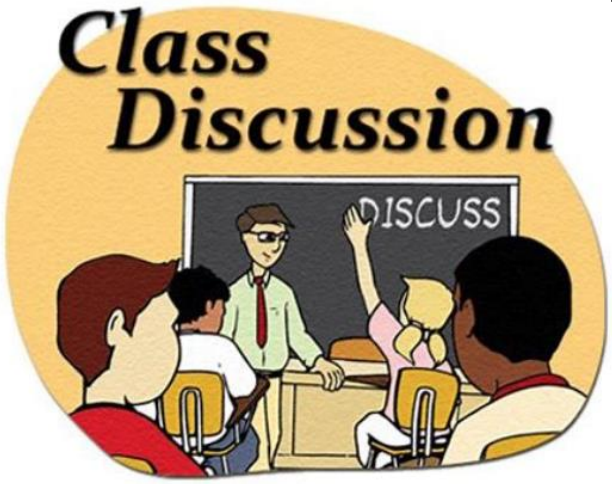
CS2001
软件工程

13. 软件设计
—软件体系结构

课件来源　　复旦大学软件学院

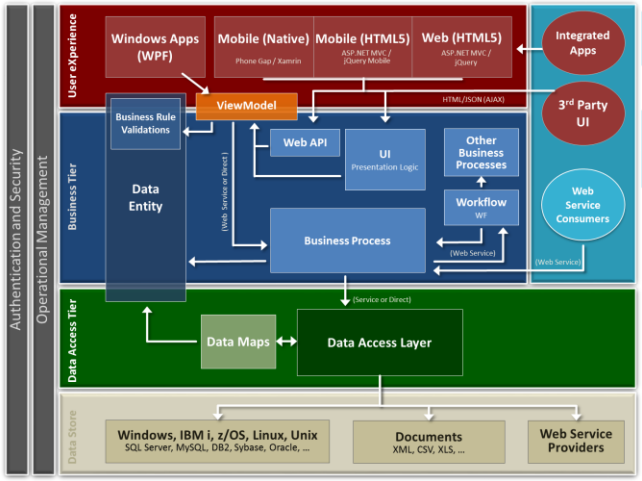
课堂讨论：软件体系结构

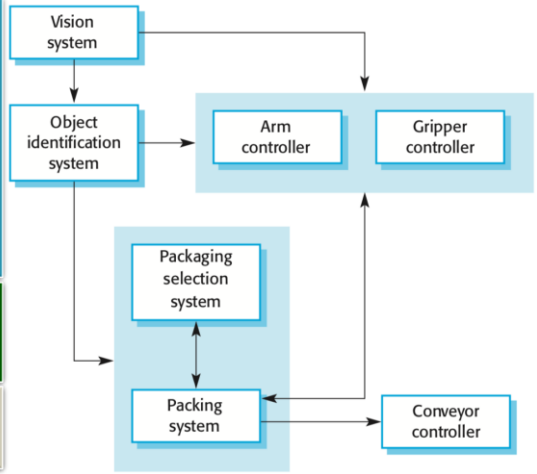


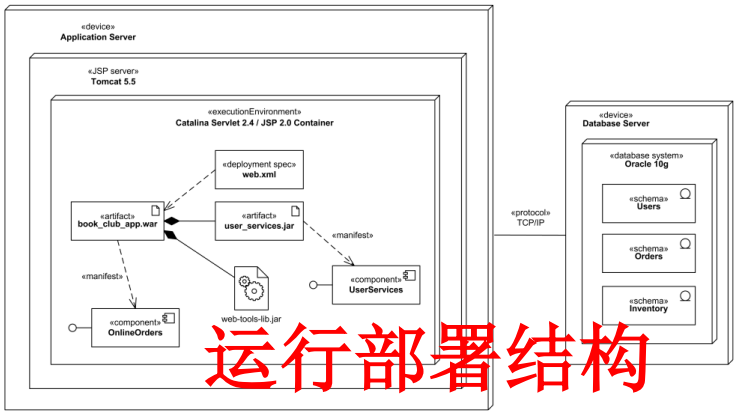
课堂讨论：软件体系结构（Software Architecture）是什么含义？与前面讲的那些软件设计是什么关系、有什么区别？

?

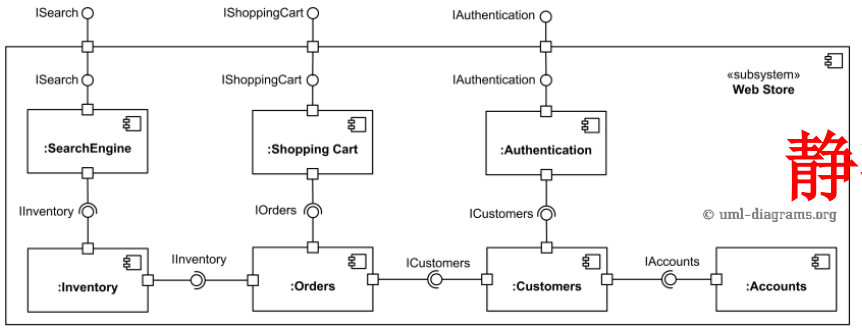
软件体系结构长什么样?



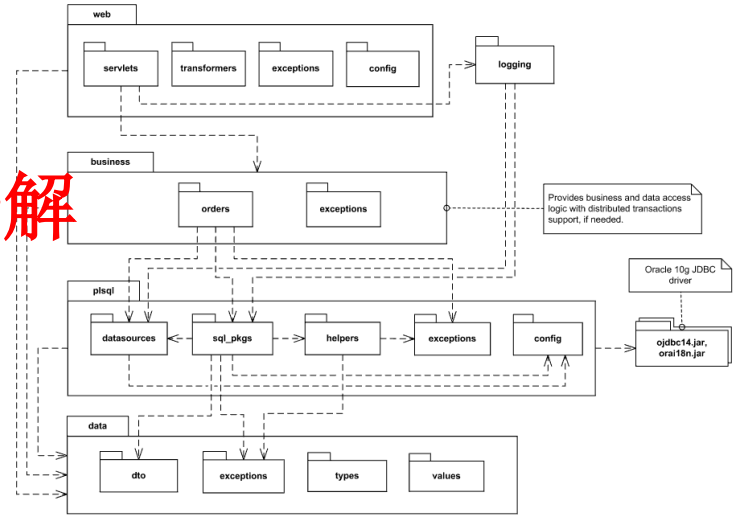


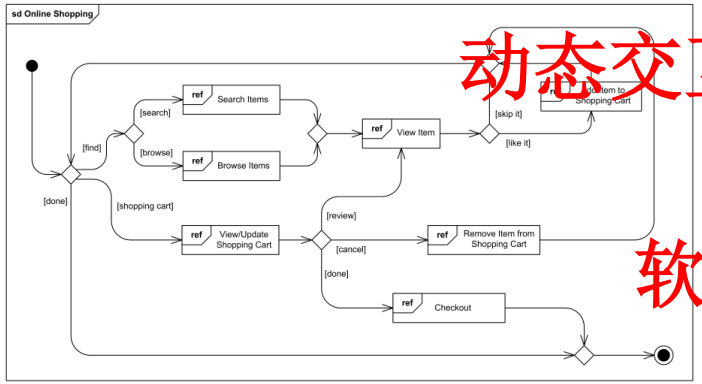


运行部署结构



静态结构分解





动态交互过程

软件系统的高层模块/子系统/组件分解

软件体系结构

- 软件系统的高层设计结构
- 反映软件解决方案的高层分解
- 具体包括
  - ✓ 一组软件组件
  - ✓ 软件组件的外部属性
  - ✓ 软件组件之间的关系
  - ✓ 全局的实现约定

软件体系结构的内容-1

- 软件组件
  - ✓ 表示软件系统高层分解的结构元素
  - ✓ 可以是简单的程序模块或类，也可以是复杂的子系统
  - ✓ 还可以包含扩展的数据库、中间件等
- 软件组件的外部属性
  - ✓ 功能性属性，例如接口、交互协议
  - ✓ 非功能性属性，例如响应时间、吞吐量

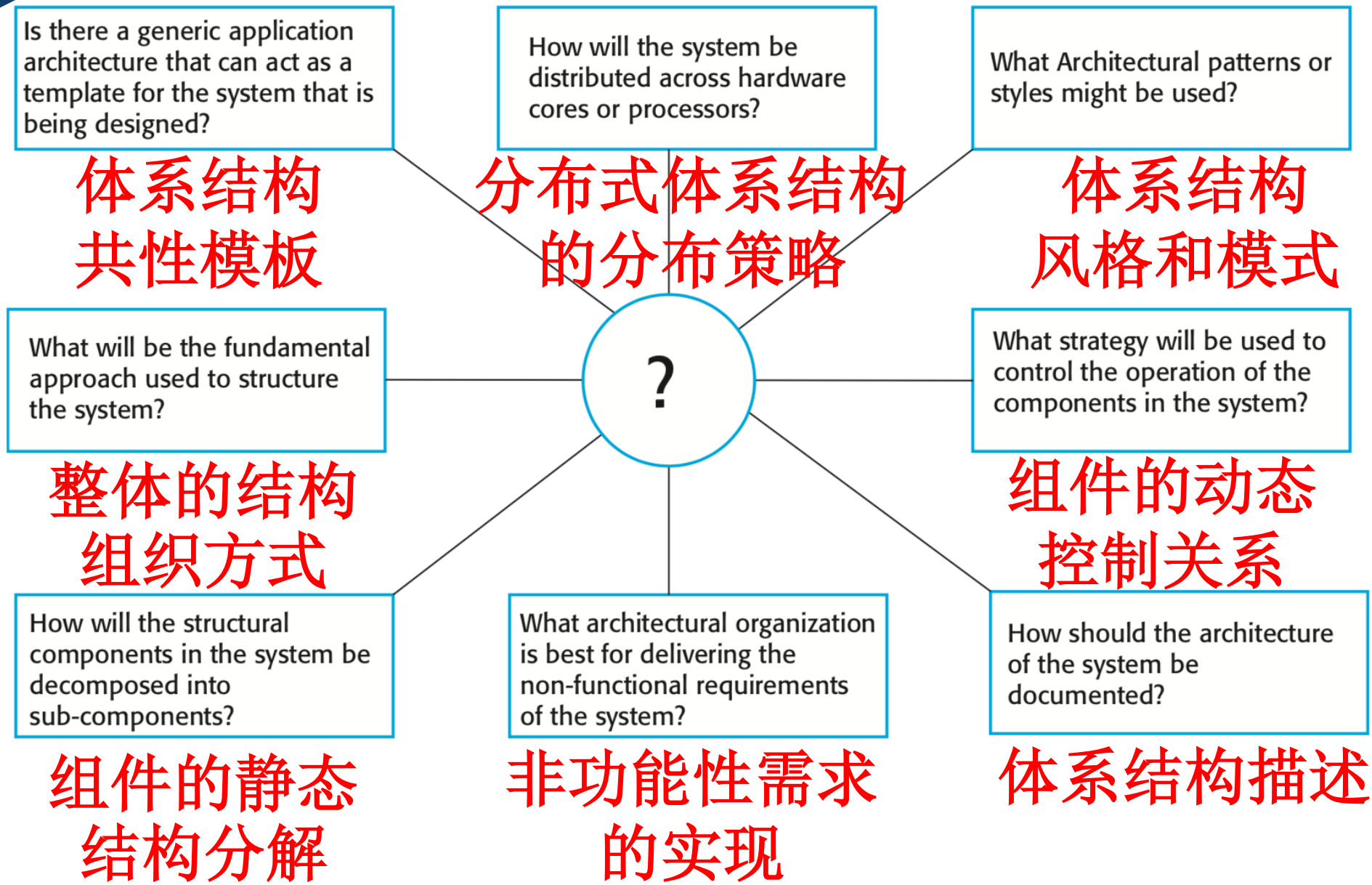
软件体系结构的内容-2

- 软件组件之间的关系
  - ✓ 静态结构关系
  - ✓ 动态交互关系
  - ✓ 运行时部署关系
- 全局的实现约定
  - ✓ 需要在系统全局作出约定的实现技术决策
  - ✓ 例如：实现语言、异常处理策略、共享资源的使用方式、包和文件的命名方式等
  - ✓ 影响最终的系统组件集成

软件体系结构的作用

- 软件系统实现方案的高层抽象
- 用于在软件开发的早期
  - ✓ 分析设计方案是否能有效满足需求
  - ✓ 考虑多种候选的技术方案
  - ✓ 识别并降低软件实现的风险
- 为后续设计、实现和测试活动中的相关涉众提供沟通和交流的基础

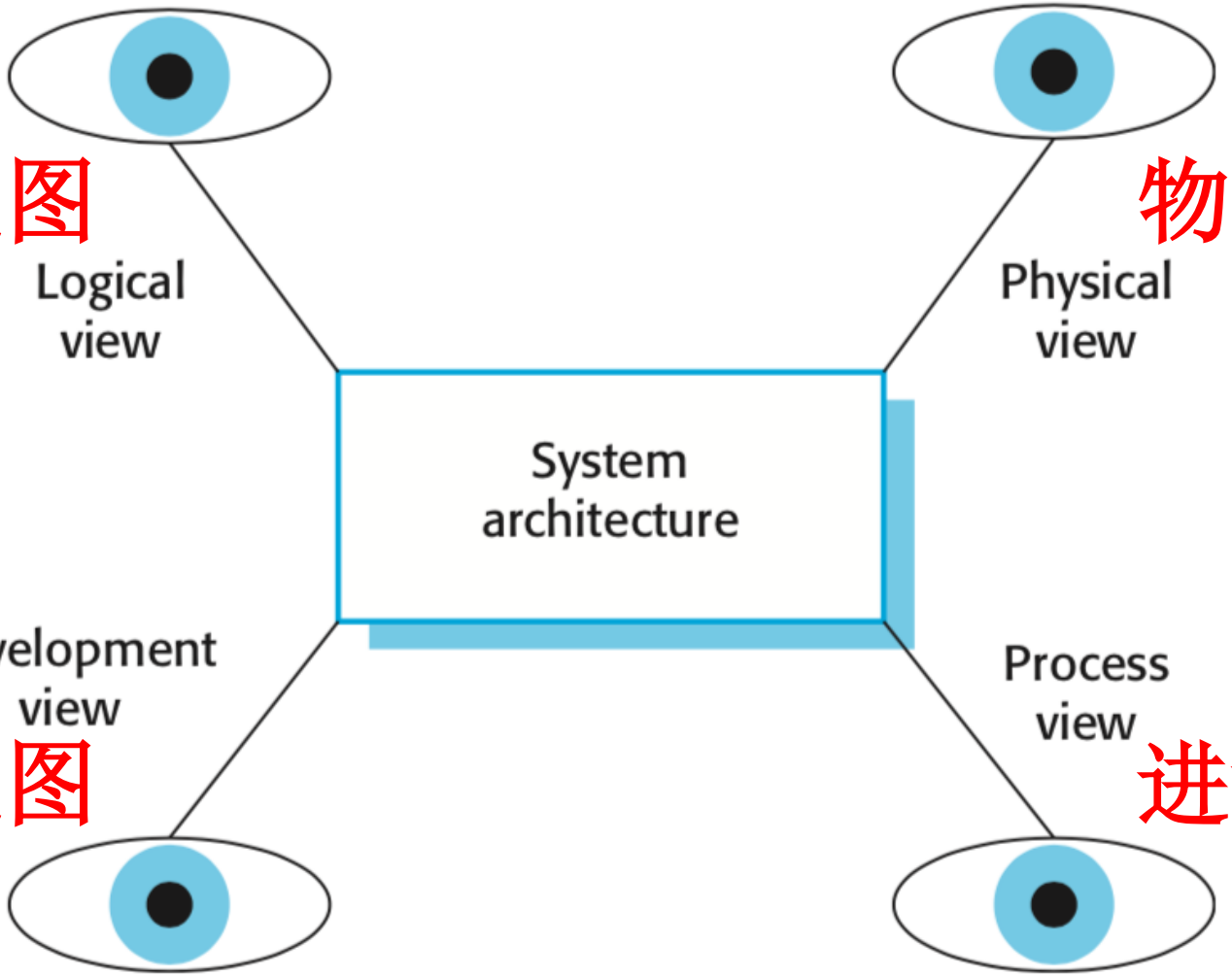
软件体系结构设计决策


Is there a generic application architecture that can act as a template for the system that is being designed?
体系结构共性模板
How will the system be distributed across hardware cores or processors?
分布式体系结构的分布策略
What Architectural patterns or styles might be used?
体系结构风格和模式
What will be the fundamental approach used to structure the system?
整体的结构组织方式
?
What strategy will be used to control the operation of the components in the system?
组件的动态控制关系
How will the structural components in the system be decomposed into sub-components?
组件的静态结构分解
What architectural organization is best for delivering the non-functional requirements of the system?
非功能性需求的实现
How should the architecture of the system be documented?
体系结构描述

非功能性质量与体系结构设计决策

性能（Performance）	多副本部署、负载均衡、关键性操作局部化
信息安全（Security）	层次化体系结构，关键性资产在最内层保护并进行高级安全确认
安全性（Safety）	安全性相关操作集中化，引入保护系统在失效时安全关闭系统
可用性（Availability）	使用冗余组件，支持不停机情况下的组件更换或更新
可维护性（Maintainability）	使用易修改的细粒度、自包含的组件，松耦合（如数据生产者与消费者分离，避免共享数据结构）

软件体系结构的多视图



《软件工程》6.2节

体系结构风格（Style）和模式（Pattern）

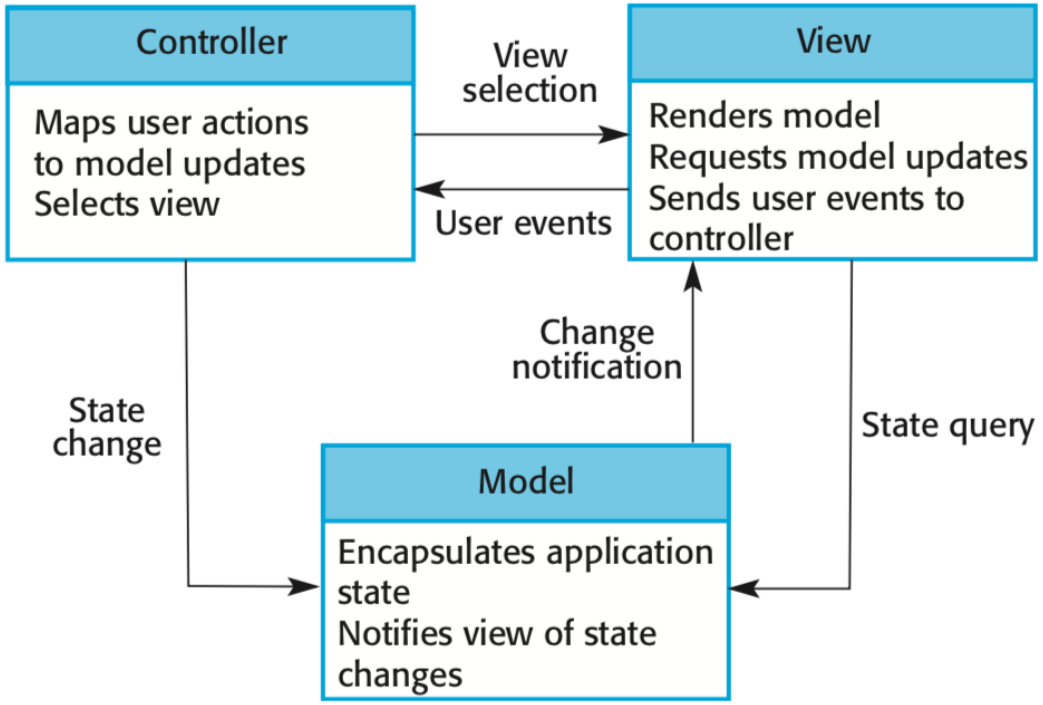
类比：建筑风格

哥特式建筑

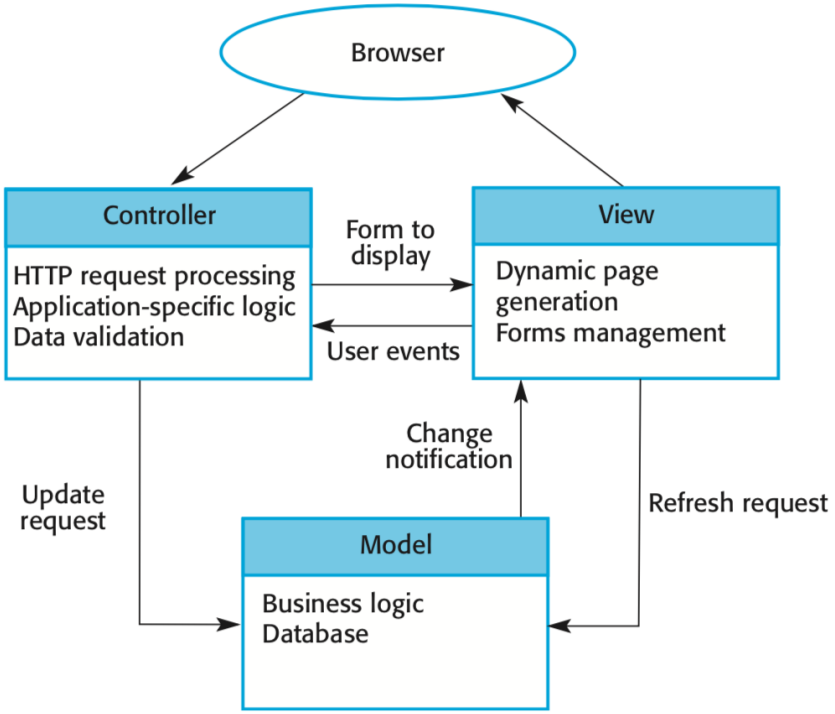
中国古典园林

巴洛克风格

MVC体系结构模式



MVC模式



使用MVC模式的Web应用体系结构

课堂讨论：软件体系结构模式与此前介绍的面向对象设计模式有什么区别？

软件体系结构模式

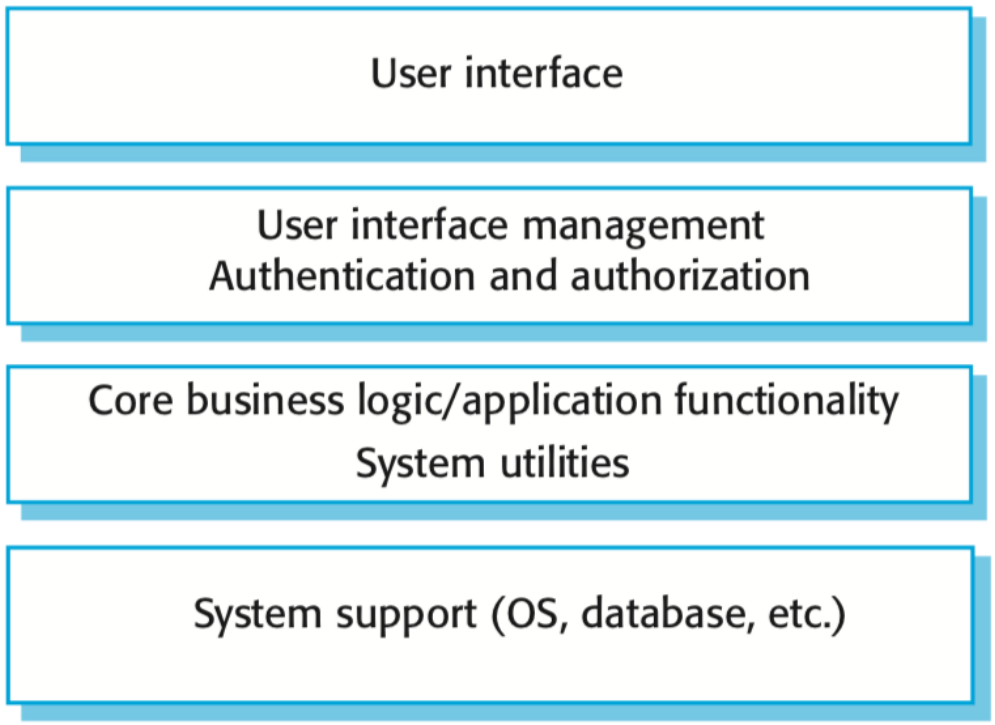
- 适合于特定设计要求、经过实践验证的体系结构设计样式
- 对于系统中所包含的组件及其交互关系的特点进行了限定
- 与面向对象设计模式相比更加宏观
  - ✓ 其中的元素往往不是类而是模块或组件
  - ✓ 对整个系统中结构组织和交互形态进行了约定，不限于某个局部设计

常用的软件体系结构模式

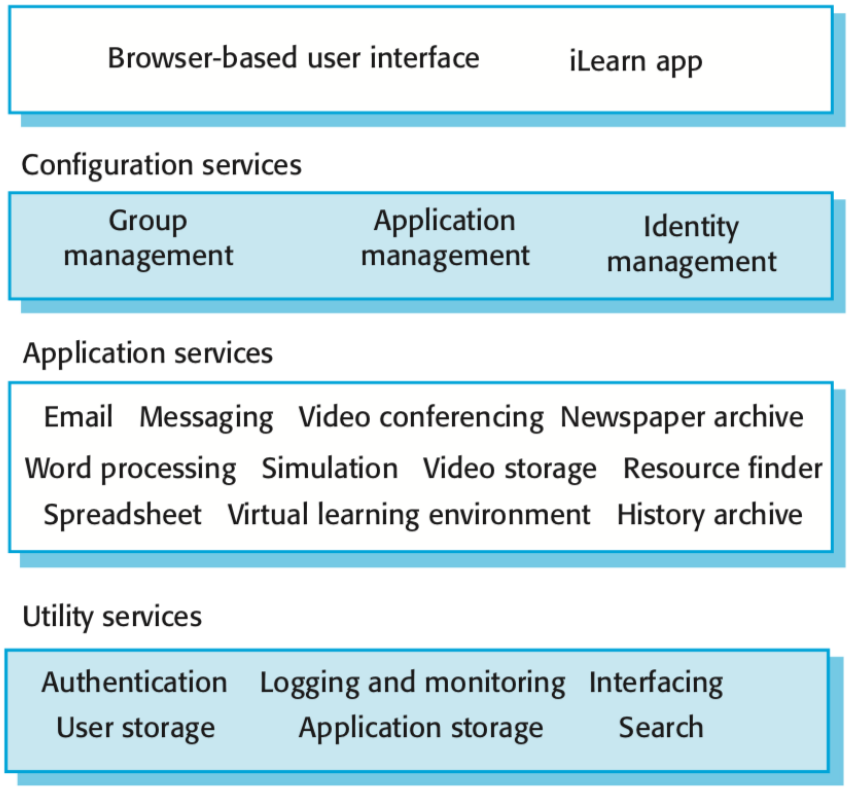
- 分层体系结构
- 知识库体系结构
- 客户端-服务器体系结构
- 管道-过滤器体系结构

《软件工程》6.3节

分层体系结构



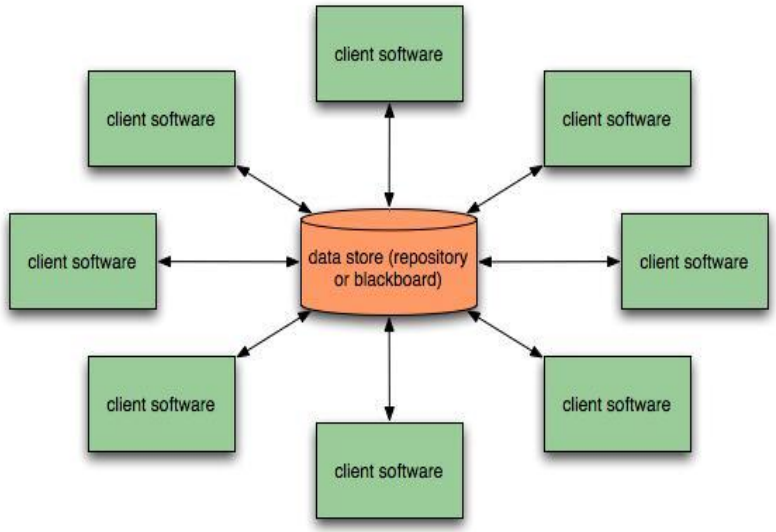
通用分层体系结构



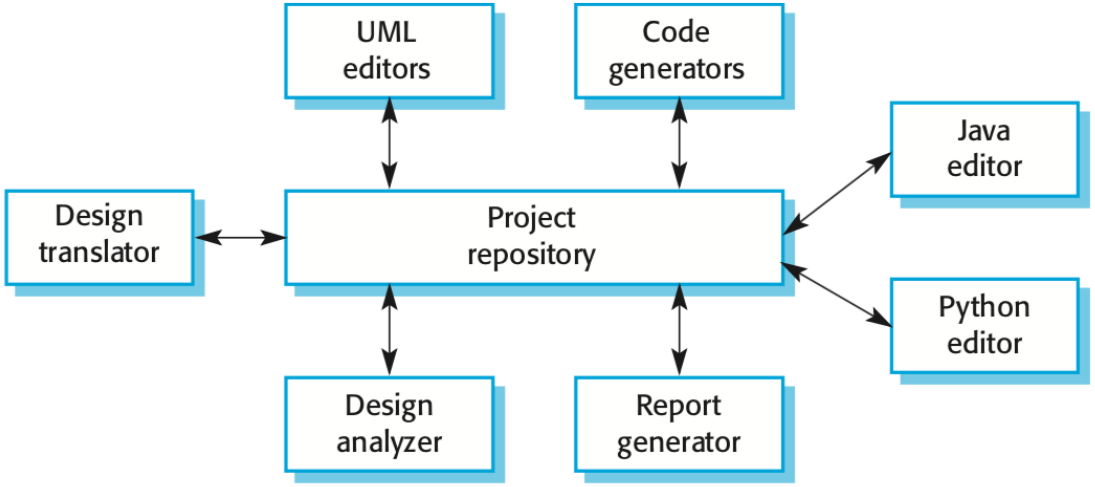
iLearn系统分层体系结构

将系统组织为多个层次，每个层次对上提供服务
层间接口通常会保持稳定

知识库体系结构



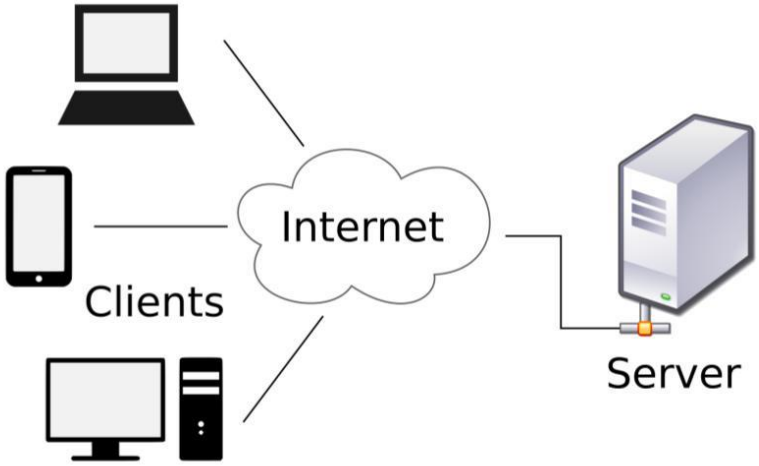
通用知识库体系结构



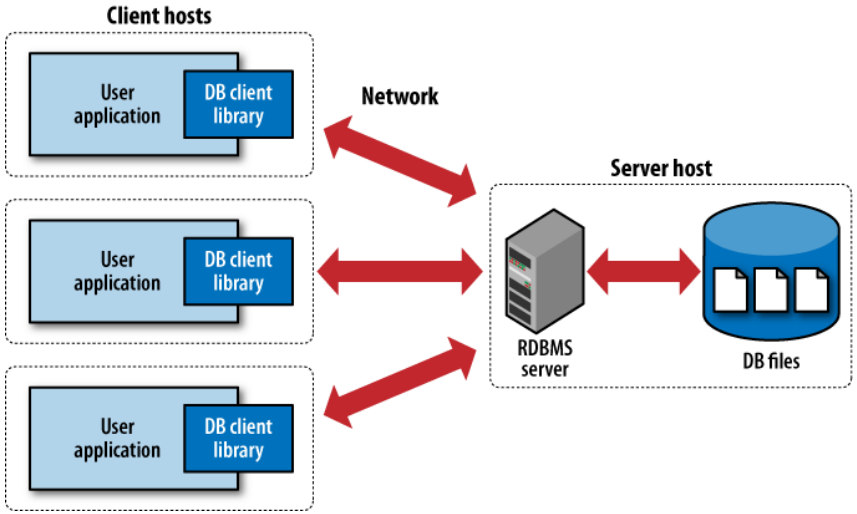
集成开发环境（IDE）的知识库体系结构

系统级数据在中心知识库中进行管理
组件之间通过共享知识库中数据的方式进行交互

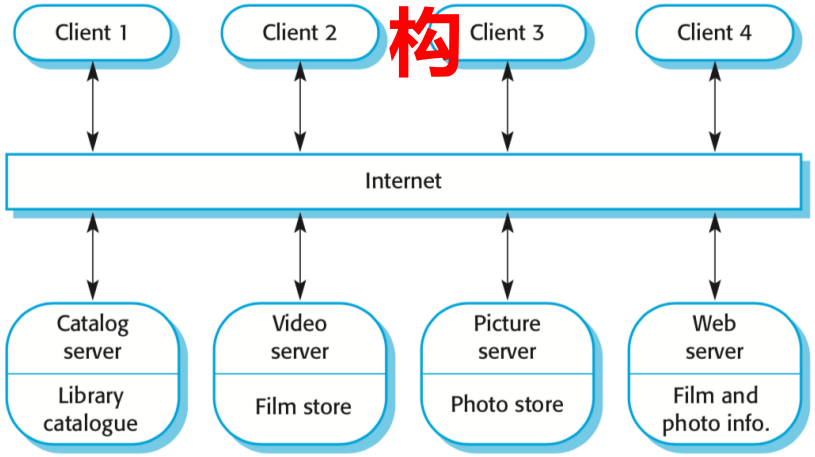
客户端-服务器（C/S）体系结构



客户端-服务器体系结构



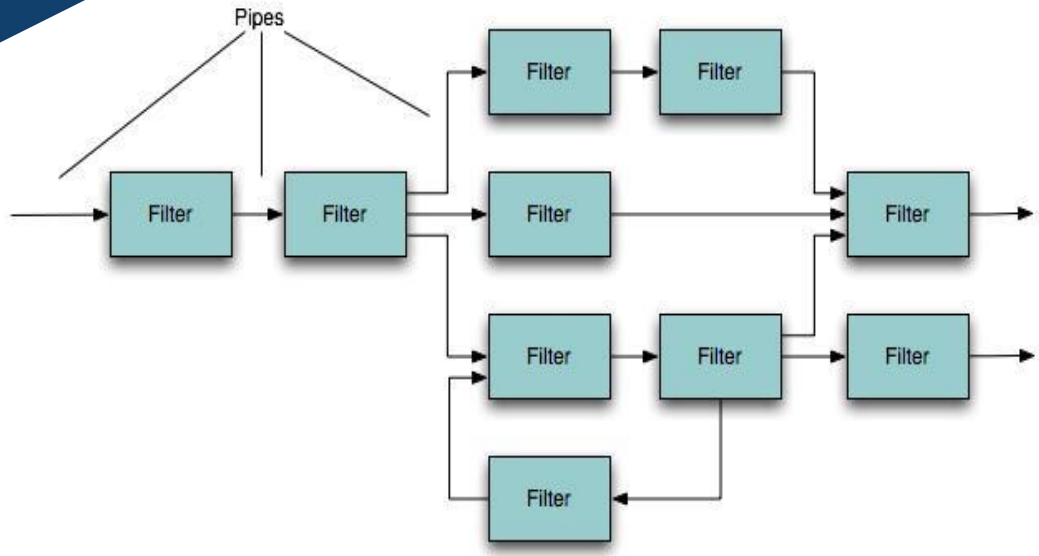
以数据库为中心的客户端-服务器体系结构



一个电影库的客户端-服务器体系结构

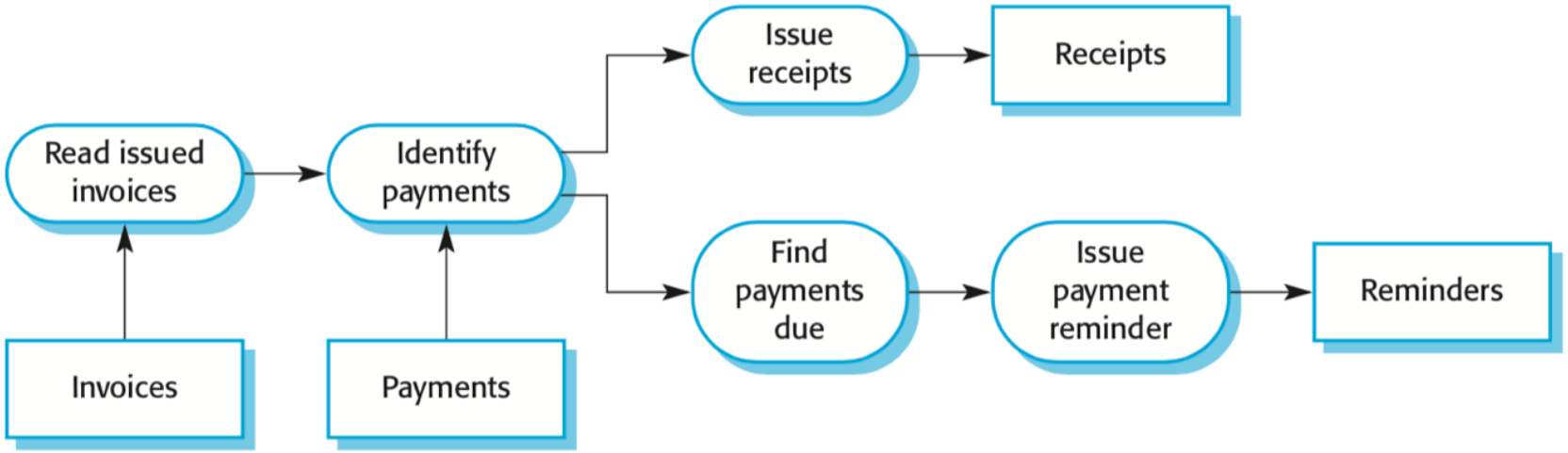
系统的通用核心功能被组织为一组服务，多个客户端通过网络共享这些服务。

管道-过滤器体系结构



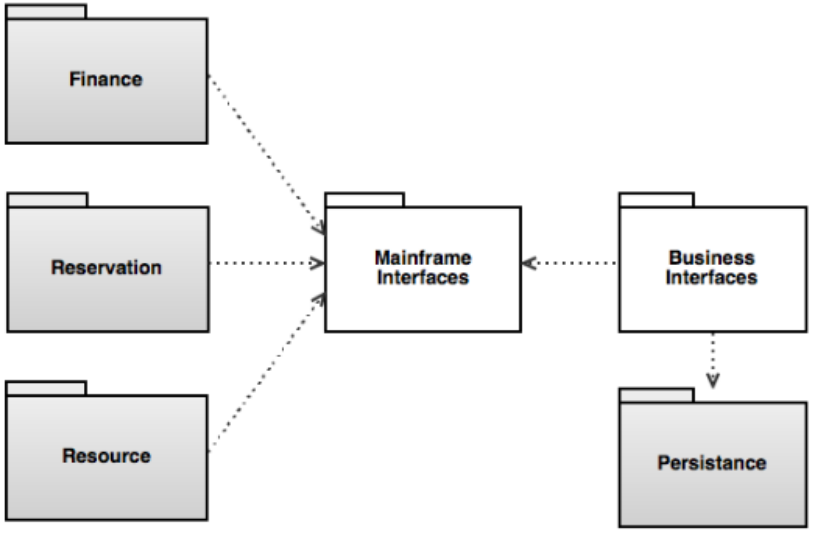
处理组件（过滤器）负责进行数据处理，连接器（管道）负责组件间的数据流动

通用管道-过滤器体系结构



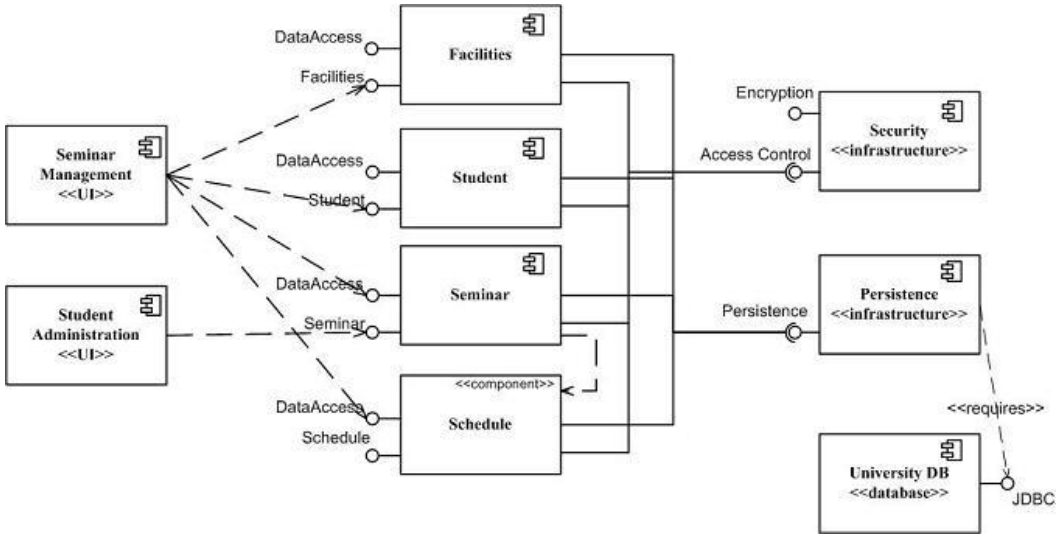
一个收款系统的管道-过滤器体系结构

软件体系结构的UML描述：静态结构



包图

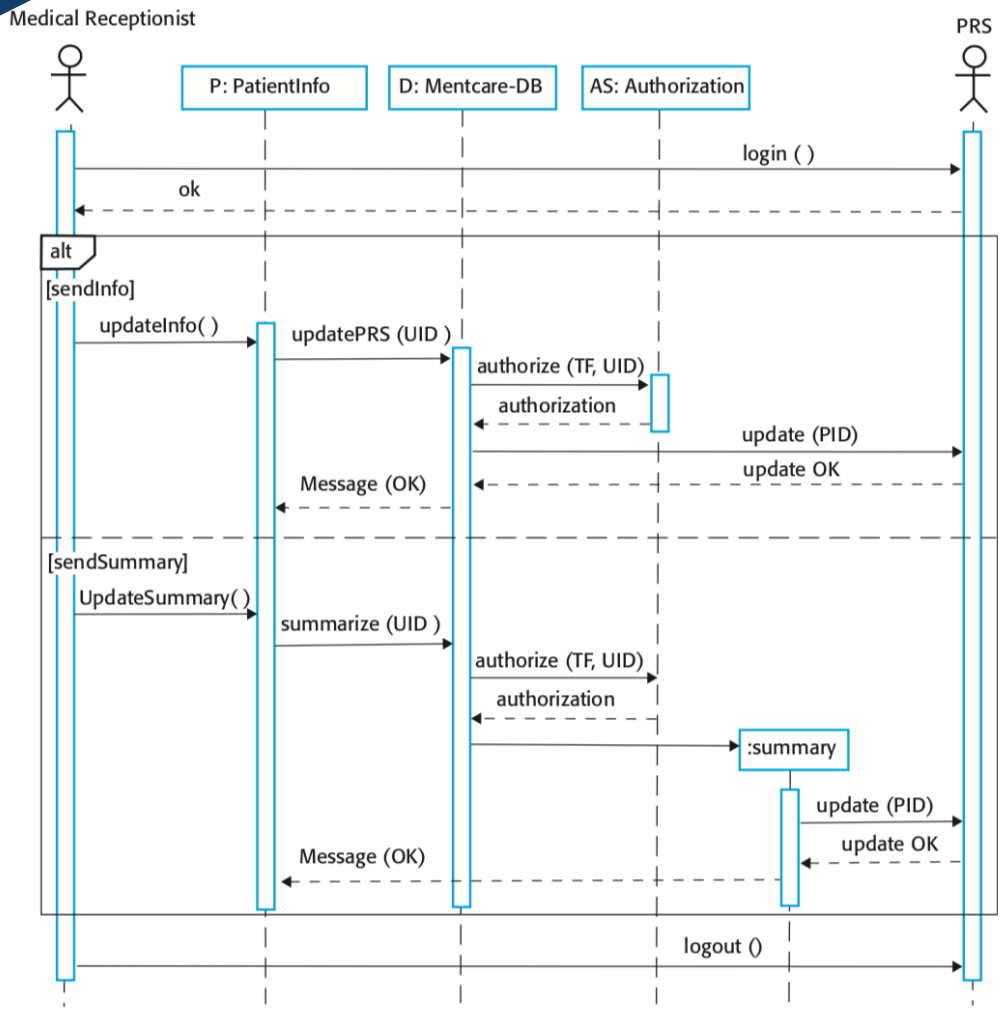
（Package Diagram）



组件图

（Component Diagram）

软件体系结构的UML描述：动态交互

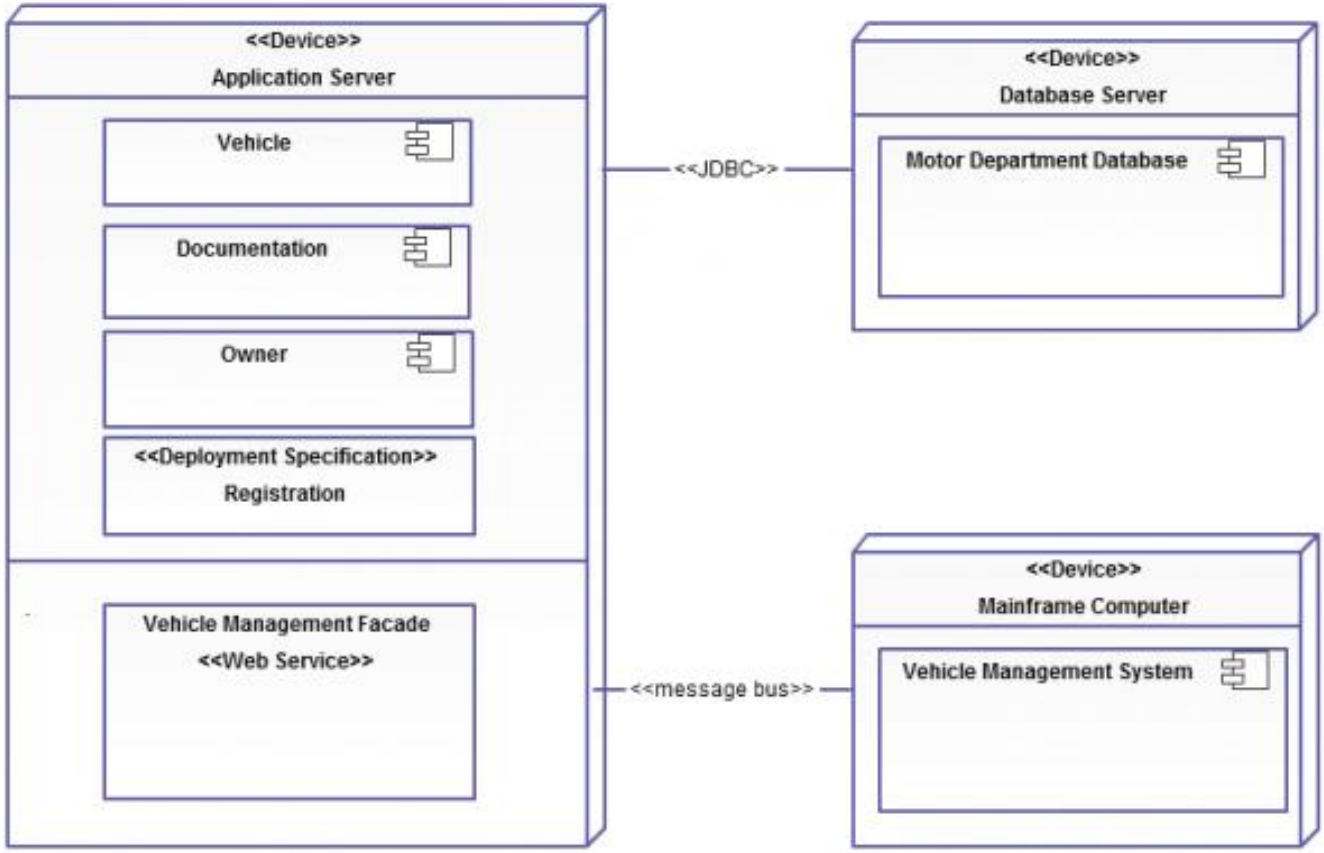


顺序图

（Sequence Diagram）

《软件工程》5.2.2节

软件体系结构的UML描述：部署结构



部署图

（Deployment Diagram）

应用体系结构案例

Software Engineering 10th Edition

Ian Sommerville's book website

Search Search

About the book/me | Case studies | Instructors Guide | Slides | Videos | Downloads | My websites

Application architectures

There are many different types of application and these each have their own individual architecture. Many of these individual architectures are instances of more general, application specific architectural patterns. Some of these have been discussed in Chapter 6 but I include examples of 3 further application architecture patterns here namely:

- Batch data processing systems
- Resource allocation systems
- Event processing systems

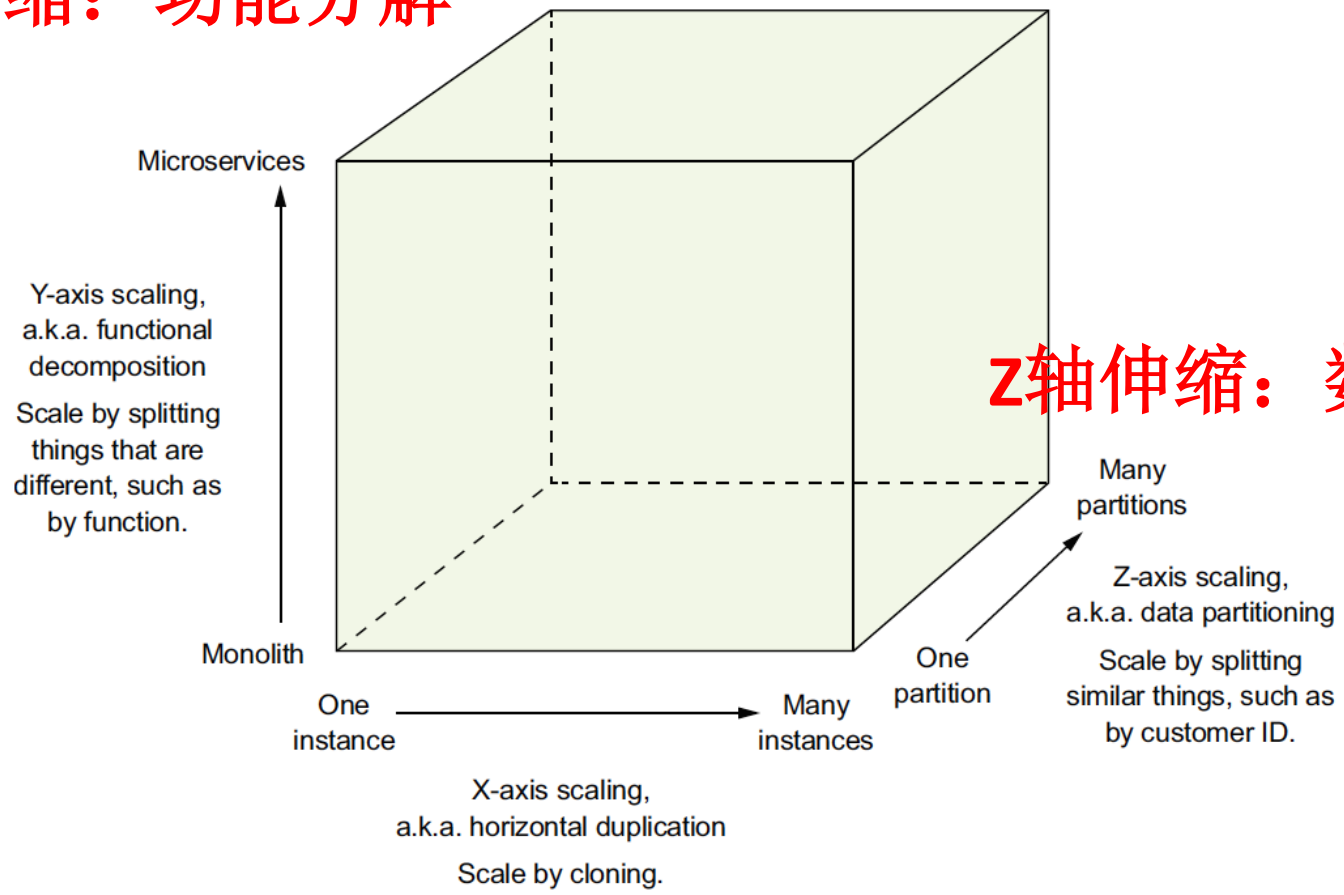
http://iansommerville.com/software-engineering-book/web/apparch/

Web软件体系结构的发展历程

- 单体服务器
- 多服务器负载均衡
- 面向服务的体系结构
- 云+容器+微服务

应用程序的扩展立方体

Y轴伸缩：功能分解

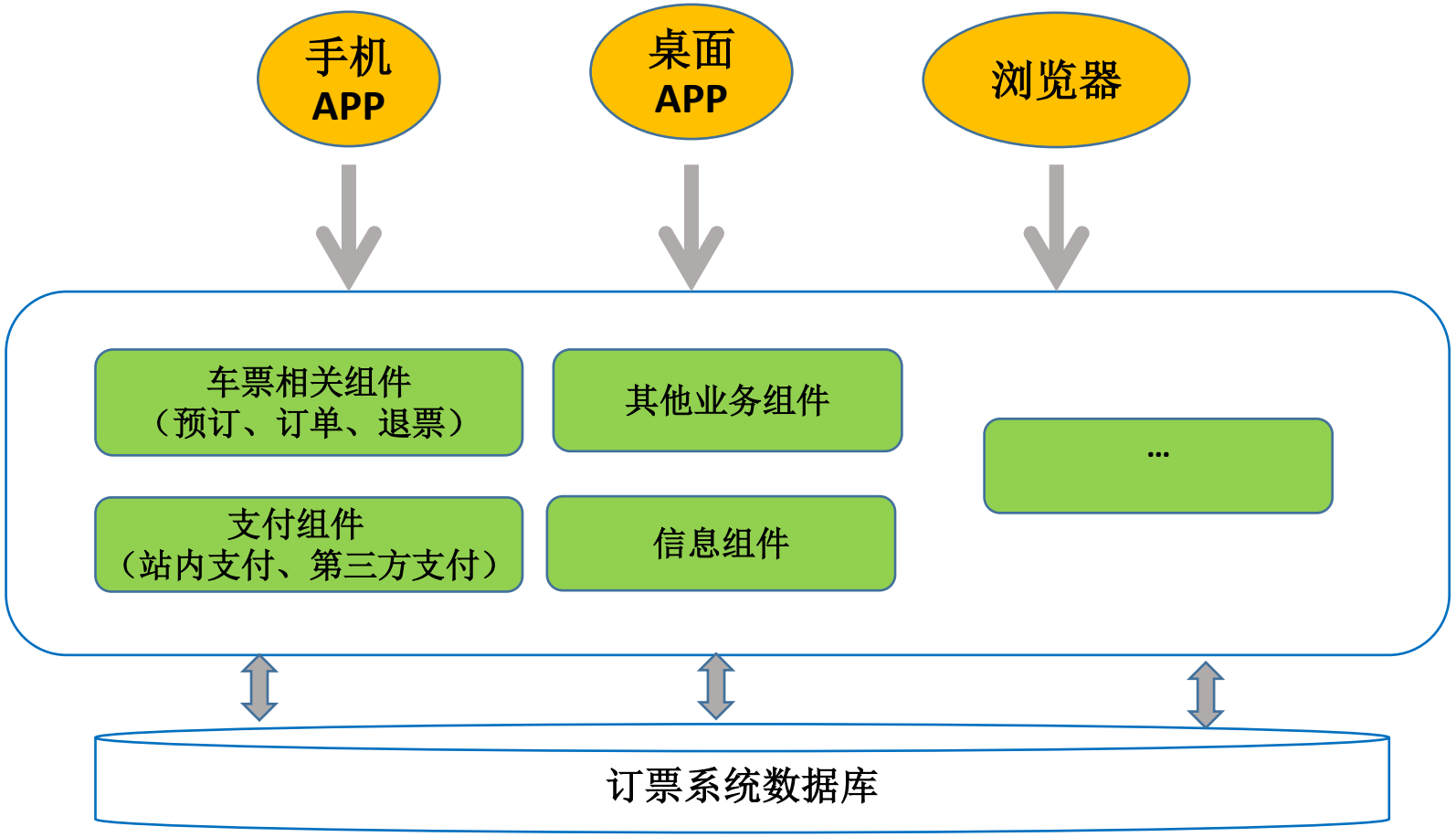


Z轴伸缩：数据划分

X轴伸缩：水平复制

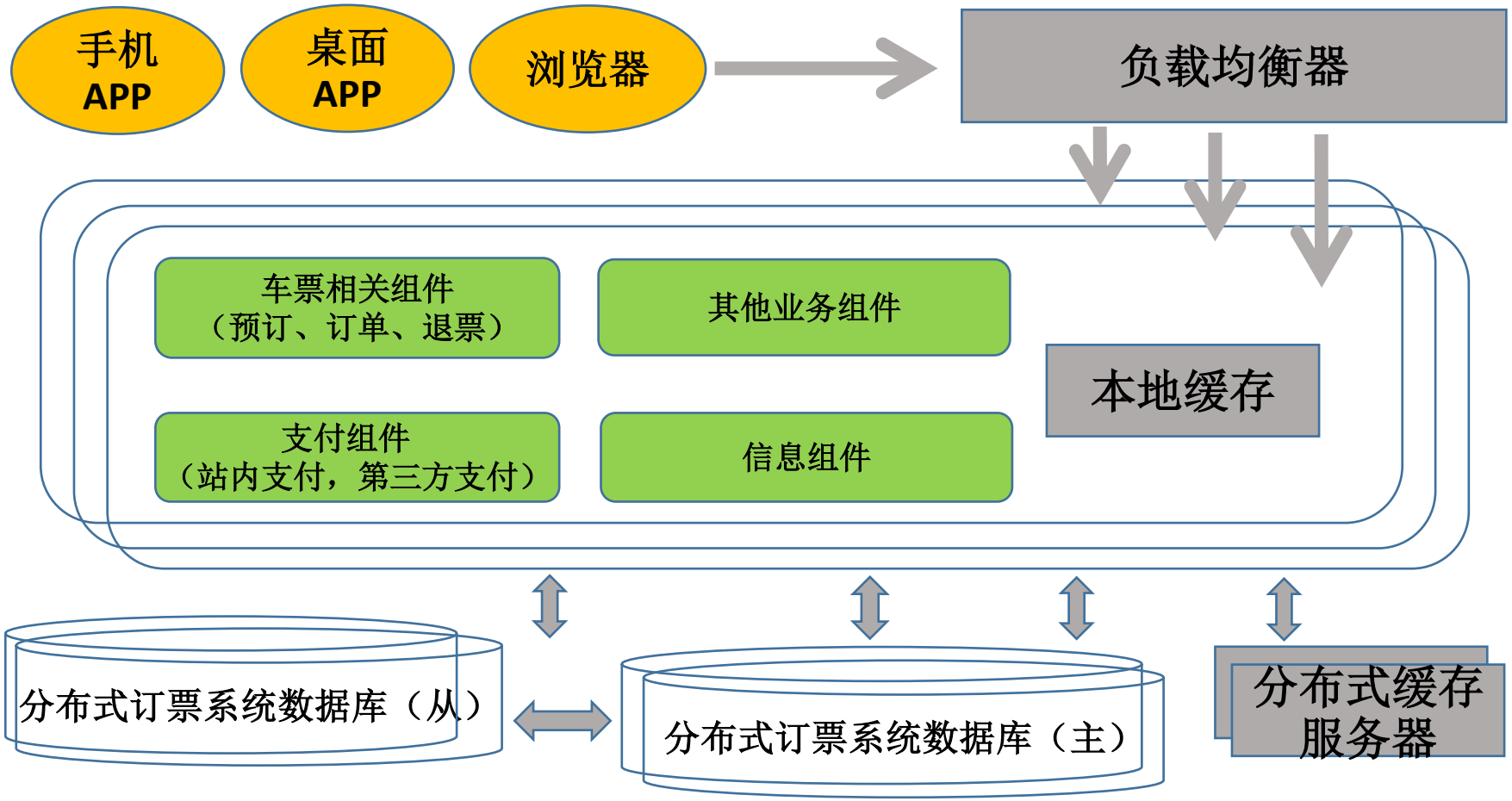
[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

单体服务器



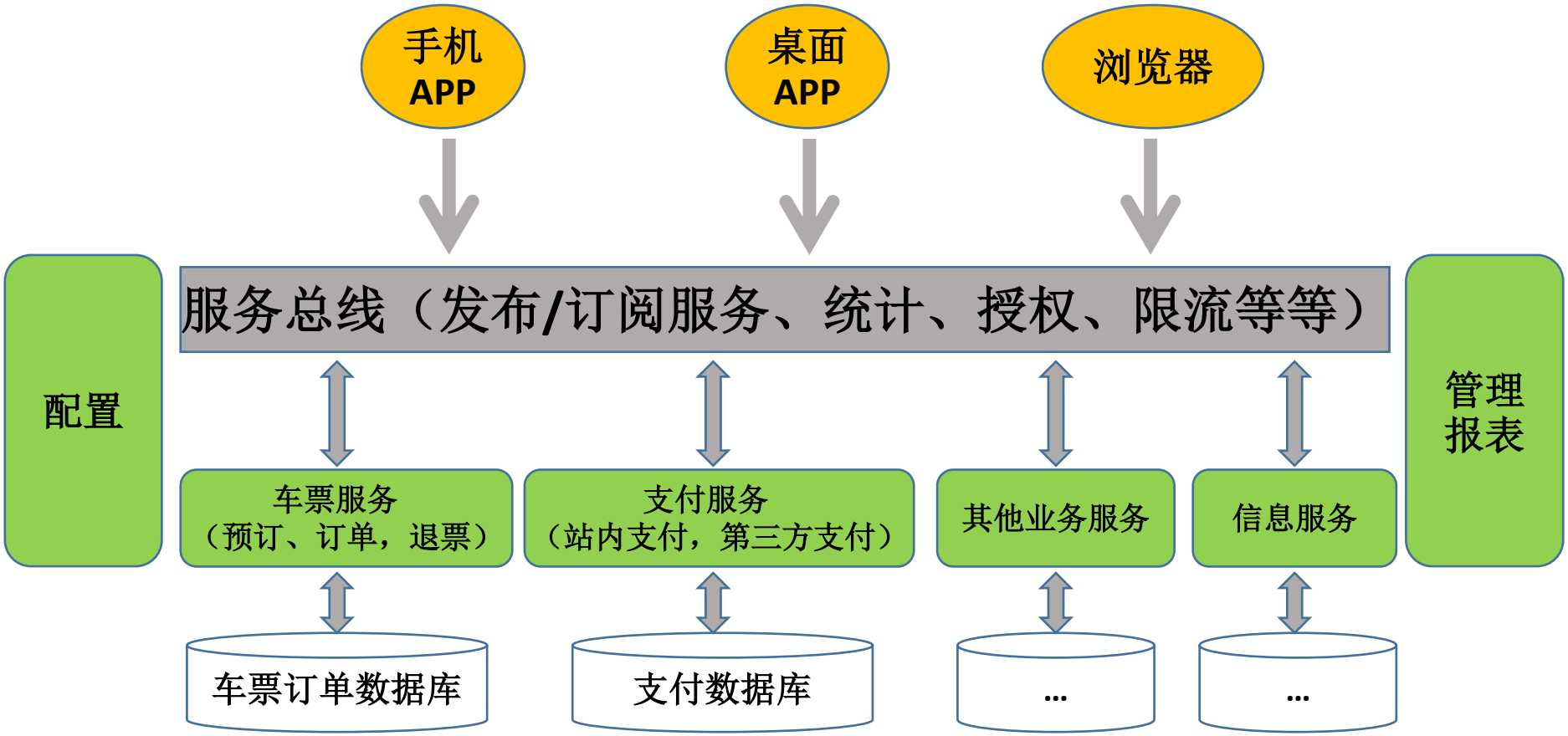
所有服务器端功能实现为一个单体应用，除数据库等基础设施之外的业务组件之间均通过本地调用来实现交互。

多服务器负载均衡



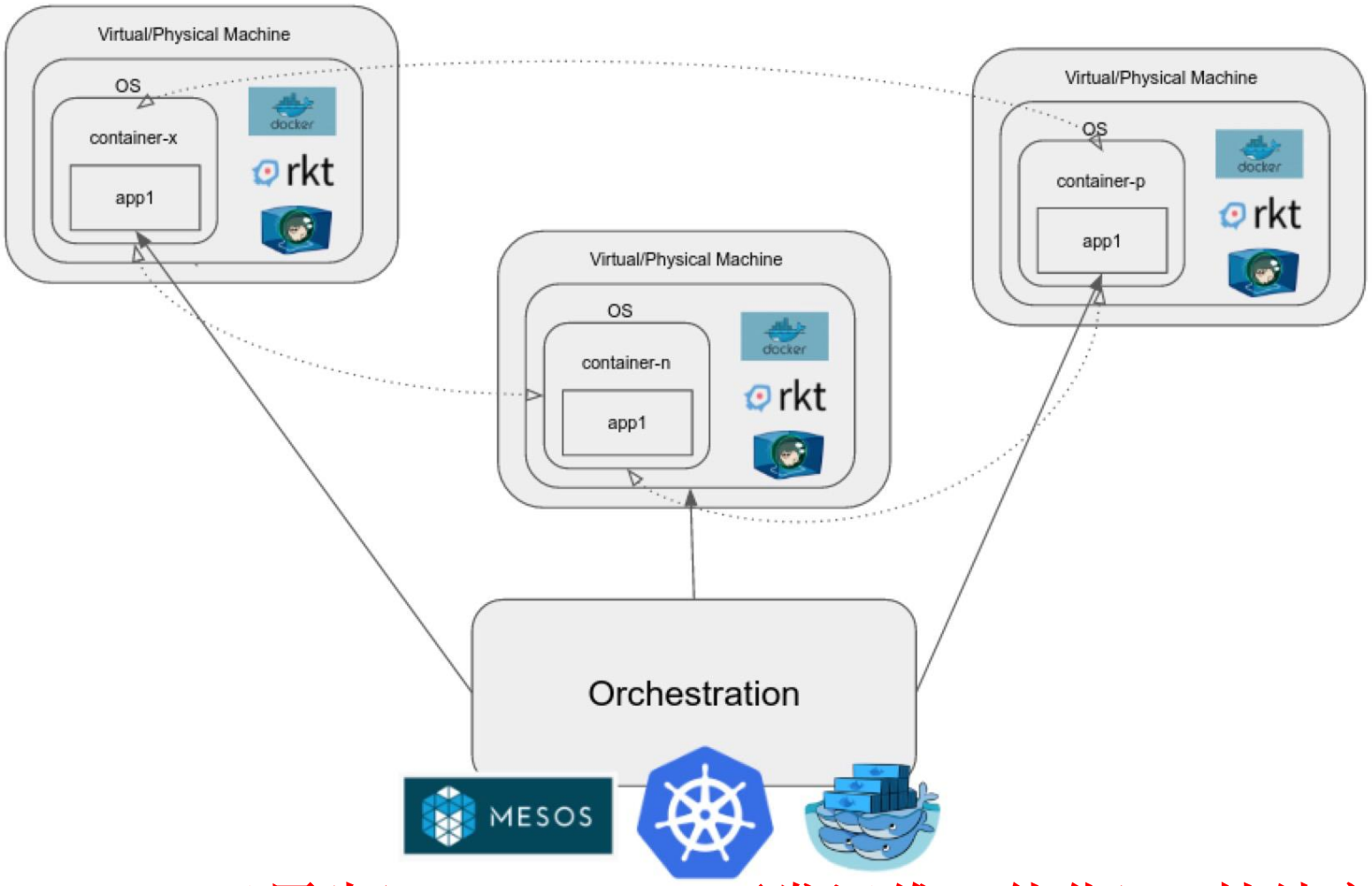
所有服务端业务功能实现为一个单体应用，部署在服务器集群上并配合分布式数据库和分布式缓存，通过负载均衡器来平衡用户请求负载，并在必要时同步不同服务器之前在线用户的Session信息

面向服务的体系结构



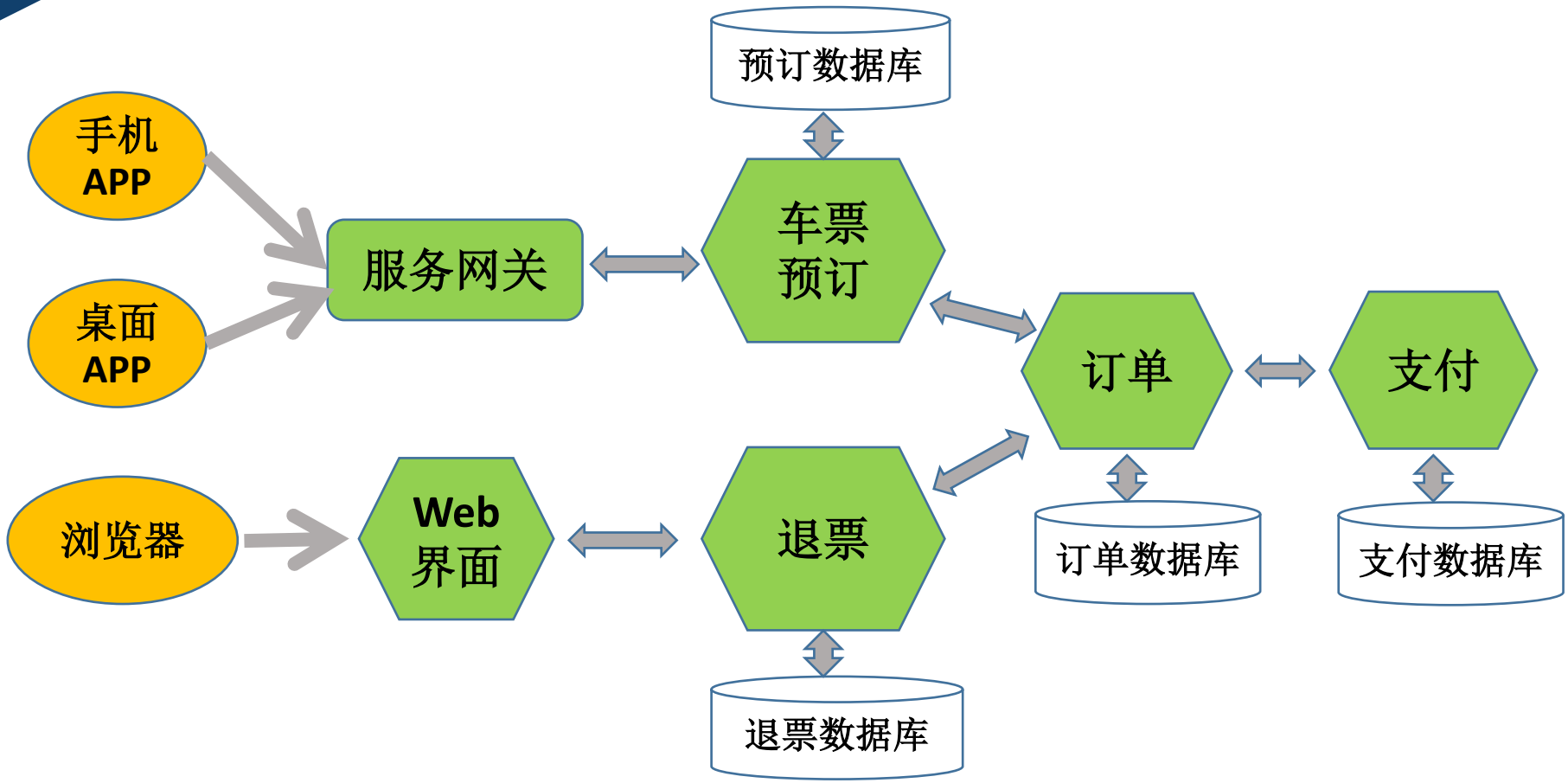
业务功能拆分到不同的应用服务中，通过服务总线整合，并通过统一的配置平台和管理报表平台管理服务，服务间交互基于服务总线做通信协议转换

云+容器



Cloud Native（云原生）= DevOps（开发运维一体化）+ 持续交付（Continuous Delivery）+ 微服务（MicroServices）+ 敏捷基础设施（Agile Infrastructure）+ 康威定律（Conways Law）+...

云+容器+微服务



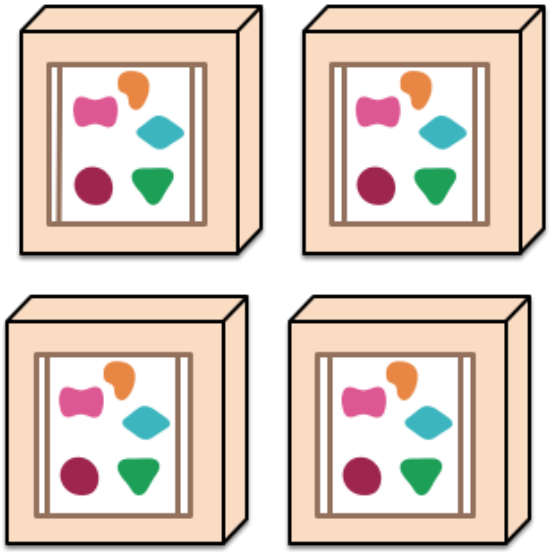
业务功能再次细分，并舍弃服务总线这种集中式管理的方式，统一由集群编排系统（如Kubernetes）管理，服务间通信大多基于轻量级的RESTful协议，对外统一由编排系统的API Gateway向外提供接口，对内由编排系统的服务注册、服务发现和负载均衡管理

微服务（MicroService）体系结构

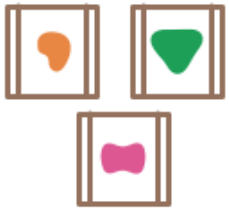
A monolithic application puts all its functionality into a single process...

巨石应用

... and scales by replicating the monolith on multiple servers

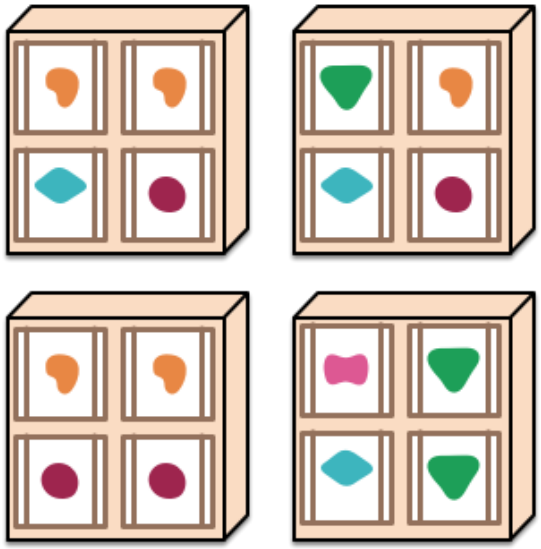


A microservices architecture puts each element of functionality into a separate service...



微服务应用

... and scales by distributing these services across servers, replicating as needed.

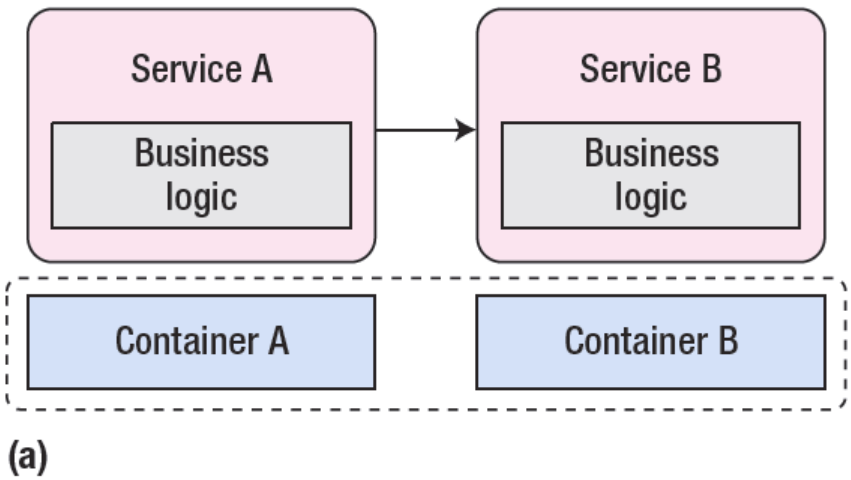


Martin Fowler

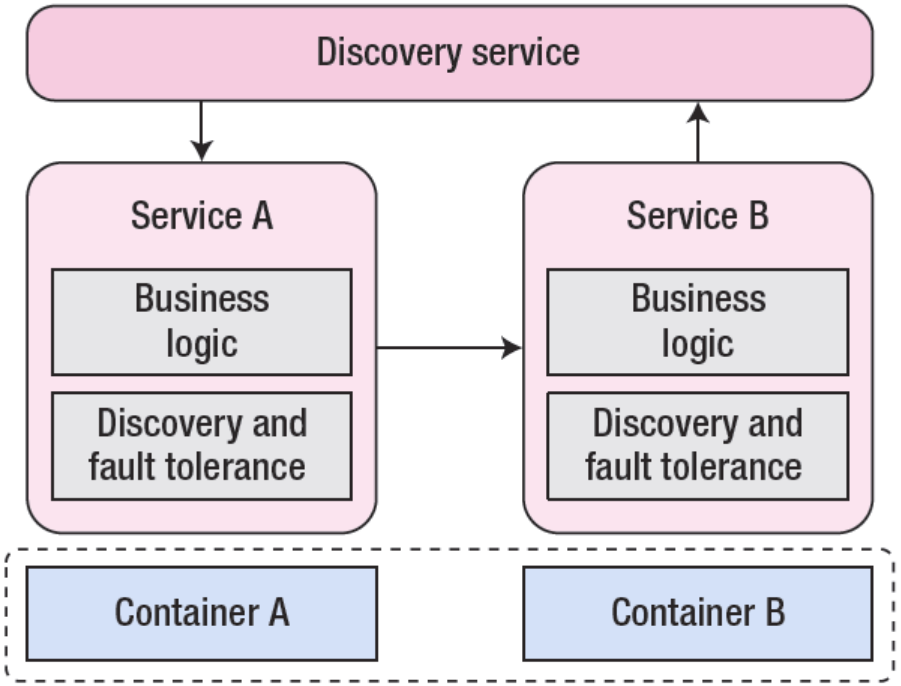
每个微服务都可以独立开发、维护、部署和伸缩，可以使用不同的语言。微服务之间通过REST等协议实现网络化调用，因此同步调用被认为是有害的。

https://www.martinfowler.com/articles/microservices.html

微服务体系结构的发展：服务发现



(a)

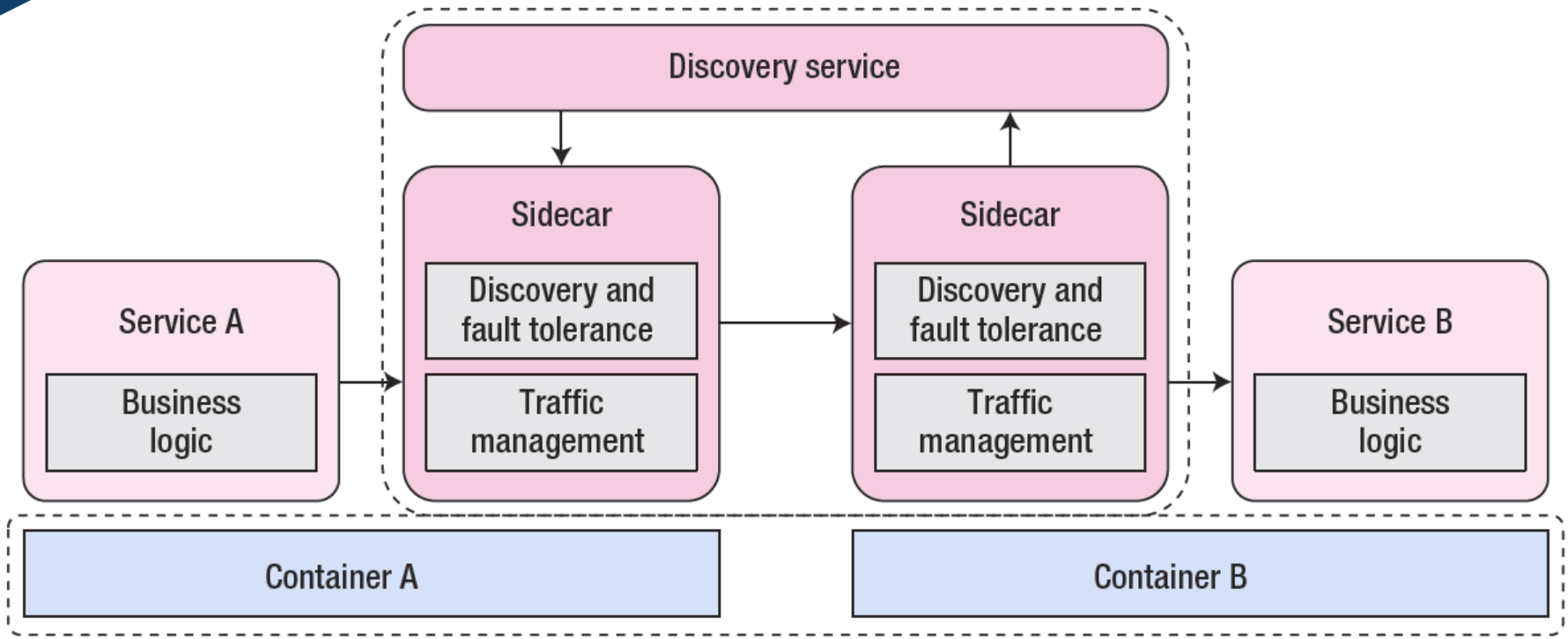


(b)

通过服务发现实现微服务之间解耦，微服务通过服务发现动态绑定所依赖的其他微服务，同时负责进行容错处理（超时处理、拒绝服务等）

Pooyan Jamshidi, et. al.. Microservices: The Journey So Far and Challenges Ahead. IEEE Software, 2018, Vol. 35, No. 3, pp. 24-35.

微服务体系结构的发展：Service Mesh

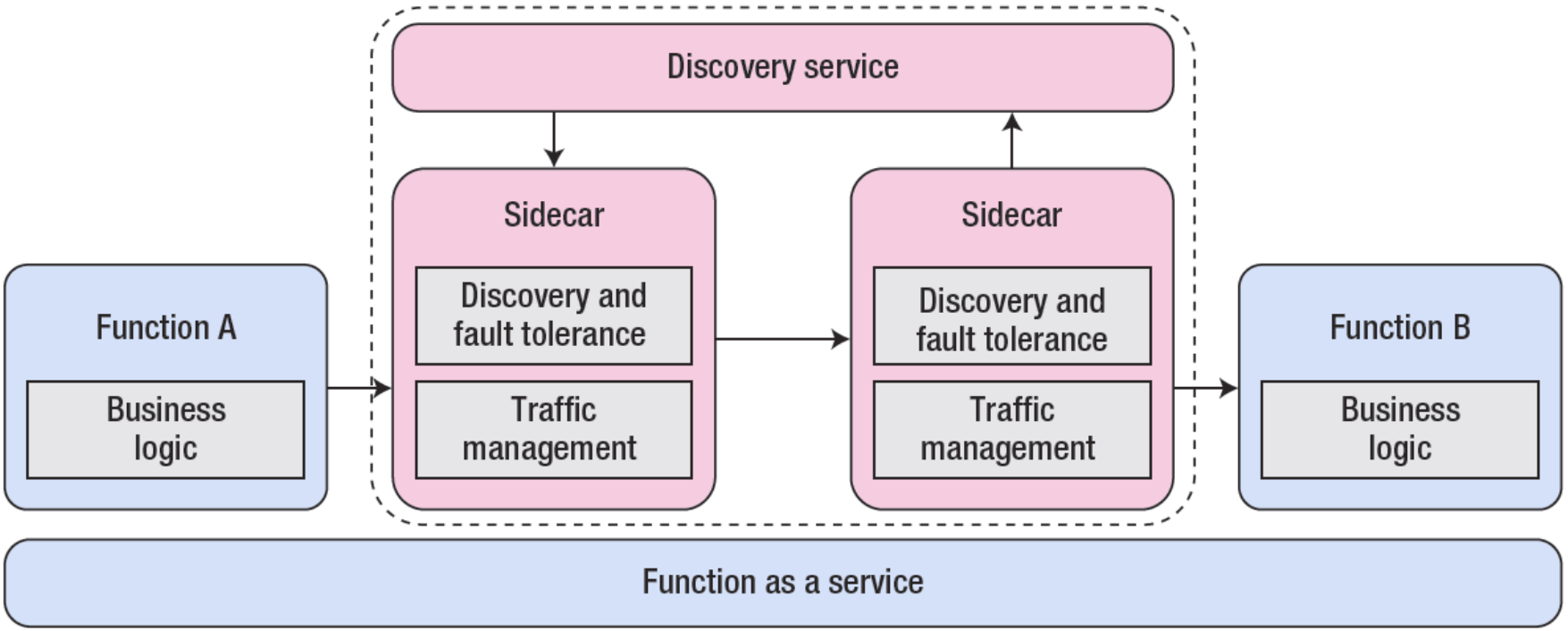


(c)

通过引入代理（Sidecar）形成服务网格，微服务专注于业务逻辑实现

Pooyan Jamshidi, et. al.. Microservices: The Journey So Far and Challenges Ahead. IEEE Software, 2018, Vol. 35, No. 3, pp. 24-35.

微服务体系结构的发展：Serverless



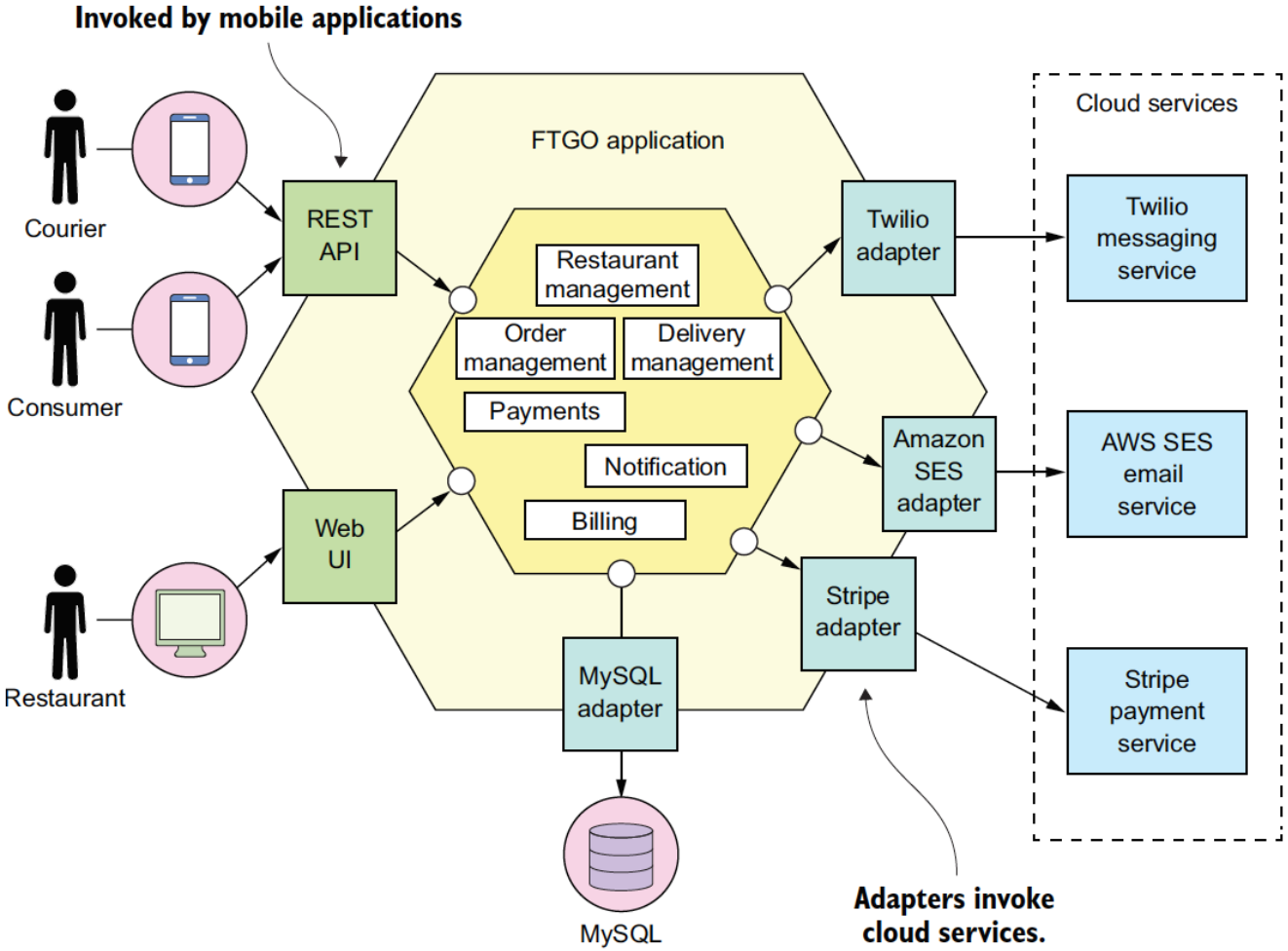
(d)

服务开发者无需关注计算资源的获取和运维，由平台来按需分配计算资源并保证服务质量，业务代码仅在调用时才激活运行，当响应结束占用资源便会释放

Pooyan Jamshidi, et. al.. Microservices: The Journey So Far and Challenges Ahead. IEEE Software, 2018, Vol. 35, No. 3, pp. 24-35.

从单体架构到微服务架构：案例分析

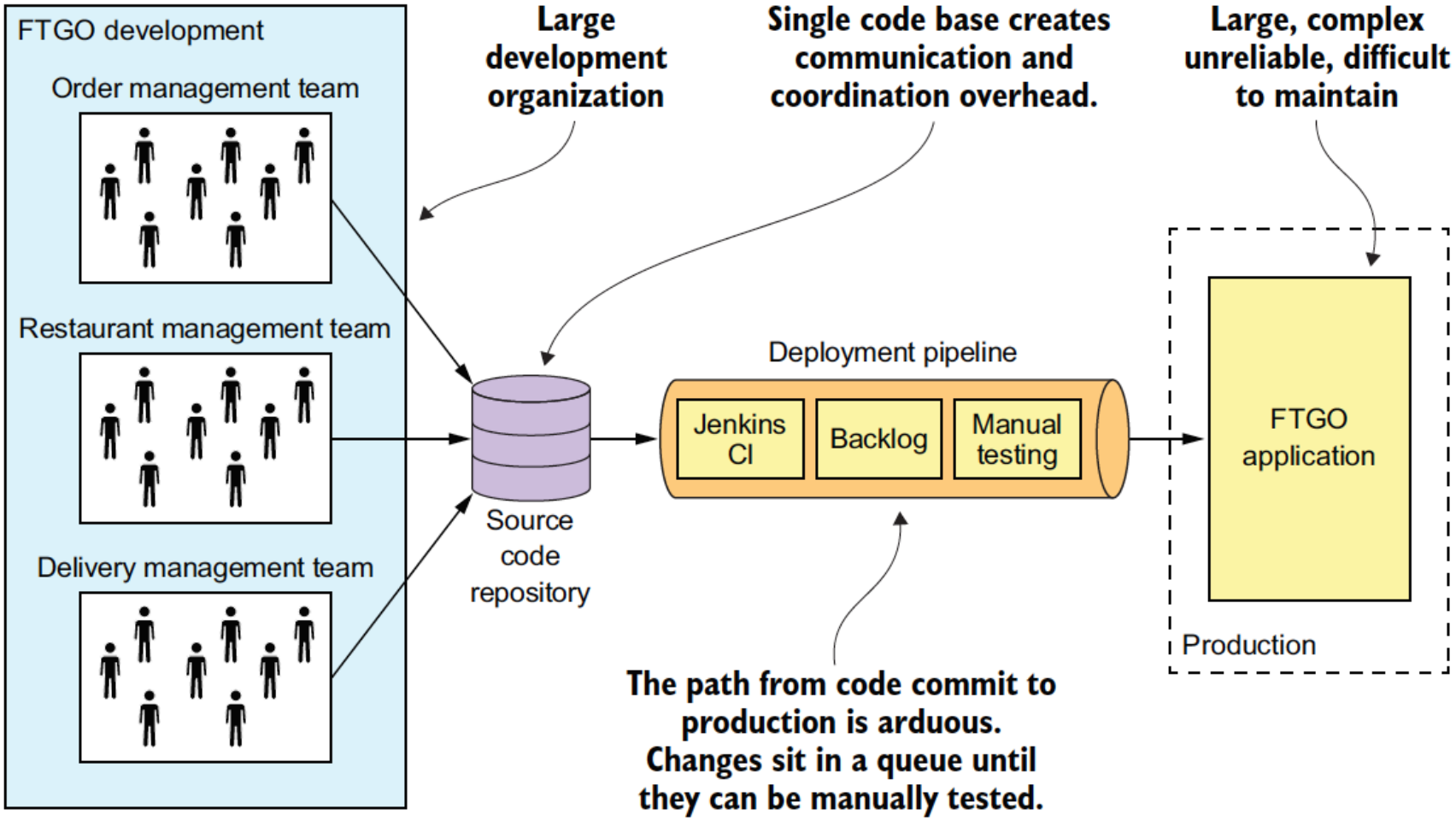
在线餐饮速递系统FTGO的单体架构



[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

从单体架构到微服务架构：案例分析

FTGO系统单体架构下的开发和发布组织



[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

从单体架构到微服务架构：案例分析

单体地狱：单体架构的优缺点

优点

1）应用开发简单
2）易于进行大规模修改
3）测试相对简单直观
4）部署简单明了
5）横向扩展比较容易

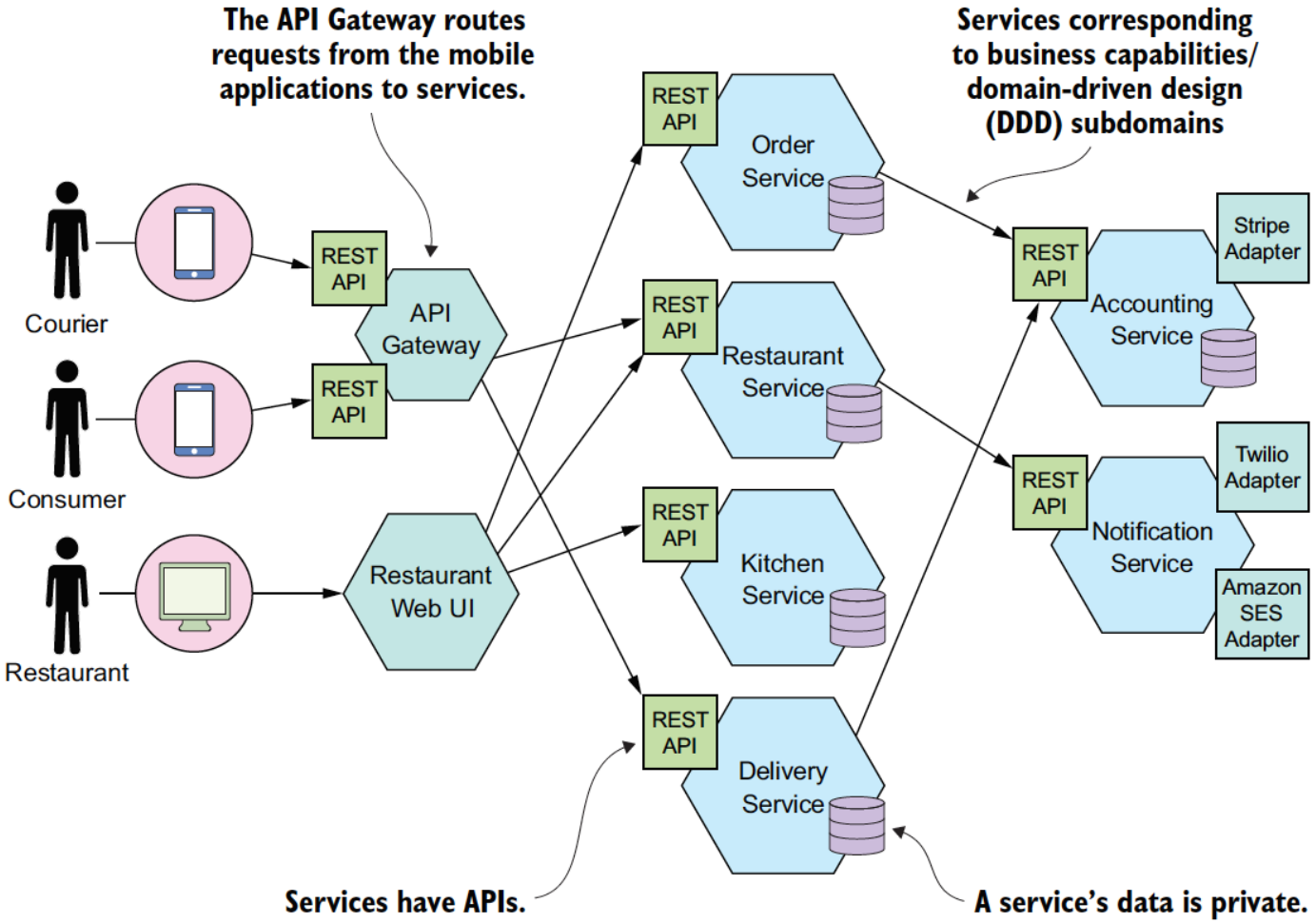
缺点

1）过度的复杂性
2）开发速度缓慢
3）从开发到部署周期长、容易出问题
4）难以扩展
5）难以保证可靠性
6）依赖过时的技术栈

[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

从单体架构到微服务架构：案例分析

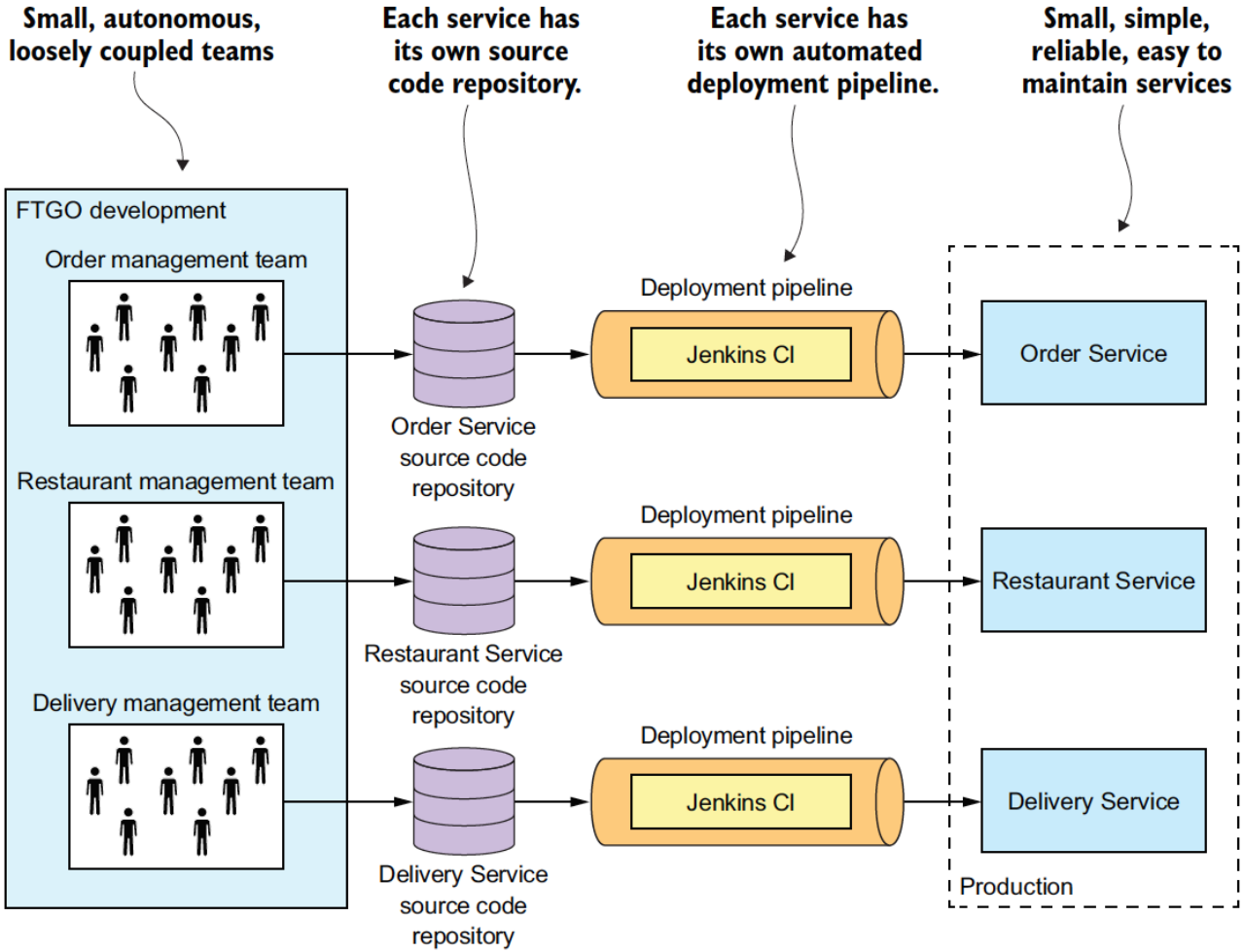
FTGO系统的微服务架构



[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

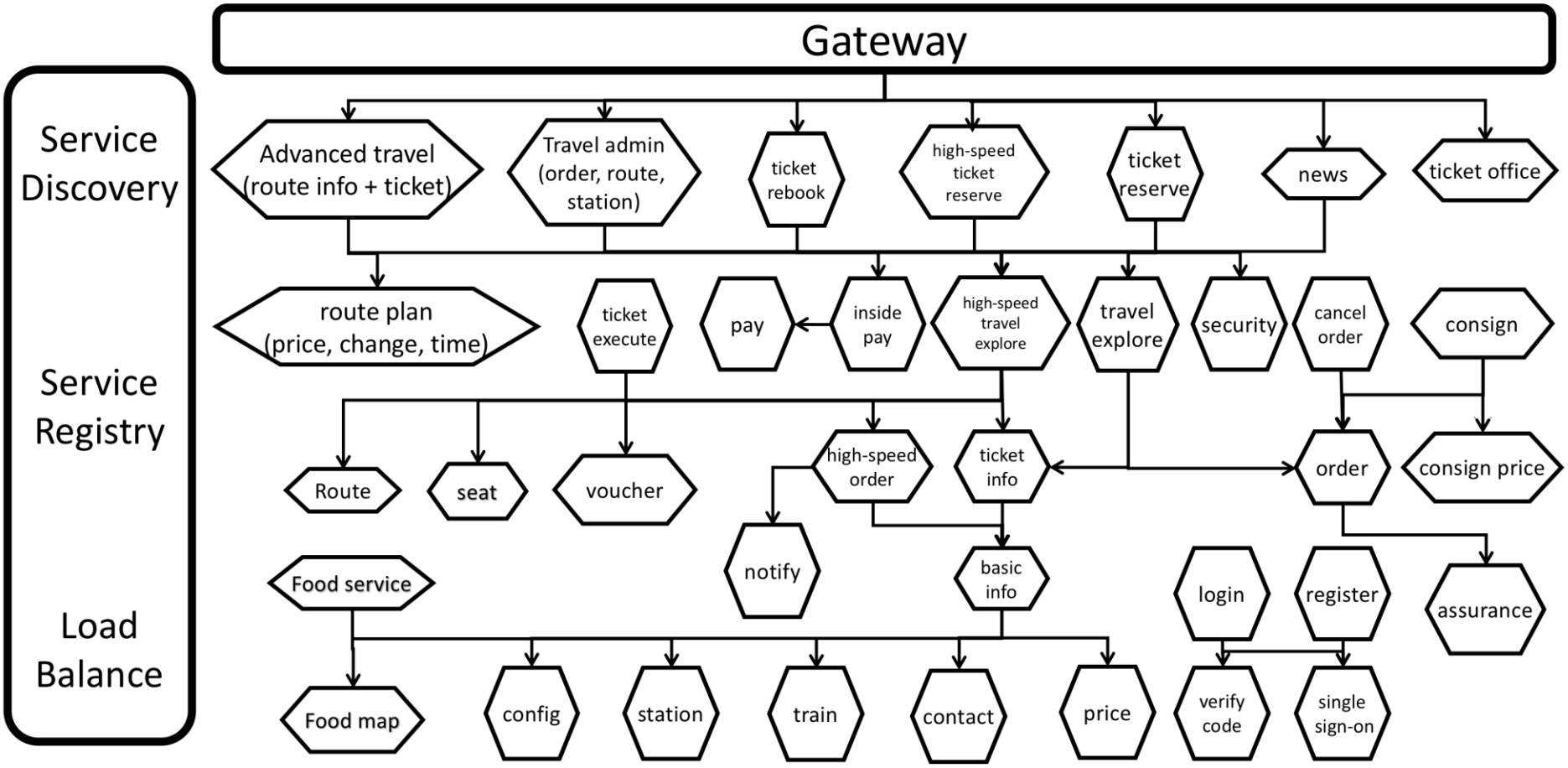
从单体架构到微服务架构：案例分析

FTGO系统微服架构下的开发和发布组织



[美] 克里斯·理查森（Chris Richardson），微服务架构设计模式（Microservices Patterns: With Examples in Java），机械工业出版社，2019.5

网络购票系统TrainTicket的微服务体系结构



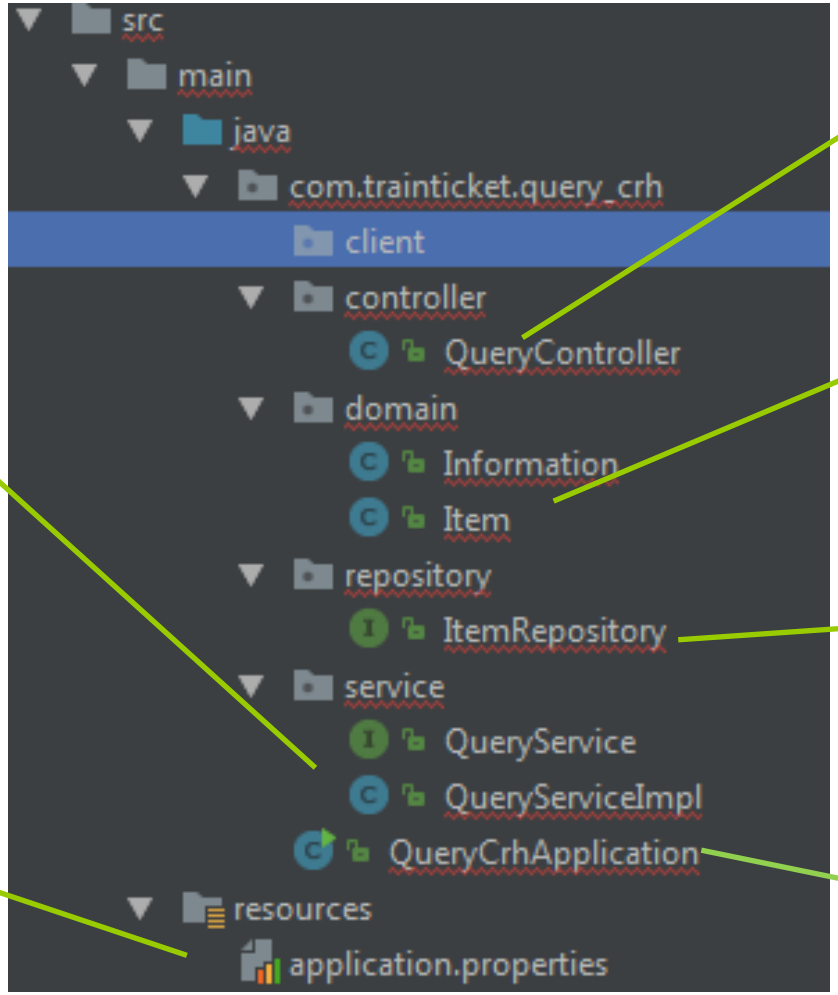
项目网址：https://github.com/FudanSELab/train-ticket

微服务应用开发技术

- 应用开发技术
  - ✓ Spring Boot
  - ✓ Spring Cloud
- 容器技术
  - ✓ Docker
- 集群技术
  - ✓ Docker Swarm
  - ✓ Kubernetes（K8S）

基于Spring Boot开发单个微服务

高铁票查询微服务： **querycrh-service**



根据URL来执行对应方法

具体功能接口及实现

数据实体定义（Mongodb DB）

定义数据库查询等操作

端口、数据库等配置

程序入口，类似main函数

微服务实现代码

注解@RestContrller

自动注入

根据URL来执行对应方法

请求的具体内容

```
@RestController
public class QueryController {

    @Autowired
    private QueryService queryService;

    @RequestMapping(path = "/querycrh",method = RequestMethod.POST)
    public List<Item> query(@RequestBody Information info){

        return queryService.query(info);
    }

}
```

声明Mongodb的结构

ID字段

车次号字段

```
@Document(collection = "item")
public class Item {

    @Id
    private String id;

    @Valid
    @NotNull
    private String trainNumber;
```

按照规定的方法名格式声明查询操作

```
@Repository
public interface ItemRepository extends CrudRepository<Item, String> {

    List<Item> findByStartingAndDestination(String staring,String destination);
}
```

微服务构建与容器部署

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.services</groupId>
    <artifactId>ts-service</artifactId>
    <version>0.1.0</version>
    <packaging>pom</packaging>
    <name>ts-service-cluster</name>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.3.RELEASE</version>
    </parent>

    <modules>
        <module>ts-login-service</module>
        <module>ts-register-service</module>
        <module>ts-sso-service</module>
        <module>ts-verification-code-service</module>
        <module>ts-contacts-service</module>
        <module>ts-order-service</module>
        <module>ts-order-other-service</module>
        <module>ts-config-service</module>
        <module>ts-station-service</module>
        <module>ts-train-service</module>
        <module>ts-travel-service</module>
        <module>ts-travel2-service</module>
        <module>ts-preserve-service</module>
        <module>ts-preserve-other-service</module>
        <module>ts-basic-service</module>
```

```
FROM java:8-jre
MAINTAINER jichao <13302010019@fudan.edu.cn>

ADD ./target/ts-order-service-1.0.jar /app/
CMD ["java", "-Xmx200m", "-jar", "/app/ts-order-service-1.0.jar"]

EXPOSE 12031
```

微服务集群配置：Docker Swarm

```yaml
version: '3'
services:

  rabbitmq:
    image: rabbitmq:management
    ports:
      - 5672:5672
      - 15672:15672
    networks:
      - my-network

  zipkin:
    image: openzipkin/zipkin
    ports:
      - 9411:9411
    networks:
      - my-network

  redis:
    image: redis
    ports:
      - 6379:6379
    networks:
      - my-network


  ts-ui-dashboard:
    build: ts-ui-dashboard
    image: ts/ts-ui-dashboard
    restart: always
    ports:
      - 80:8080
    networks:
      - my-network

  ts-login-service:
    build: ts-login-service
    image: ts/ts-login-service
    restart: always
    ports:
      - 12342:12342
    networks:
      - my-network
```

```
---

## local runtime environment

build:

mvn build:

mvn -Dmaven.test.skip=true clean package

docker-compose -f docker-compose.yml build

docker build:

docker-compose build
(docker-compose -f docker-compose.yml build)

docker-compose up -d

docker-compose down


start the ticket microservice applciation (single node):

docker-compose -f docker-compose.yml up -d

docker-compose down

docker-compose logs -f
```

```
##  clustering runtime environment(docker swarm):

build:

mvn clean package

docker-compose build

docker-compose up

docker swarm init --advertise-addr 10.141.211.161

docker swarm join-token manager

docker swarm join-token worker


app tag:

docker tag ts/ts-ui-dashboard 10.141.212.25:5555/cluster-ts-ui-dashboard


app local registry:

docker push 10.141.212.25:5555/cluster-ts-ui-dashboard


deploy app (docker swarm):

docker stack deploy --compose-file=docker-compose-swarm.yml my-compose-swarm
```

微服务集群配置：Kubernetes

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: 10.141.212.25:5555/cluster-ts-redis
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 6379

---

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: ts-account-mongo
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: ts-account-mongo
    spec:
      containers:
      - name: ts-account-mongo
        image: 10.141.212.25:5555/cluster-ts-mongo
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 27017
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
    - port: 6379
  selector:
    app: redis

---

apiVersion: v1
kind: Service
metadata:
  name: ts-account-mongo
spec:
  ports:
    - port: 27017
  selector:
    app: ts-account-mongo

---

apiVersion: v1
kind: Service
metadata:
  name: ts-route-mongo
spec:
  ports:
    - port: 27017
  selector:
    app: ts-route-mongo

---
```

TrainTicket系统微服务调用链示意图

阅读建议

- 《软件工程》第5、6、17、18章

快速阅读后整理问题
在QQ群中提出并讨论

CS2001
软件工程

End

13. 软件设计
—软件体系结构