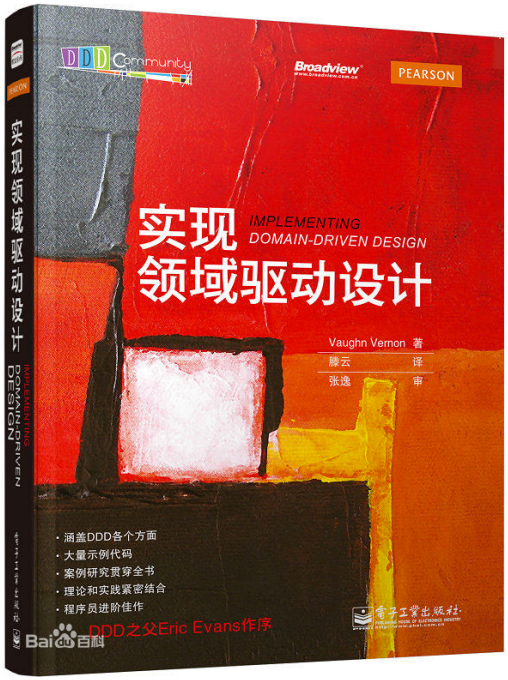
DDD—没那么难

滕云

自我简介

- 滕云
- ThoughtWorks 程序员
- Java/Linux/Devops/DDD
- 关注设计/架构/匠艺
- 《实现领域驱动设计》译者



领域

驱动

设计

不知所云

面向对象进阶

数据驱动 VS 领域驱动

忘掉你的数据库

数据驱动

- 数据库优先
- 算法和数据机械结合
- 技术导向
- 代码不能反映业务
- 业务逻辑分散
- 扩展性差

VS

领域驱动

- 领域模型优先
- 算法和数据有机结合
- 业务导向
- 代码即是设计
- 业务逻辑内聚
- 扩展性佳

行为饱满的领域对象

贫血对象

```
public class Person {
    private String name;
    private int age;
    private String phoneNumber;
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public String getPhoneNumber() { return phoneNumber; }
    public void setPhoneNumber(String phoneNumber) { this.pho
}
```

上帝对象

```
public class Order {
    private Item items;
    private Date placedDate;
    addItem();
    removeItem();


    private double price;
    private int discount;
    pay();
    discount();
    issueInvoice();


    private String address;
    private String phoneNumber;
    private String postCode;
    private Status currentStatus;
    getCurrentStatus();
    changeAddress();

}
```

选购

付款

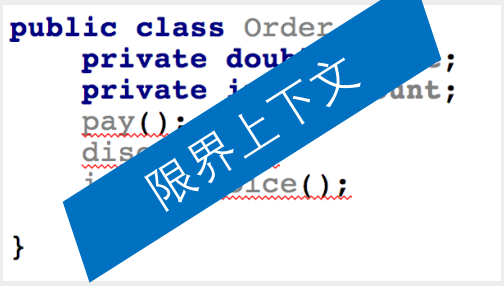
物流

怎么办?

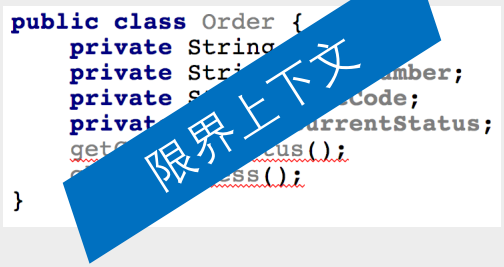
怎么拆?

选购

付款



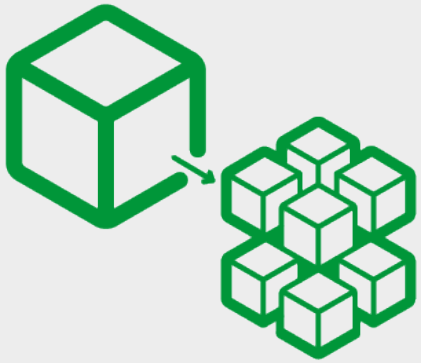
物流



一个领域概念在一个限界上下文中不应有二义性

再往前走一点点

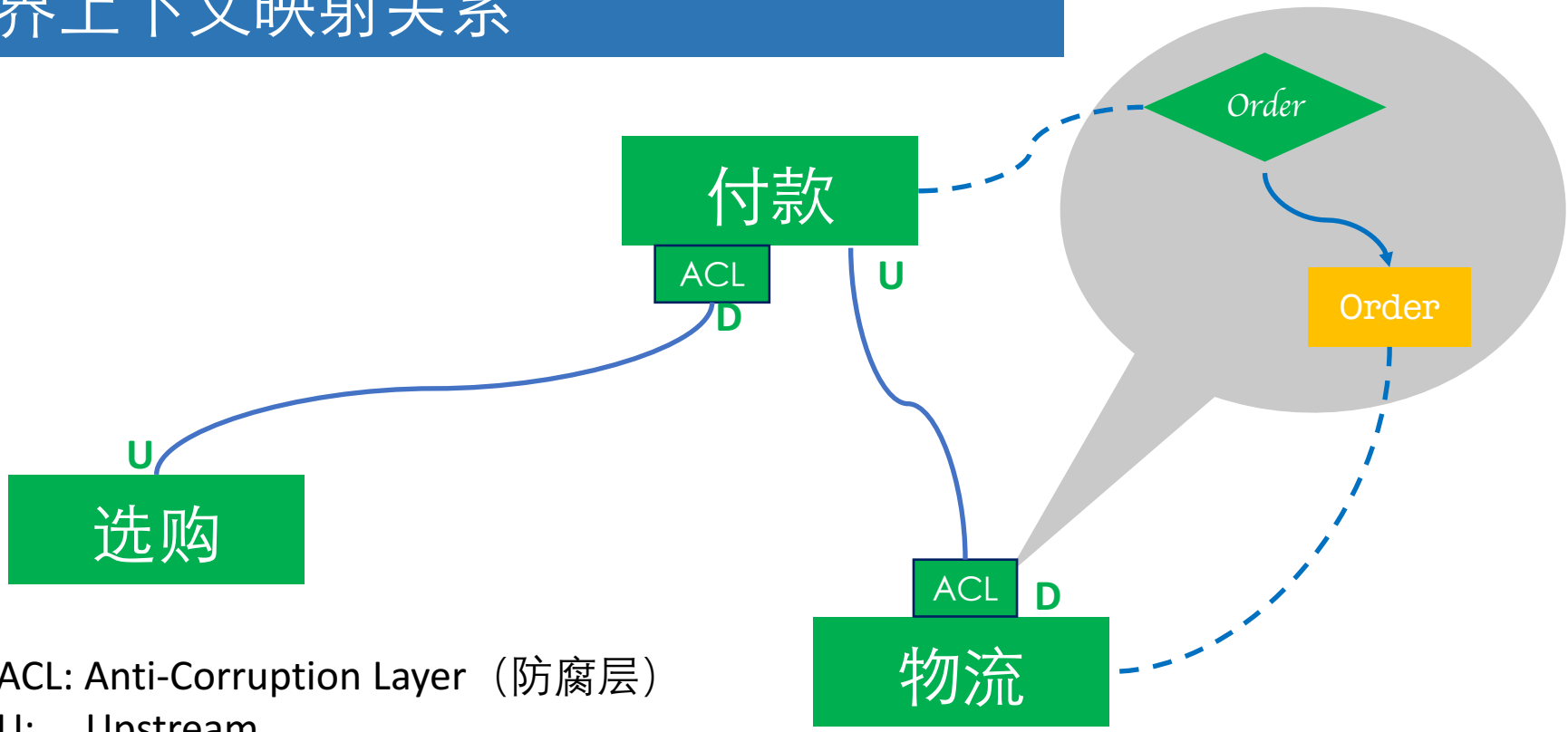
微服务



DDD

通过限界上下文拆分微服务

限界上下文映射关系

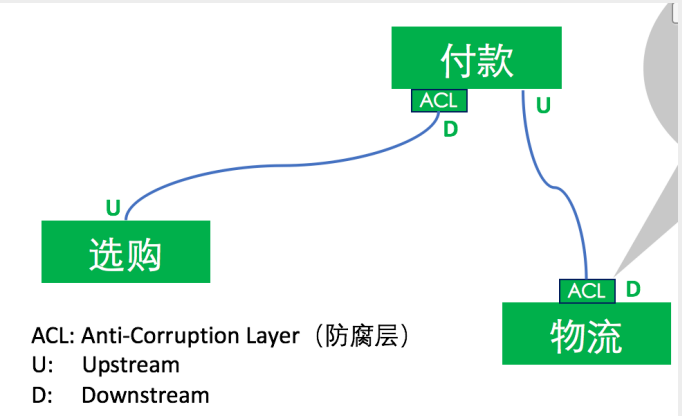


ACL: Anti-Corruption Layer（防腐层）
U: Upstream
D: Downstream

战略设计

战术设计

战略设计


付款
ACL
D
U
U
选购
ACL
D
物流
ACL: Anti-Corruption Layer（防腐层）
U: Upstream
D: Downstream

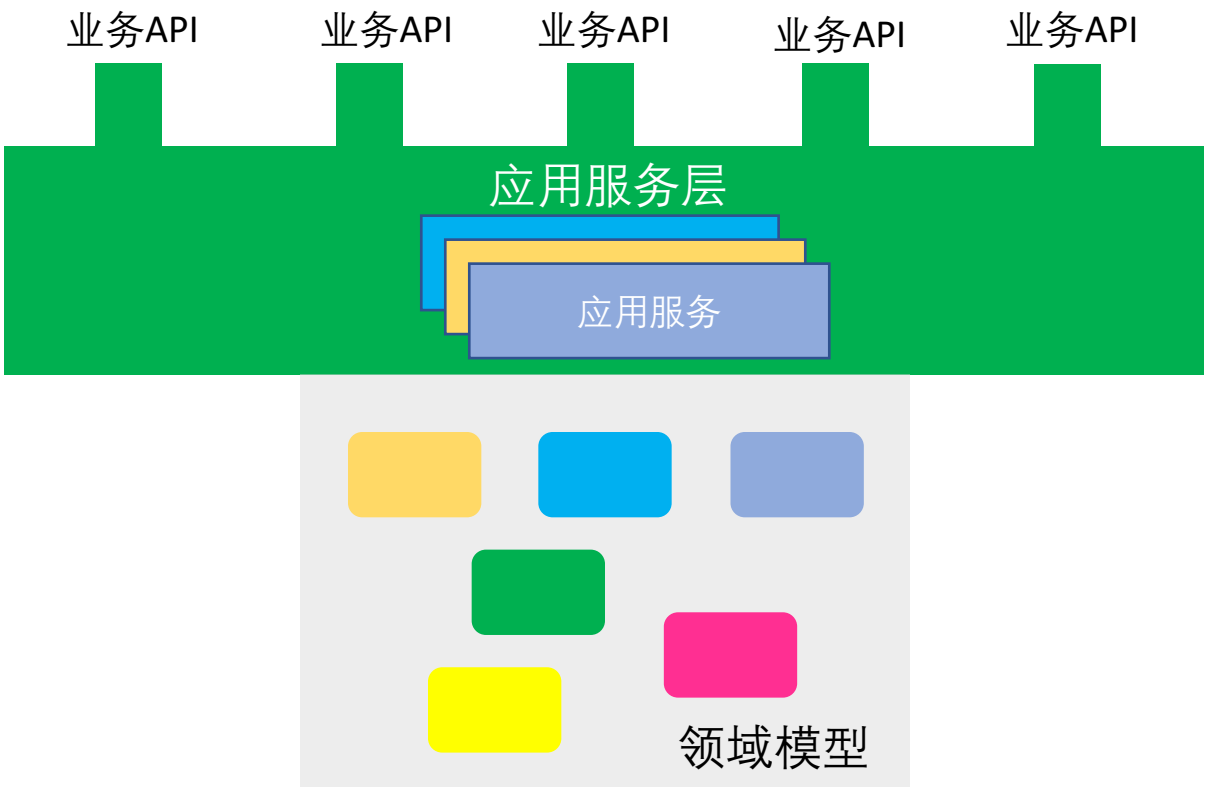
战术设计


领域事件
聚合
模块
工厂
实体
应用服务
聚合根
领域服务
值对象
资源库

战术设计

- 指导我们编码的
- 着眼于单个限界上下文

业务门面——应用服务(Application Service)



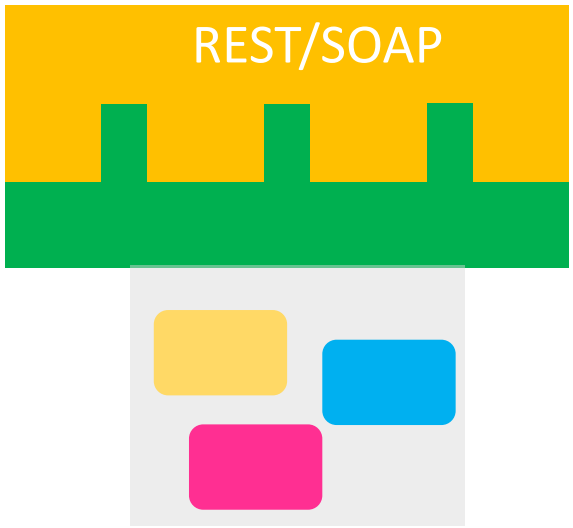
例子：更改订单物流地址

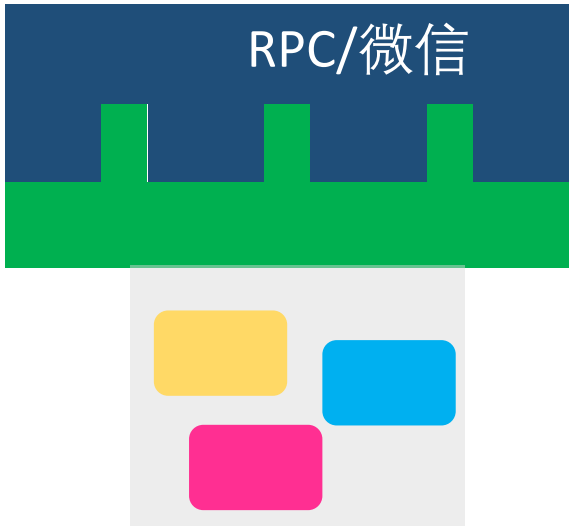
```
public class OrderApplicationService {
    @Transactional
    public void changeAddress(String orderId, String newAddress) {
        Order order = orderRepository.byId(orderId);
        order.changeAddress(newAddress);
        orderRepository.save(order);
    }
}
```

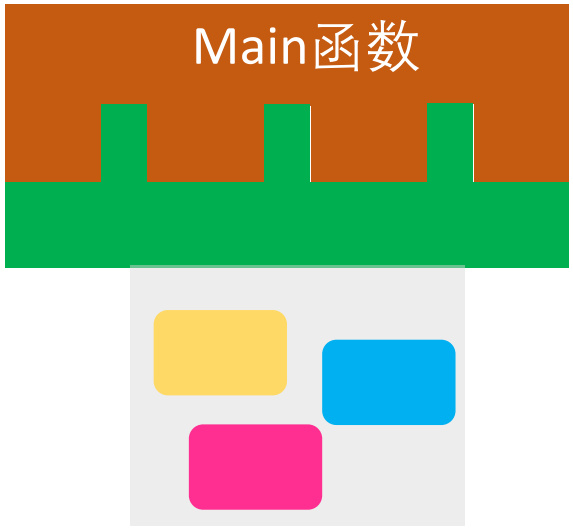
业务门面——应用服务(Application Service)

- 语言级别API
- 与UI/通信协议无关
- API与业务用例一一对应
- 很薄的一层
- 起协调代理作用
- 本身并不包含业务逻辑
- 事务边界

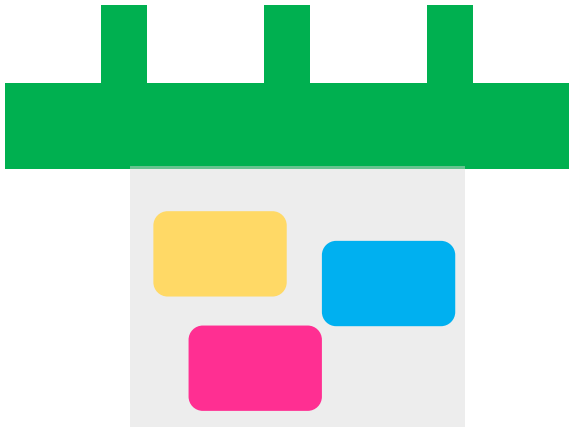
业务门面——应用服务(Application Service)



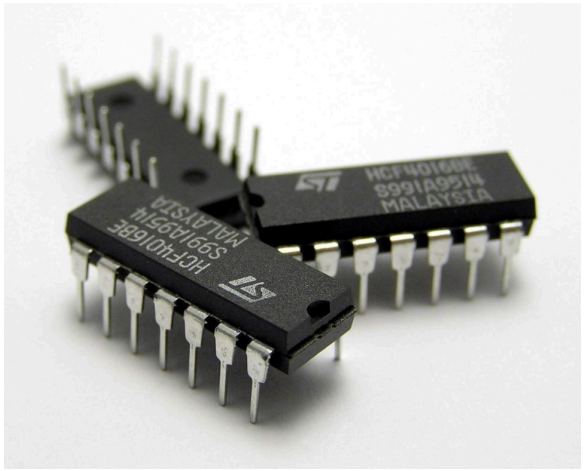




业务门面——应用服务(Application Service)



≈



业务逻辑到底放在什么地方?

业务逻辑到底放在什么地方?

- 核心领域对象
- 高度内聚
- 一致性边界

聚合根(Aggregate Root)

应用服务直接调用聚合根

```
public class OrderApplicationService {
    @Transactional
    public void changeAddress(String orderId, String newAddress) {
        Order order = orderRepository.byId(orderId);
        order.changeAddress(newAddress);
        orderRepository.save(order);
    }
}
```

聚合根

聚合根

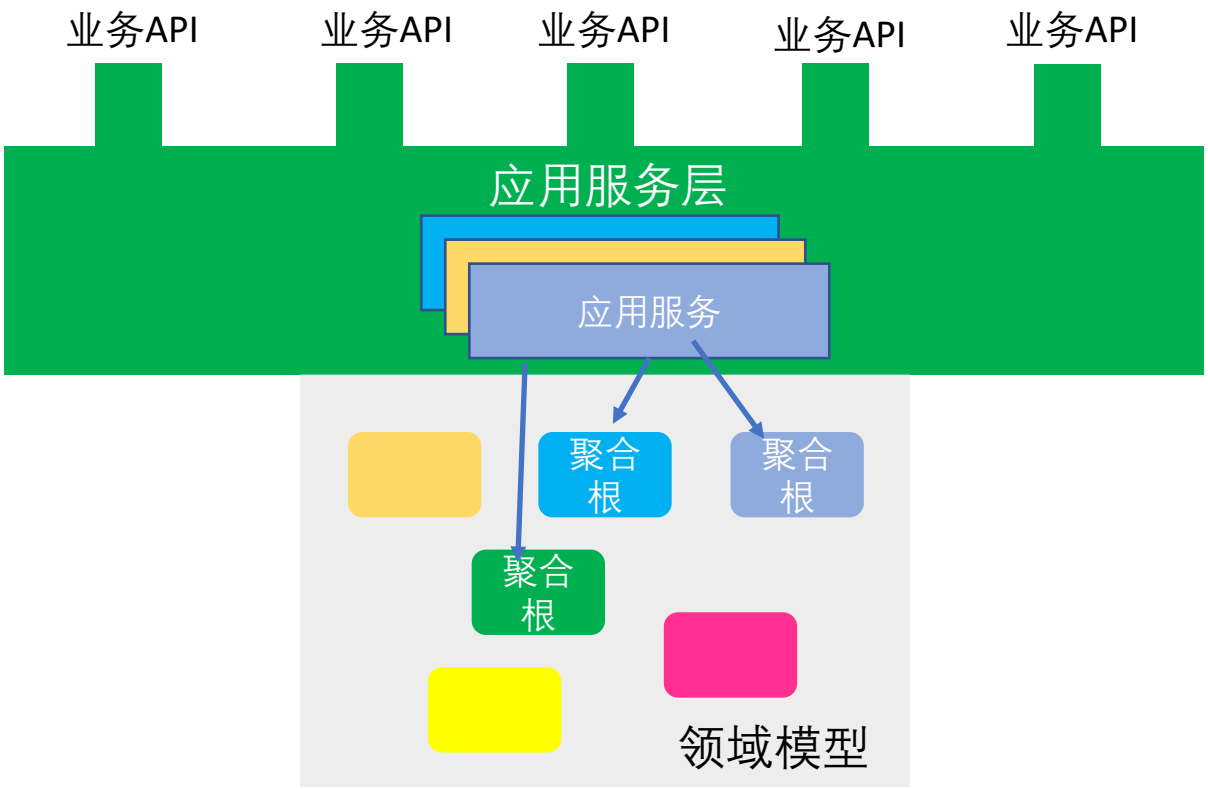
- 对外边界
- 其内部的所有业务操作都必须经过聚合根
- 一次业务用例对应一次事务对应一个聚合根
- 聚合根之间的数据一致性通过最终一致性完成

聚合根

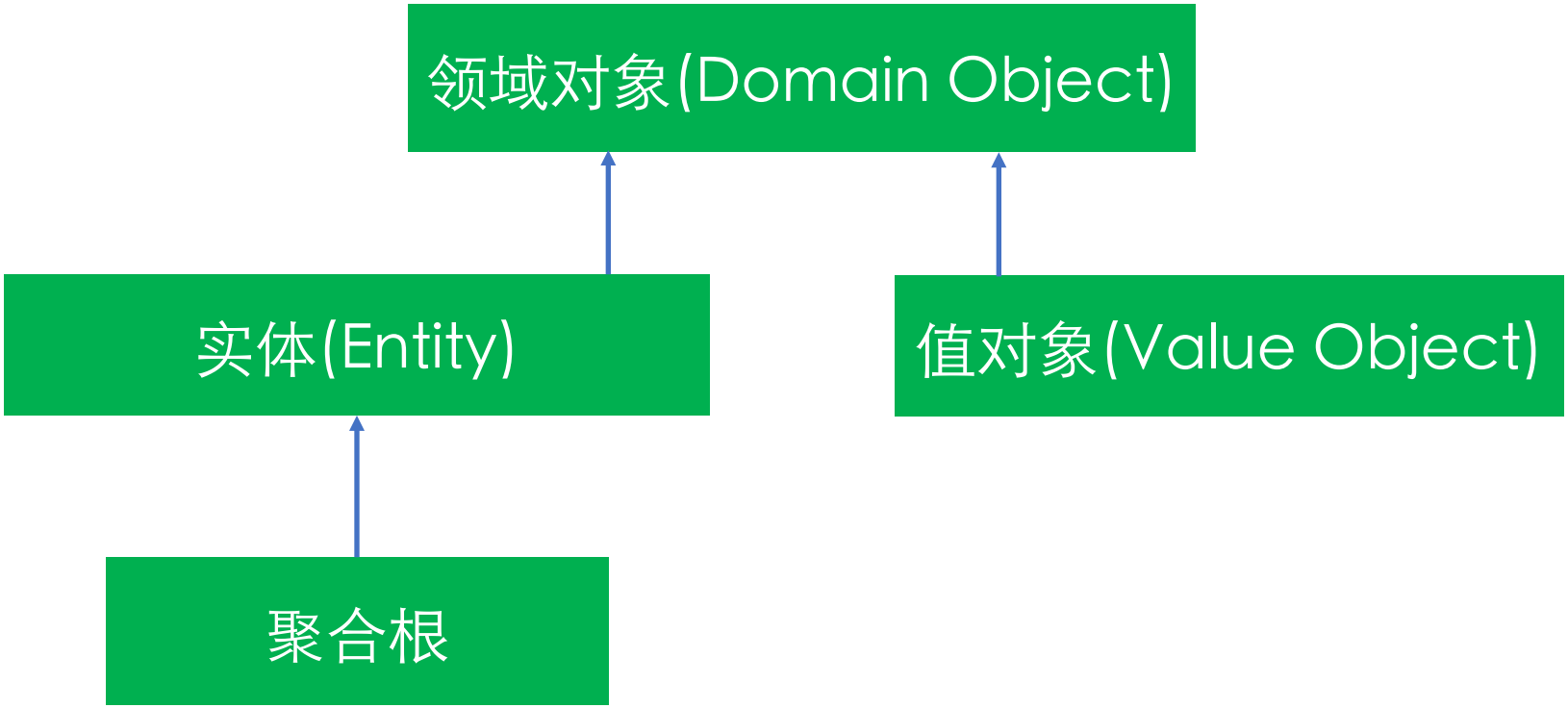
```
//Good
order.removeItem(item);


//Bad
List<Item> items = order.getItems();
items.remove(item);
```

应用服务直接调用聚合根



领域对象



实体

- 具有生命周期
- 有唯一标识
- 通过Id判断相等性
- 增删改查/持久化
- 可变
- 比如Order/Car

VS

值对象

- 起描述性作用
- 无唯一标识
- 通过属性判断相等性
- 实现Equals()方法
- 即时创建/用完即扔
- 不可变(Immutable)
- 比如Address/Color

尽量将领域概念建模成值对象


中国人民银行
100
壹佰圆
AA00000000
100
AA00000000

商品交易

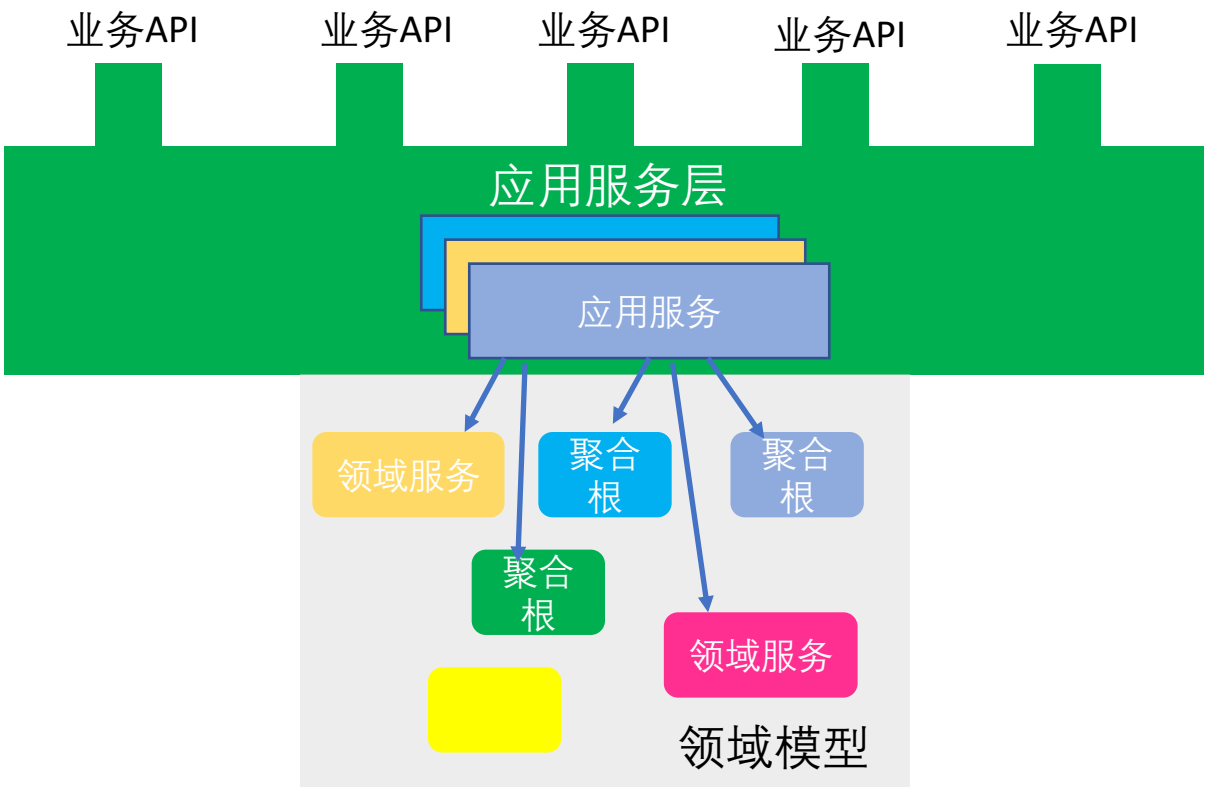
VS

货币追踪

有时领域逻辑放在领域对象上不合适

领域服务(Domain Service)

领域服务 vs 应用服务


业务API
业务API
业务API
业务API
业务API
应用服务层
应用服务
领域服务
聚合根
聚合根
聚合根
领域服务
领域模型

最后，数据库

ORM

DAO

Active Record

资源库(Repository)

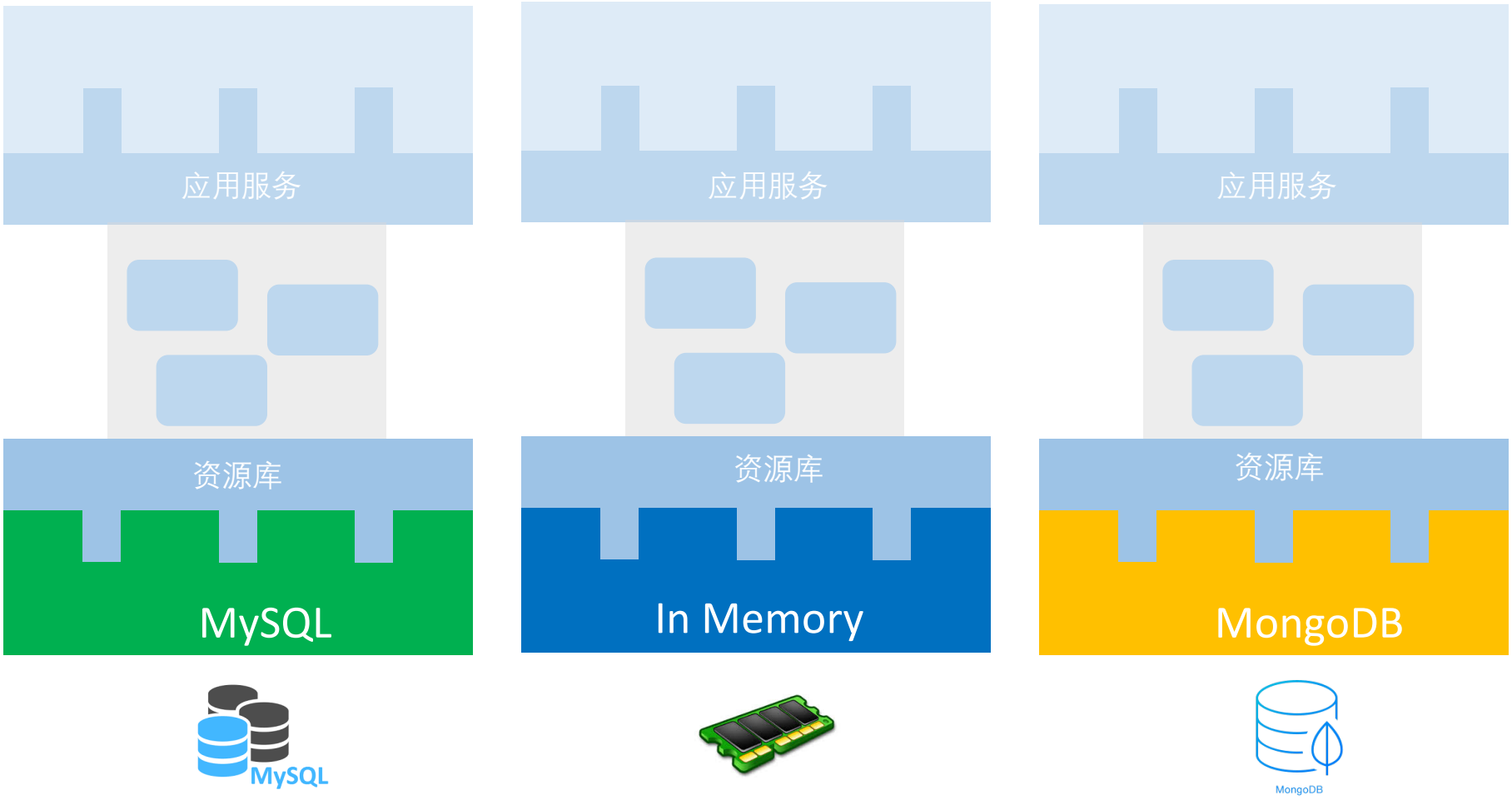
DAO VS 资源库

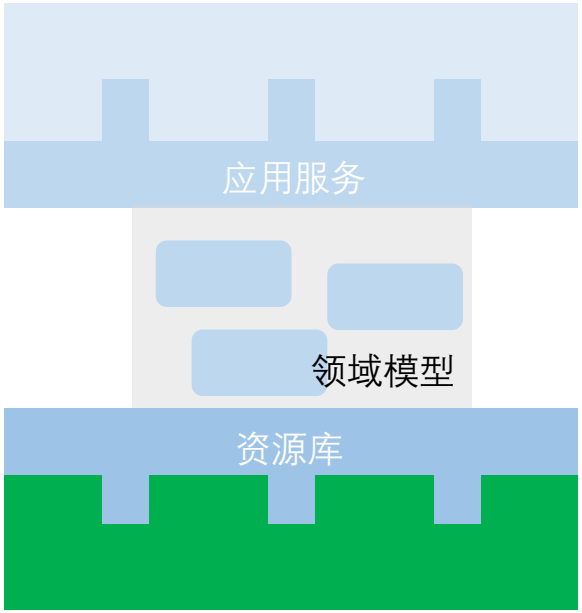
资源库

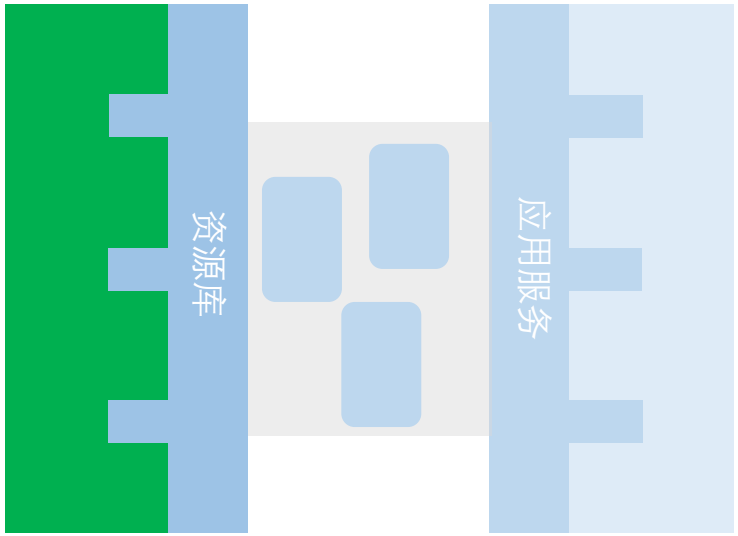
- 可插拔性
- 聚合根的集合

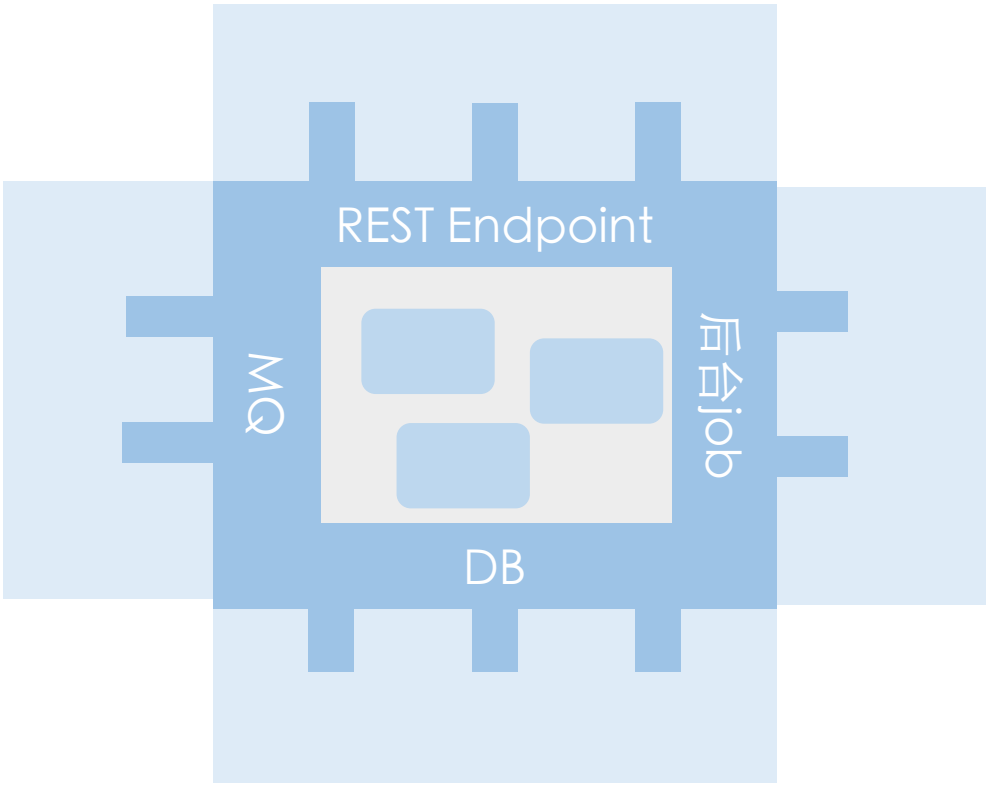
应用服务直接调用资源库(Repository)

```
public class OrderApplicationService {
    @Transactional
    public void changeAddress(String orderId, String newAddress) {
        Order order = orderRepository.byId(orderId);
        order.changeAddress(newAddress);
        orderRepository.save(order);
    }
}
```

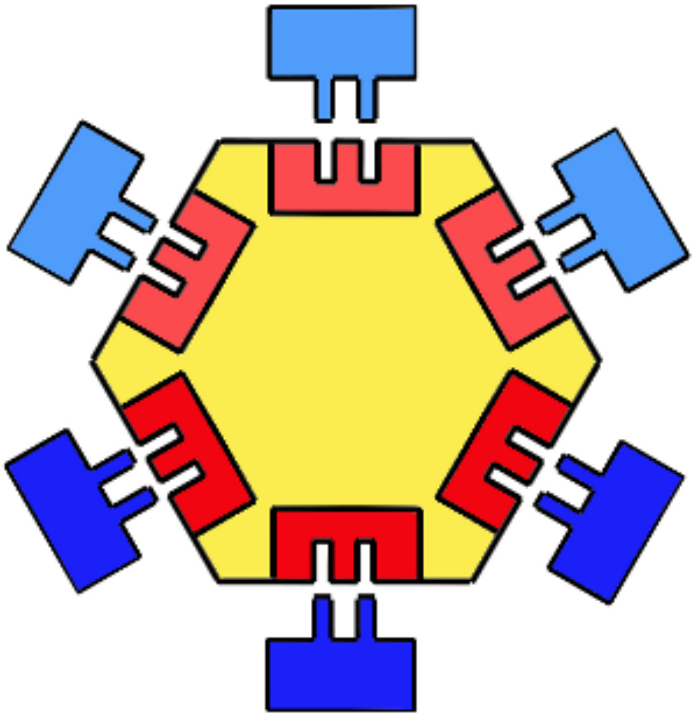

应用服务
资源库
MySQL
应用服务
资源库
In Memory
应用服务
资源库
MongoDB
MySQL
MongoDB

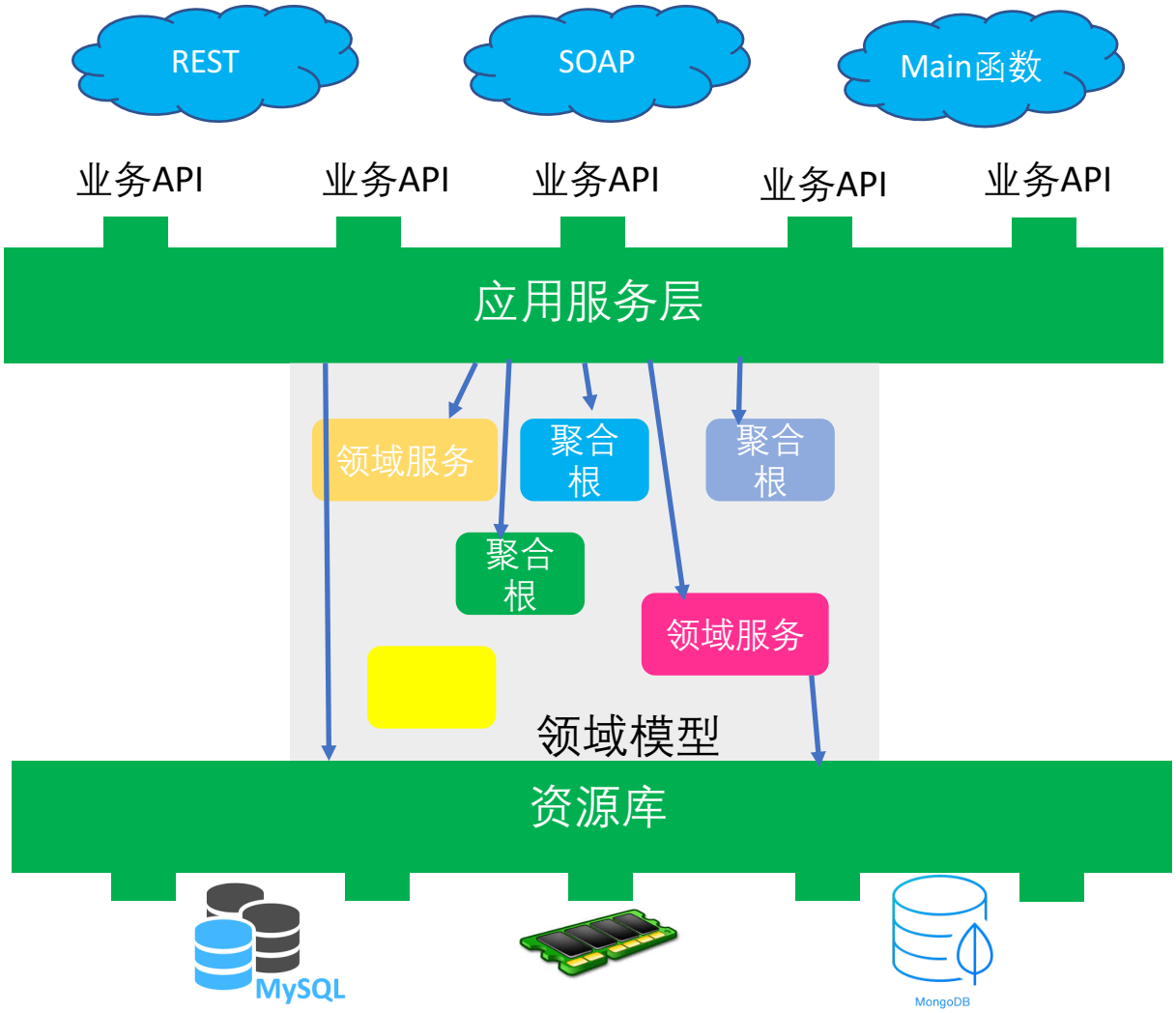

应用服务
领域模型
资源库


资源库
应用服务


REST Endpoint
MQ
后台job
DB

六边形架构




REST
SOAP
Main函数
业务API
业务API
业务API
业务API
业务API
应用服务层
领域服务
聚合根
聚合根
聚合根
领域服务
领域模型
资源库
MySQL
MongoDB

谢 谢