宁波DOTNET社区线下交流分享活动
OCTOBER 2018

- 如何正确对待高性能诉求?
- 如何保证代码的可维护性?

1.1关于性能，常见的入手方向

编程语言

数据库

缓存

Async/await

开发框架

负载均衡
集群

还有其他入手的方向么?

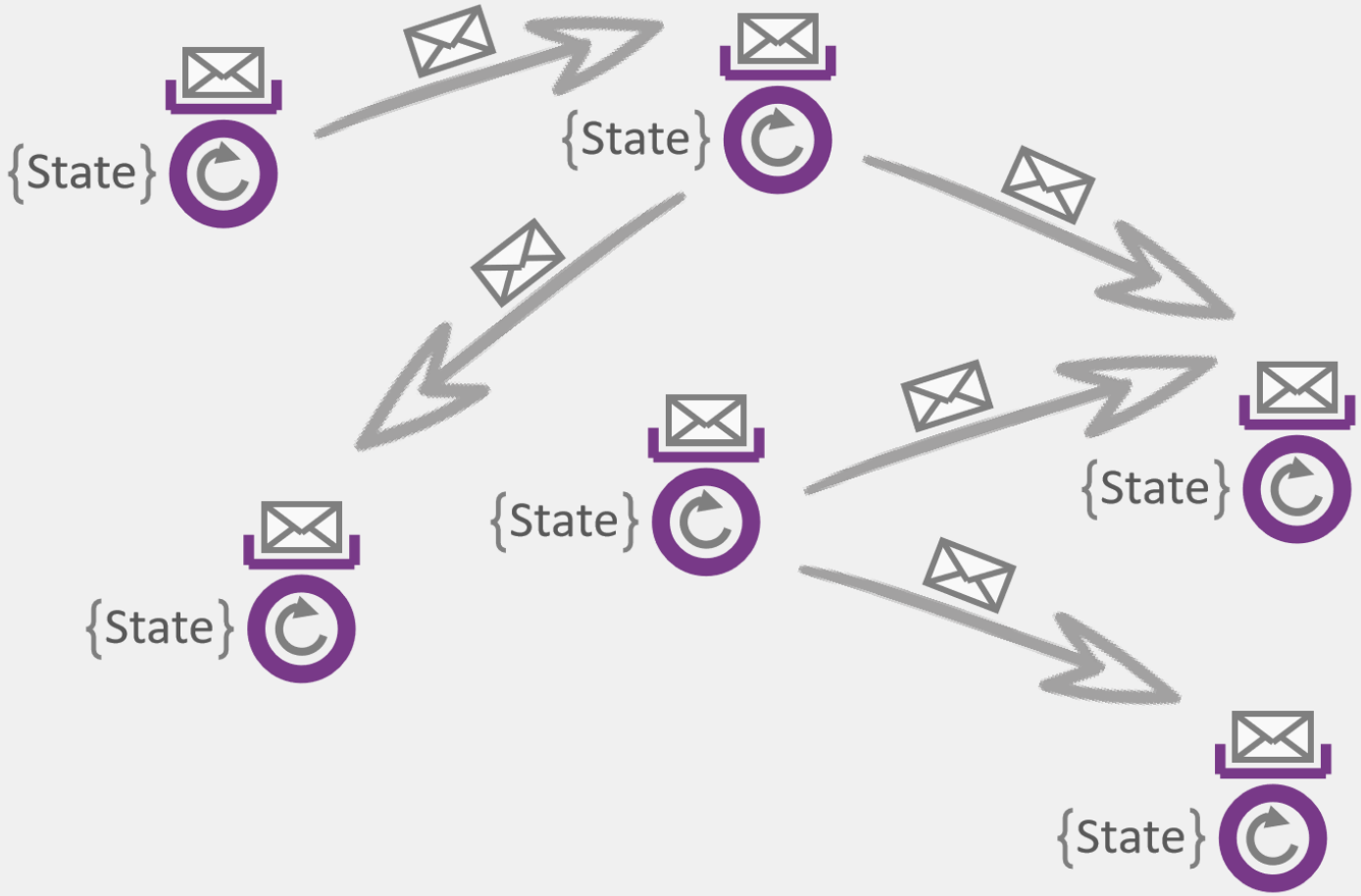
1.2思考一下无状态开发模式的问题?

开发框架

无状态		有状态	
优势	劣势	优势	劣势

1.3极端高并发需求的应对

Actor framework



Actors on receiving a message can

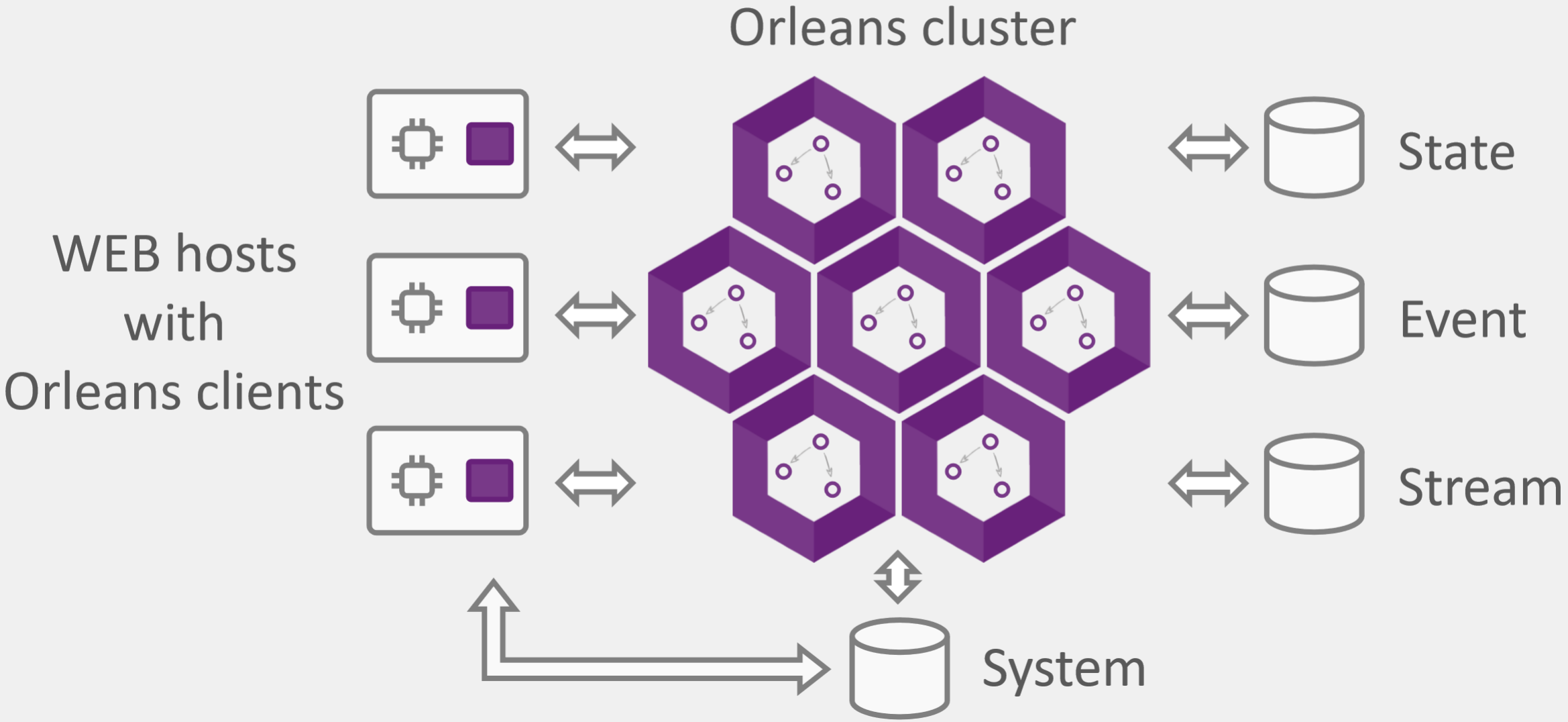
- spawn new actors
- send messages
- change state

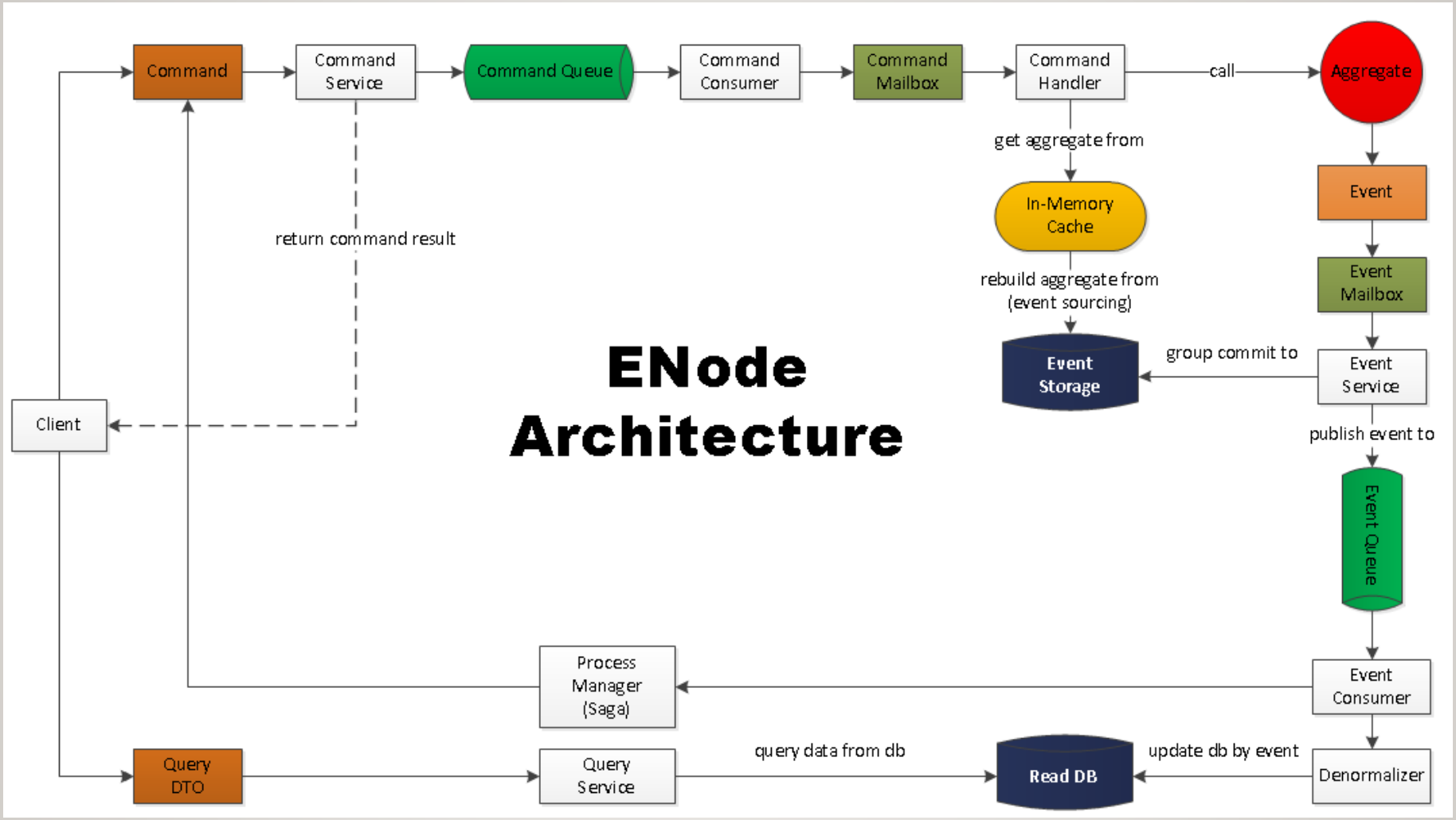
Note

- state is internal
- messages are processed one by one

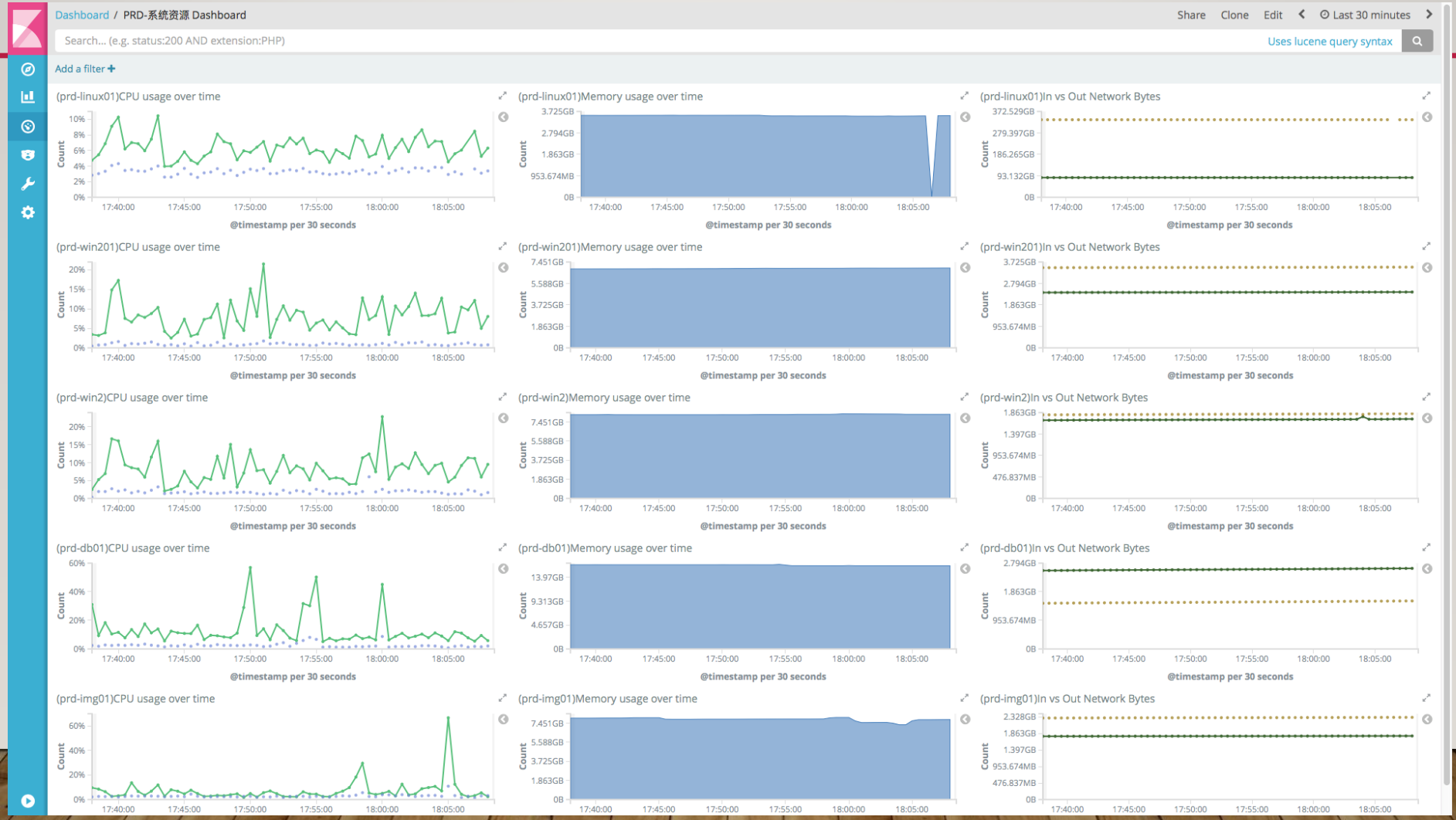
Actor ≈ Object on steroids

Stateful 3-tier architecture

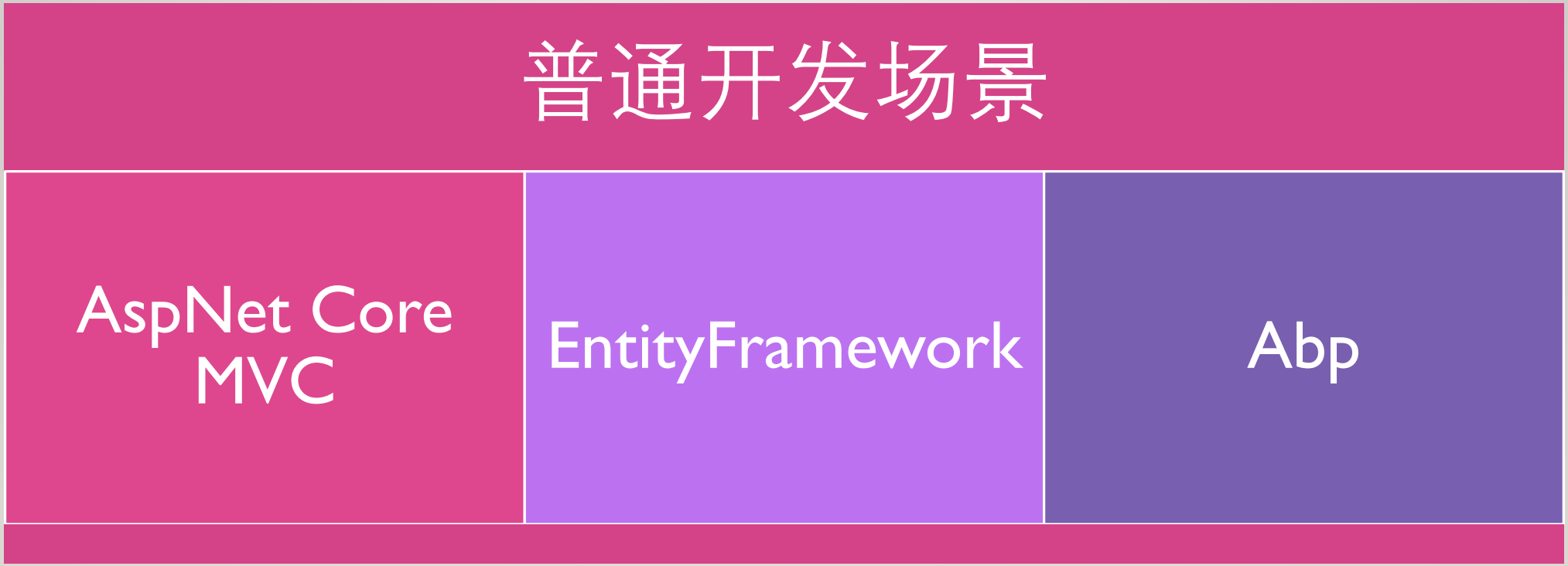



Command
Command Service
Command Queue
Command Consumer
Command Mailbox
Command Handler
call
Aggregate
get aggregate from
In-Memory Cache
Event
Event Mailbox
rebuild aggregate from (event sourcing)
Event Storage
group commit to
Event Service
return command result
ENode Architecture
Client
publish event to
Event Queue
Process Manager (Saga)
Event Consumer
Query DTO
Query Service
query data from db
Read DB
update db by event
Denormalizer

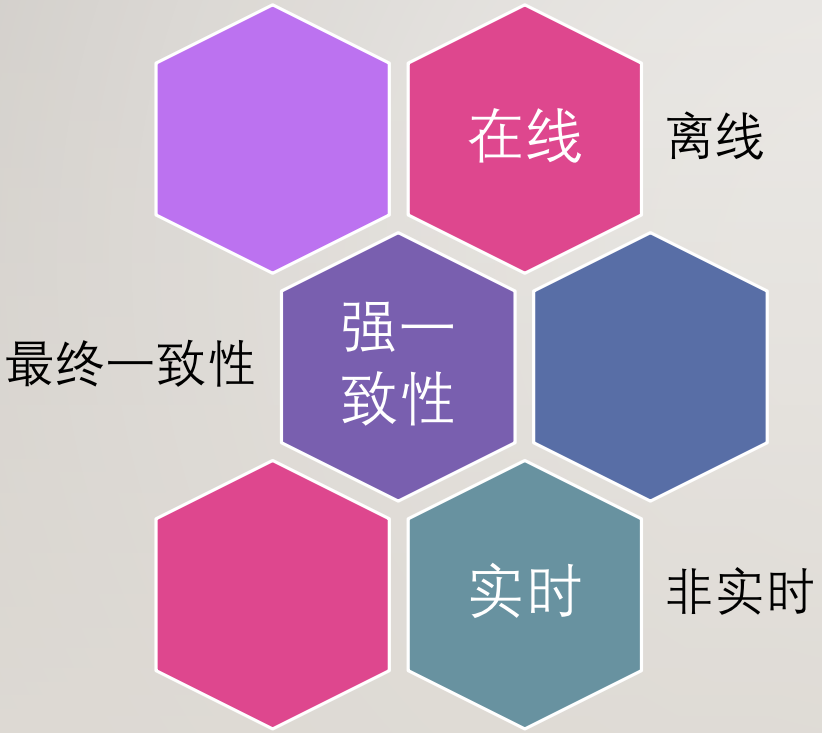
1.4服务器资源被浪费了



1.5普遍的无状态方式开发情况下，我们还可以做些什么？



1.5普遍的无状态方式开发情况下，我们还可以做些什么？



度量分析的手段

执行耗时

APM监控

2.1代码可维护性的本质是什么？

在可读性上，统一阅读体验。

项目结构

- 如何组织程序集

命名空间

- 逻辑纬度的组织（相对于程序集的物理纬度）

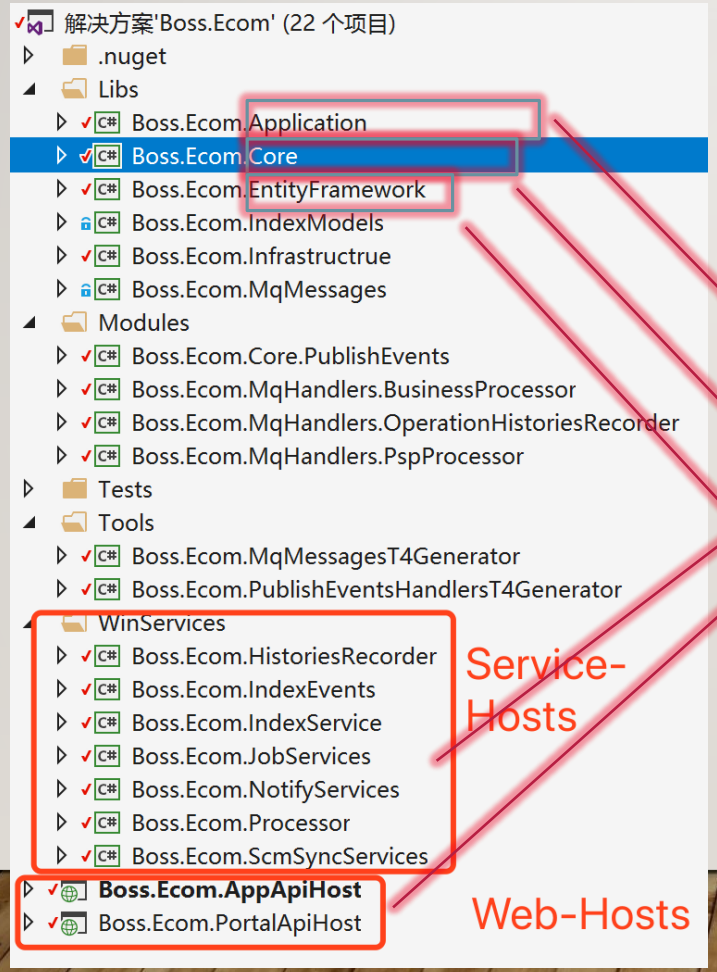
目录结构

- 如何组织代码文件

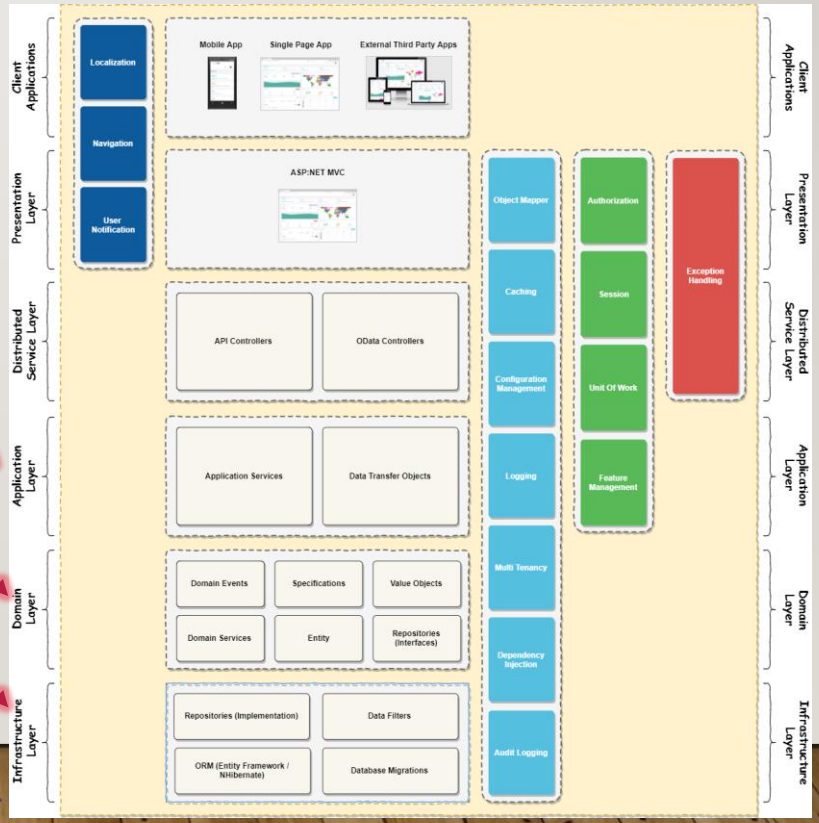
文件内容

- 类、字段、属性、方法、变量
- **命名规范**

2.1代码可维护性的本质是什么？
组织代码-项目层次



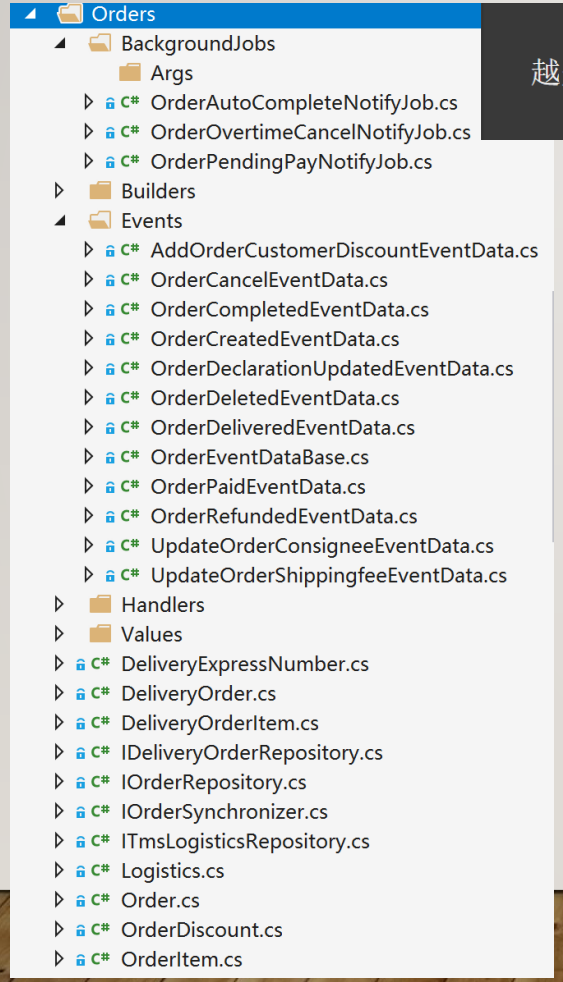
程序集命名体现了水平分层的层次



2.1代码可维护性的本质是什么？
组织代码-命名空间及目录

命名空间是垂直划分以组织代码

2.1代码可维护性的本质是什么？
组织代码-命名空间及目录



越是在上级空间中的对象，越具有可共享倾向（暂不考虑其他保护级别），越是往下级目录靠，其实就是越特殊

如果说，

> 程序集是水平划分代码（划分层次，程序集名称的最后一段代表其所在层次）

那么，

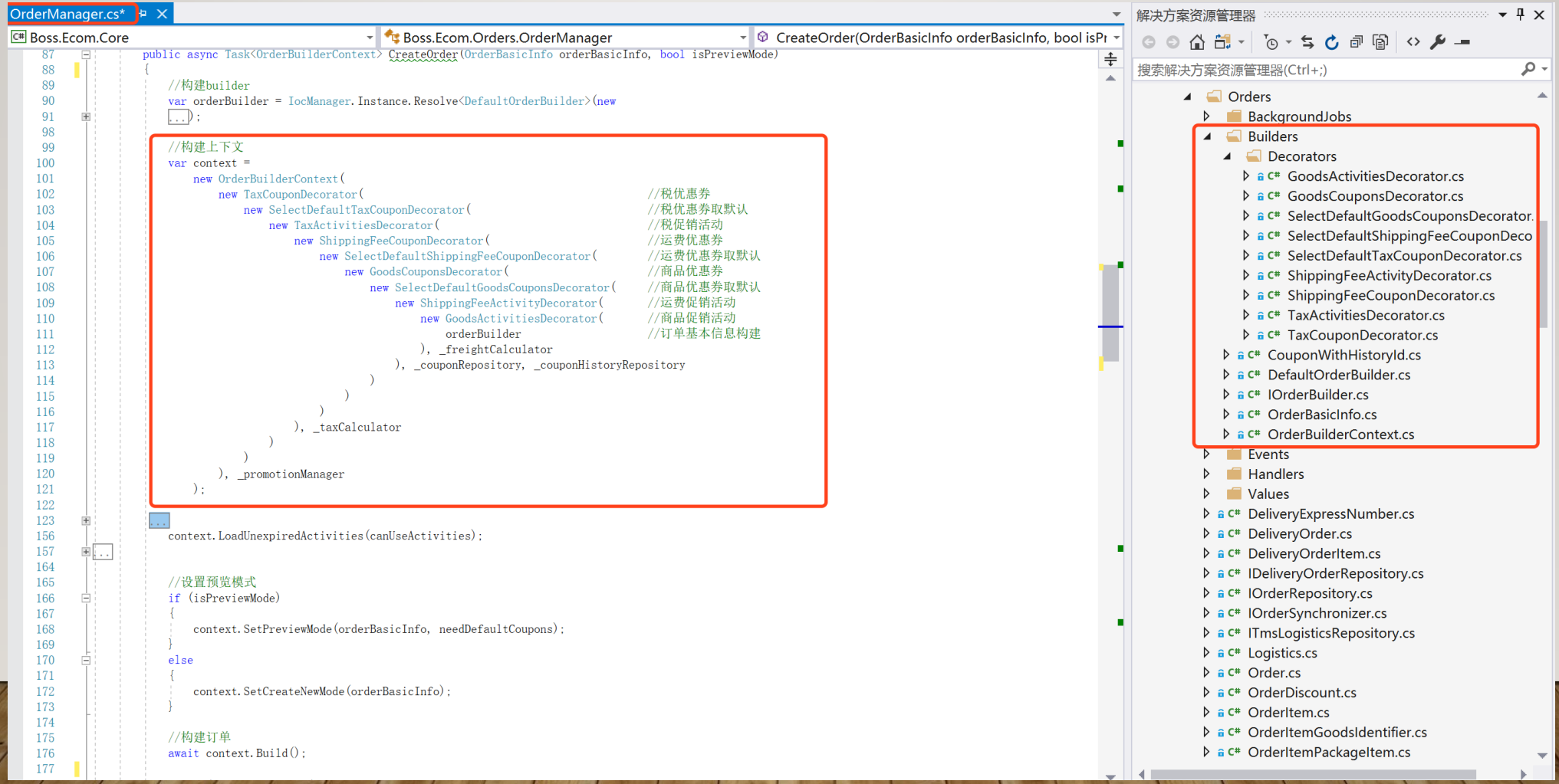
> 命名空间是垂直划分代码（划分概念，命名空间Personball.Demo.abc中的abc代表这部分代码所专注的领域和概念或者说是提供哪方面的功能）

这是正交的关系。

为了提供功能，必然需要跨越多个层次，在每个层次中都保证同样的目录结构（vs新建类或接口的所处命名空间），那么

> 同一个领域或概念在不同层次中是天然可见的，不需要特地用using语句引入命名空间

2.1代码可维护性的本质是什么？
组织代码-类文件和语句



2.1代码可维护性的本质是什么？

在可扩展性上，追求优雅。

代码的约束设计

- Access level
- internal、sealed、abstract、泛型约束等等
- 约束才能保证调用方式的一致

关注变更的影响范围

- 单一职责
- 关注点分离
- DI

OOP设计模式

- 几乎所有的编码实践原则实际都是在考虑代码可维护性问题

2.2代码可维护性还意味着什么？

无痛的阅读体验

人力切换成本低

项目稳定持续的演进

无后顾之忧的业务变更

可复用

易扩展

2.3代码的本质是?

指挥计算机

1和0

代码?

数据?

处理特定问题

2.4领域驱动设计（DDD）到底是啥？

领域边界

只关注边界内的业务

业务建模

实体、流程

一个模型，

易维护

易理解

易复用

跨职能交流

统一语言，降低沟通成本

不断学习、细化和完善业务

更快的响应变更

2.4领域驱动设计


push state change with
SERVICES
DOMAIN EVENTS
access with
REPOSITORIES
access with
express model with
express change with
ENTITIES
encapsulate with
express identity with
act as root of
encapsulate with
AGGREGATES
express state &
computation with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain
expressions with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich
model with
keep model
unified by
CONTINUOUS INTEGRATION
names
enter
SHARED KERNEL
SHARED KERNEL
BOUNDED CONTEXT
CORE DOMAIN
interdependent
contexts form
overlap allied
contexts through
relate allied contexts as
CUSTOMER /SUPPLIER
work in
autonomous, clean
assess/overview
relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
support multiple
clients through
segregate the
conceptual messes
free teams
to go
OPEN HOST SERVICE
translate and insulate
unilaterally with
loosely couple
contexts
through
formalize
GENERIC SUBDOMAINS
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS
PUBLISHED LANGUAGE

2.5关于在业务代码中应用设计模式的建议

```
/// <summary>
/// 分润账户余额（可提现余额）
/// </summary>
11 个引用|personball, 281 天前|1 名作者, 3 项更改
public int BalanceInCent { get; private set; }

/// <summary>
/// 冻结金额（提现中）
/// </summary>
8 个引用|personball, 281 天前|1 名作者, 3 项更改
public int FrozenInCent { get; private set; }
```

关闭setter

```
/// <summary> 余额增加
2 个引用|[redacted], 243 天前|2 名作者, 4 项更改
public PspAccountHistory Increase(int amountInCent, string remark)...

/// <summary> 余额冻结
1 个引用|personball, 320 天前|1 名作者, 2 项更改
public void Freeze(int amountInCent)...

/// <summary> 冻结解冻
1 个引用|personball, 320 天前|1 名作者, 2 项更改
public void Unfreeze(int amountInCent)...
/// <summary> 冻结扣减
1 个引用|personball, 320 天前|1 名作者, 2 项更改
public PspAccountHistory Decrease(int amountInCent, string remark)...
```

添加methods

```
public PspAccountHistory Increase(int amountInCent, string remark)
{
    if (amountInCent < 0)
    {
        throw new ArgumentOutOfRangeException(nameof(amountInCent), $"{nameof(amountIn
    }

    if (amountInCent == 0)
    {
        return null;
    }

    BalanceInCent += amountInCent;

    DomainEvents.Add(new Events.PspAccountIncreasedEventData
    {
        AccountId = this.Id,
        AccountLevelId = this.AccountLevelId,
        IncreasedAmountInCent = amountInCent,
        OrganizationId = this.OrganizationId,
        UserId = this.UserId,
        IncreasedType = increasedType
    });

    return history;
}
```

考虑事件

2.5关于在业务代码中应用设计模式的建议

欢迎加入宁波DOTNET社区