应用核心application core的代码单元

+ 适配器依赖port（调用port 或 实现port）和工具，但是application core仅依赖port。
* **主动适配器（Primary/Driving Adapters ）** – 主动调用应用核心Application Core，触发应用执行某项活动

+ **主动适配器** -> 依赖**端口** -> 注入**端口实现**,
+ 例如主动适配器Controller -> 依赖端口ServiceInterface -> 注入端口实现ServiceImpl
* **被动适配器（Secondary/Driven Adapters）** – 被应用核心Application Core调用来执行动作，实现应用定义的端口

+ **端口** -->**被动适配器**实现端口逻辑-> 包装系统外部工具,
+ 例如端口RespositoryInterface -> 被动适配器Mysql实现MysqlRepositoryImpl -> 调用Mysql数据库

几边形不重要，可以N多边，重点是Port & Adapter思想，关于端口和适配器架构的优势：

* 核心业务在最中心（最重要），
* 核心业务依赖Port，Adapter依赖Port（主动） 或 Adapter为Port的具体实现（被动），
* 核心业务逻辑与实现细节（技术框架、底层存储、工具、传输通信机制等）相隔离，
* 保持核心业务逻辑的可重用性，

* 可通过（注入）不同Adapter实现来切换技术实现，避免技术框架、工具、供应商锁定，
* 且基于mock Port的形式更易于测试。

### 四、洋葱架构Onion Architecture

洋葱架构（Onion Architecture） 在2008年由Jeffrey Palermo提出。




洋葱架构 构建在Port & Adapter架构（又称六边形架构）之上，在Port & Adapter架构中仅提到2层：

1）外层 – 表示传输（通信）机制和基础设施infrastructure

2）内层 – 业务逻辑

而**洋葱架构在DDD的基础上，将内层（业务逻辑层，Business Logic）进一步划分**，最终为：

* Adapters，即六边形架构中的适配器Adapter层
+ User Interface、Infrastructure、Tests
* Application Core，应用核心层，也即原来的六边形架构中心的Business Logic层
+ Application Services – 应用服务册层
+ Domain Services – 领域服务层
+ Domain Model – 领域模型层

且**明确了依赖方向**：

* 外层依赖内层
* 内层不知道外层
* 且在不破坏依赖方向的前提下，外层亦可以直接调用任一内层（不一定是相邻接的2层），参考CQRS中Query实现（Application Services直接调用DAO）

### 五、整洁架构Clean Architecture

2012年Robert C. Martin (又名Uncle Bob) 提出了**整洁架构Clean Architecture**，整洁架构将EBI、六边形架构Hexagonal、洋葱架构Onoin等整合成一个可以实际落地的架构。


与洋葱架构相比，整洁架构调整如下：

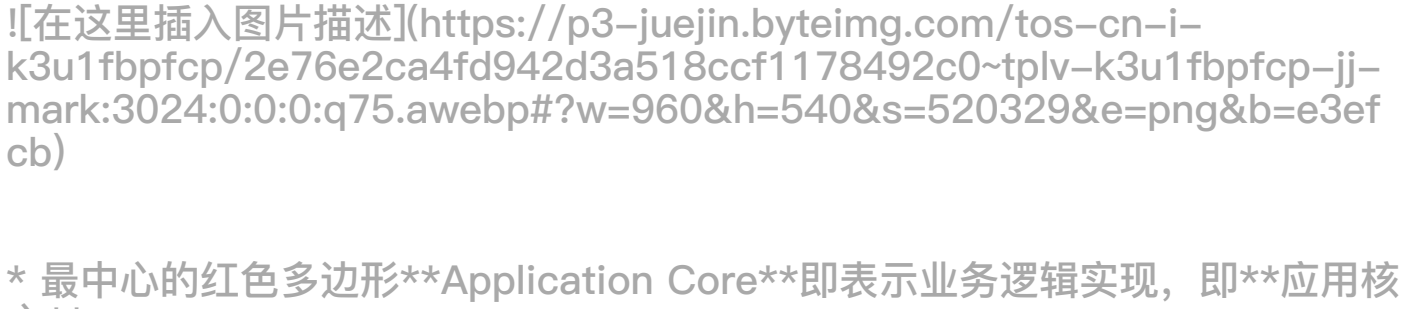
* Application Services调整为Use Cases
* Domain Services, Domain Model调整为Entities

整洁架构并没有带来突破性的概念或模式，但是：

* 它发掘了某种程度上被遗忘了的概念、规则和模式；
* 它澄清了一些实用且重要的概念、规则和模式；
* 它告诉我们如何把所有的概念、规则和模式整合起来，形成一种构建复杂应用并保持可维护性的标准套路

### 六、清晰架构Explicit Architecture

2017年Herberto Graca在其[软件架构编年史](http://cxyroad.com/”https://herbertograca.com/2017/07/03/the-software-architecture-chronicles/”)系列文章中提出**清晰架构Explicit Architecture**，即将DDD, Hexagonal, Onion, Clean, CQRS等进行融合后的架构。




* 最中心的红色多边形**Application Core**即表示业务逻辑实现，即**应用核心**
+ 红色多边形的边界即表示**端口Port**，即应用核心的入口/出口定义
+ **Application Layer** – 应用层，包括：
– **Application Services**，业务用例的编排服务即及其interface定义，应用服务的作用通常如下：
* 使用 Repostitory 查找一个或多个实体；

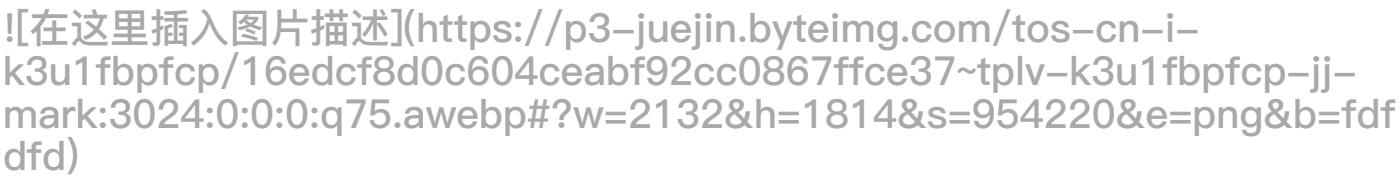
* 让这些实体执行一些领域逻辑；
* 再次使用 Repostitory 让这些实体持久化，有效地保存数据变化；
* 触发应用事件（如发送邮件、调用第三方API、发送MQ消息等）。
– **CQRS命令/查询处理器**
– **Event Listener事件监听器**
– **Port端口定义**，如ORM 接口Repostitory、搜索引擎接口、消息接口等等
+ **Domain Layer** – 领域层，这一层含了数据和操作数据的逻辑，它们只和领域本身有关，独立于调用这些逻辑的业务过程。它们完全独立，对应用层完全无感知。
– **Domain Services** – 领域服务，封装涉及多实体（相同或不同实体类型）的领域逻辑，且领域服务间可以相互调用。
– **Domain Models** – 领域模型，在架构的正中心，完全不依赖外部任何层次的领域模型。它包含了那些表示领域中某个概念的业务对象，如实体、值对象、枚举以及其它领域模型种用到的任何对象（如领域事件Domain Events，简单理解为MQ消息）。
* 红色多边形的外侧左半圆部分即为**主/主动适配器（用户界面User Interface实现）**
+ 如Spring MVC中的Controller实现
+ Command Query Bus 命令查询总线
* 红色多边形的外侧右半圆部分即**次/被动适配器（基础设置Infrastructure实现）**
+ 如数据持久化实现Mysql、短信通知实现、MQ通知、搜索引擎ES实现等
+ Event Bus 事件总线
* 依赖方向由外到内，且内层不知道外层（参见之前洋葱架构）

采用**按组件分包Package By Component**，即整合传统按层分包Package By Layer和按特性分包Package By Feature，**组件Component**可以理解为专属于特定一个领域内的业务逻辑服务和数据访问逻辑的组合，也可以理解为特定领域，如账单、用户、评论或帐号等。

> A “component” in this sense is a combination of the business and data access logic related to a specific thing (e.g. domain concept, bounded context, etc).

在清晰架构中可以理解为：

* 先按照层次进行分包
+ 表现层Presentation
+ 业务核心层Application Core
+ 基础设施层Infrastructure
* 之后每一层次再按照特性分包

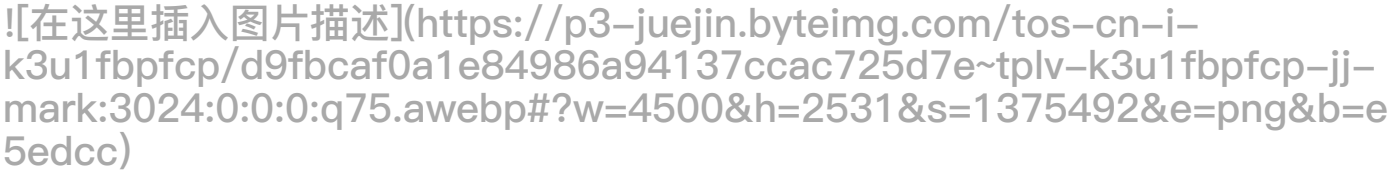



正如Herberto Graca在[DDD, Hexagonal, Onion, Clean, CQRS, … How I put it all together](http://cxyroad.com/”https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/”)中写到：

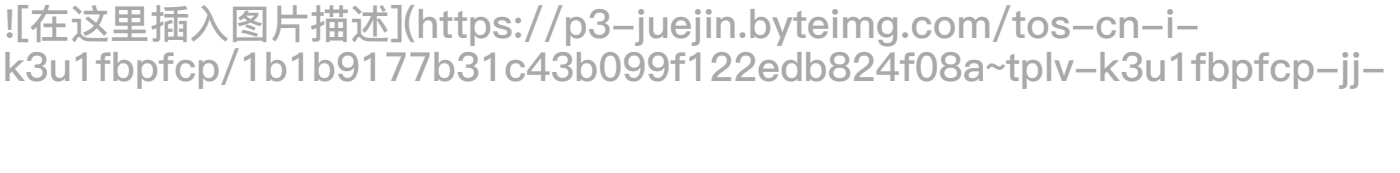
> 清晰架构只是一份指南！
>
> 应用才是疆域，现实情况和具体用例才是运用这些知识的地方，它们才能勾勒出实际架构的轮廓！
>
> 我们需要理解所有这些模式，
>
> 但我们还时常需要思考和理解我们的应用需要什么，我们应该在追求解耦和内聚的道路上走多远。
>
> 这个决定可能受到许多因素的影响，
>
> 包括项目的功能需求，也包括构建应用的时间期限，应用寿命，开发团队的经验等等因素。

---

附国内大神翻译的清晰架构中文版：




关于传统分层、六边形、洋葱、整洁、清晰架构的演进可参见下图：



mark:3024:0:0:0:q75.awebp#?w=2117&h=3637&s=1191309&e=png&b=f8f1ec)

### 七、汇总

综合以上架构，给出我认为合理的、可以落地的架构如下图（后续会根据实际的使用体验进行完善）：

> **注：**
>
> D3S（DDD with SpringBoot）为本作者使用DDD过程中开发的框架，
>
> 源码地址：[gitee.com/luoex/d3s](http://cxyroad.com/”https://gitee.com/luoex/d3s”)

![在这里插入图片描述](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/25e6ee047fc447b1a6101ec54c588b8d~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=2048&h=1511&s=420714&e=jpg&b=fdfdfd)

### 八、后续...

* 清晰架构中跨组件通信
* CQRS处理
* 共享内核等
* Event Driven
* 限界上线文、上下文映射

---

**参考：**

**EBI：**

[herbertograca.com/2017/08/24/...](http://cxyroad.com/”https://herbertograca.com/2017/08/24/ebi-architecture/”)

**端口和适配器架构：**

[11.端口和适配器架构(译) – https://www.jianshu.com/p/f39f4537857e](http://cxyroad.com/"https://www.jianshu.com/p/f39f4537857e")

[herbertograca.com/2017/09/14/...](http://cxyroad.com/"https://herbertograca.com/2017/09/14/ports-adapters-architecture/")

**洋葱架构：**

[herbertograca.com/2017/09/21/...](http://cxyroad.com/"https://herbertograca.com/2017/09/21/onion-architecture/")

[jeffreypalermo.com/2008/07/the...](http://cxyroad.com/"https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/")

**整洁架构：**

[herbertograca.com/2017/09/28/...](http://cxyroad.com/"https://herbertograca.com/2017/09/28/clean-architecture-standing-on-the-shoulders-of-giants/")

[blog.cleancoder.com/uncle-bob/2...](http://cxyroad.com/"https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html")

**清晰架构：**

[herbertograca.com/2017/11/16/...](http://cxyroad.com/"https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/")

[17.清晰架构(01): 融合 DDD、洋葱架构、整洁架构、CQRS...(译) – https://www.jianshu.com/p/d3e8b9ac097b](http://cxyroad.com/"https://www.jianshu.com/p/d3e8b9ac097b")

**DDD：**

[herbertograca.com/2017/09/07/...](http://cxyroad.com/"https://herbertograca.com/2017/09/07/domain-driven-design/")