## Devops第3章

注：（）内容为相关说明，供理解，不用背

1.什么是一个好的软件架构

（在现实生活中，没有一个架构是完全可以通用的，更没有一个最好的架构可以适用于任何使用场景。）

软件架构是对用户的需求，公司对业务的需求，公司未来发展的需求，以及成本条件的折中，在这些限制条件下，我们才能探求一个好的架构设计，同时，一个好的架构设计也不是一成不变的，随着时间的发展，公司的业务方向会发生变化，需求会发生变化，相应的成本预算也会发生变化，所以架构也自然随着这些先决条件的变化而不断演化。

一个好的架构的首要目标是能够支持这种演化，让演化能够按照预期的方式进行扩展，而不是整个架构级别的重构。

3.常见的软件架构的视图有哪几种？描述每种视图的作用和涉众

4+1模型：逻辑视图，开发视图，进程视图，物理视图+场景

1）逻辑视图：逻辑视图主要是为了支持功能性需求，即系统需要给用户提供哪些服务。涉众：最终用户

（在这个视图中，会利用面向对象的方式（抽象，封装，继承）把系统分解成一系列领域对象和类）

（一般用类图描述/数据驱动则是E-R图）

2）进程视图：进程视图主要从性能，可用性等一些非功能需求的角度，从多个抽象层次去描述系统。涉众：系统设计和继承人员

（着重强调系统的并发和分布，系统完整性，容错性，以及把逻辑视图中的抽象元素放到进程视图中时，哪个控制进程去运行一个 操作）

（组件：任务，进程）

3）开发视图：开发视图主要关注软件开发实现过程中的真正的软件模块以及如何组织，同时也用来做任务分配，成本评估，制定开发计划，跟踪开发进度，考虑软件的重用性和可移植性等，是制定产品线的一个基础。涉众：开发者和技术经理

（组件：模块，子系统）

4）物理视图（部署视图）：物理视图主要考虑把一些非功能性的需求，比如可用性，可靠性（容错性），性能（吞吐量）以及横向扩展性。涉众：系统设计者

（组件：节点）（C、F、K：CPU、内存、磁盘）

5）场景视图：把以上视图的设计元素结合在一起，利用对象场景图和对象交互图来描述系统中一些重要用例的场景，对最重要的一些需求进行抽象。涉众：最终用户和开发者，主要用来

1. 作为驱动的工具，用在架构过程中发现架构元素（场景驱动的架构设计）
2. 作为验证的角色，用在架构设计完成后，在纸上进行架构原型的测试和验证。

4.相比传统软件架构，互联网软件系统有什么特点

1. 用户量大，并发高
2. 业务变化快，强调敏捷
3. 系统多（可能有几个，甚至成百上千个不同功能系统同时运行）
4. 对系统可用性要求更高，追求24\*7零宕机时间

5.阐述单体架构的优缺点

优点：（后面章节找的）

易于测试；

IDE友好；

便于共享

为人所熟知

容易部署

缺点：

多个团队协作开发效率低，需要额外的协调工作

系统的可维护性非常差，尤其是对大型的应用

任何小的改动部署都会影响整个系统重新部署

系统复杂了以后，很难做扩展

遇到性能问题，只能整体加机器，难以拆分

6.阐述事件驱动架构中Mediator模式和Broker模式的区别

Mediator模式适合需要很多步骤来处理一个事件，同时又希望这些步骤的组合在未来容易更改的场景，当一个事件通过消息队列发给Event Mediator的时候，Event Mediator会根据预配置的步骤分成多个子事件，通过Event Channel分发给具体的Event Processor。步骤控制放在独立的一个Event Mediator，Event Processor只处理单一的业务逻辑。

Broker模式和Mediator模式最大的区别就是没有Event Mediator，它的消息流直接通过消息通道分发到下游的消息处理器。Event Processor除了做好自己的业务逻辑，还要明白事件的来源和去向。

当业务流程比较固定的时候，Broker模式更易于维护，当业务流程变化比较大的时候，Mediator模式更易于修改。

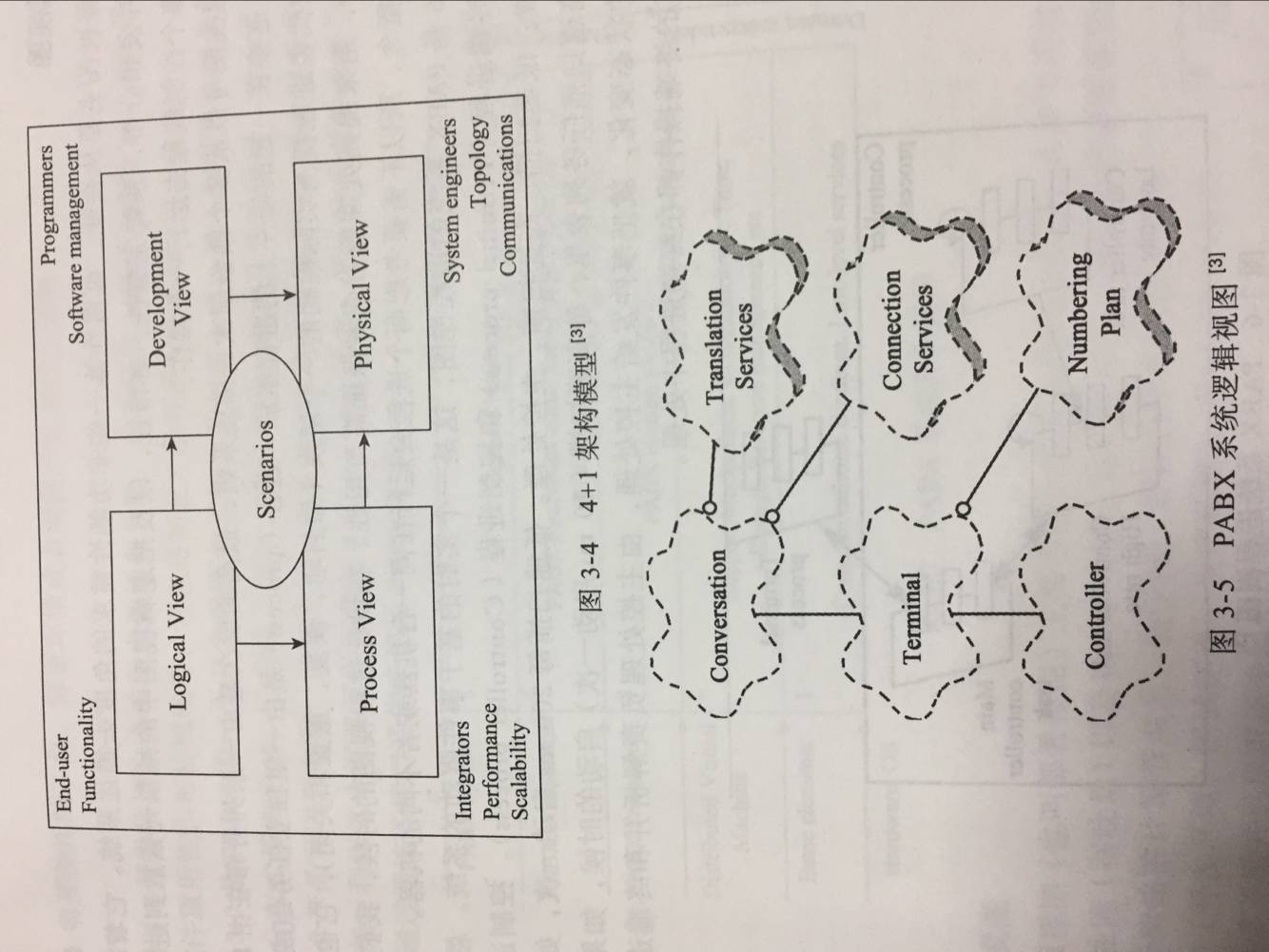
7.从运维角度，阐述微服务架构的优点

1. 每个服务可以被独立的开发、部署，其他服务几乎不受影响，也不需要跟其他服务的开发人员进行额外的协作和等待
2. 局部修改，局部更新。当运维对单体应用进行修改时，可能要先把应用停止运行后才去修改，而微服务只需逐步修改和更新即可；
3. 故障隔离，非全局。单体应用中只要一个模块有故障，应用就无法正常运行。但微服务的故障隔离性、业务可持续性都非常高；
4. 资源利用率高。单体应用的资源利用率低，而使用微服务，可以按需分配资源，资源利用率会非常高。

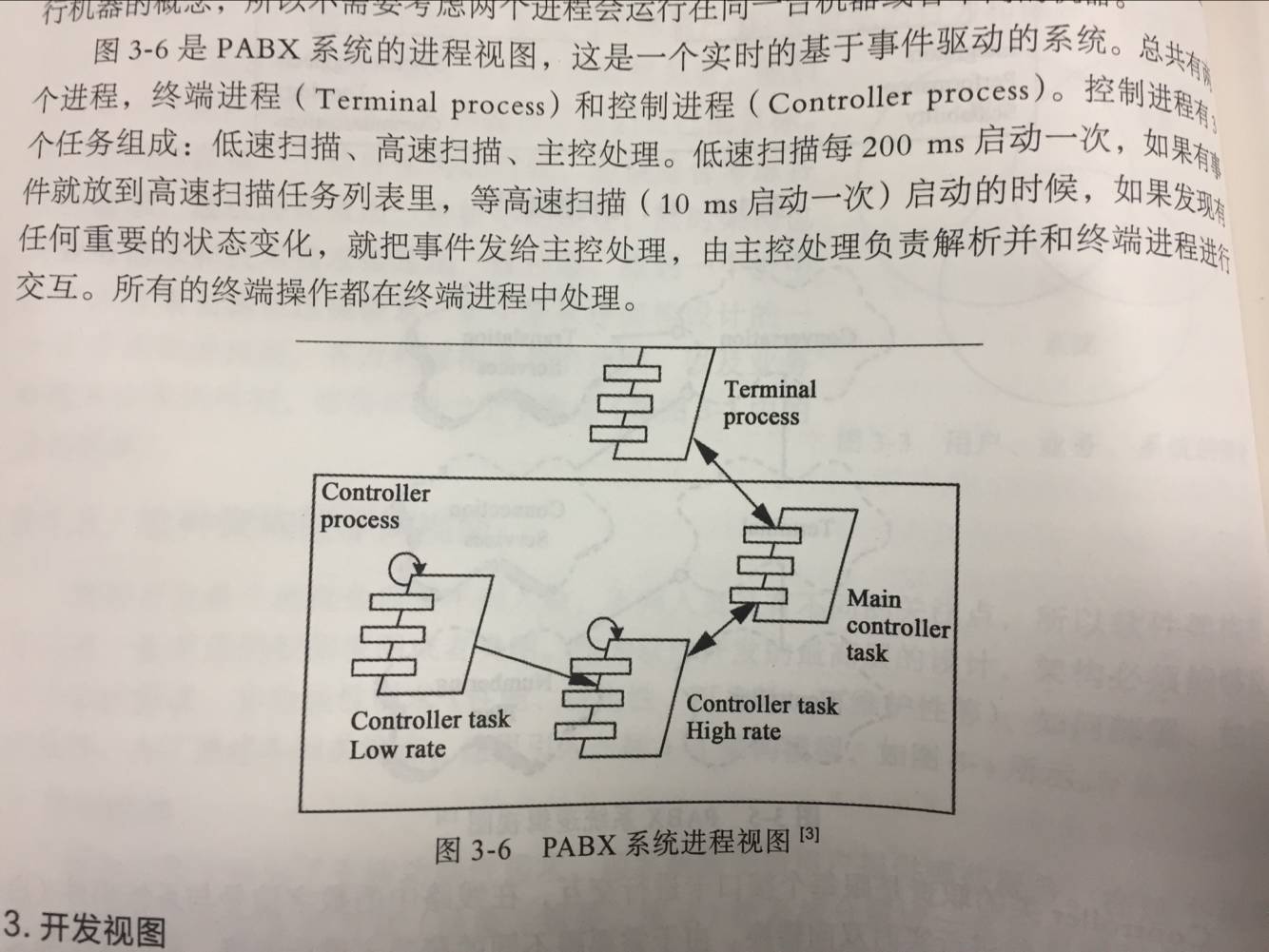
8，9，10-以4+1架构模型描述和演进教学支持系统

略-附上书上4+1模型示例：

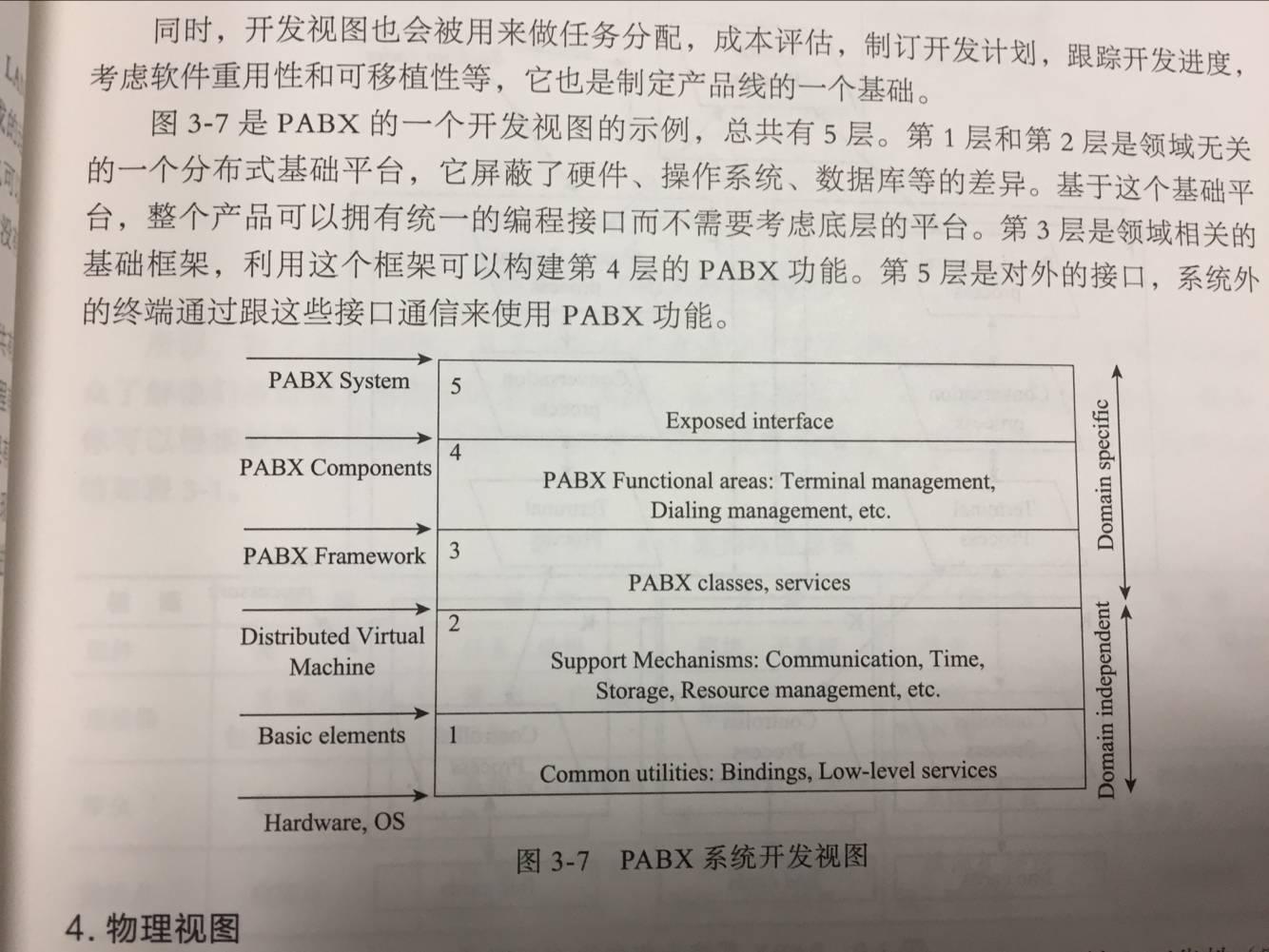
1. 逻辑视图



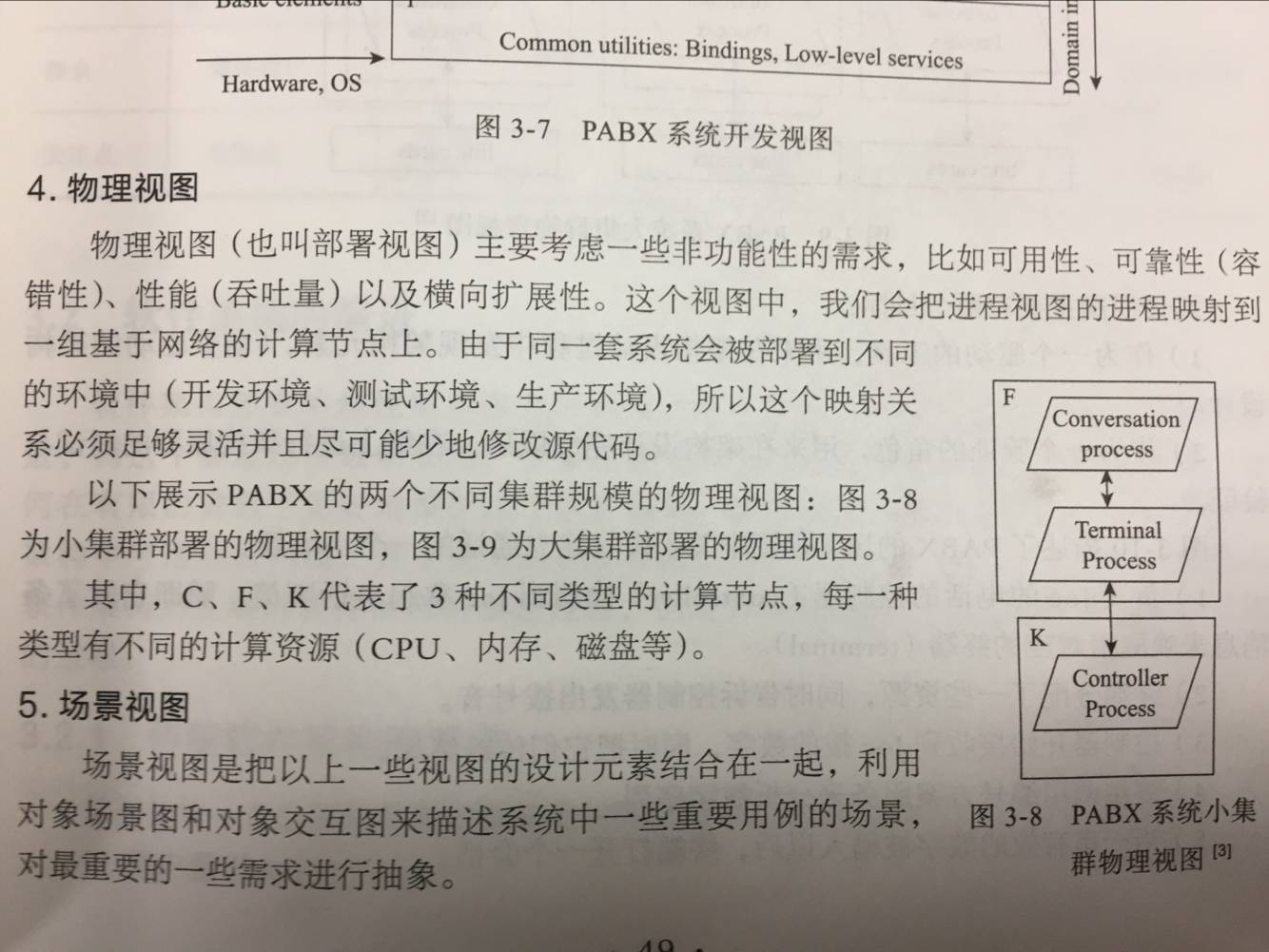
1. 进程视图



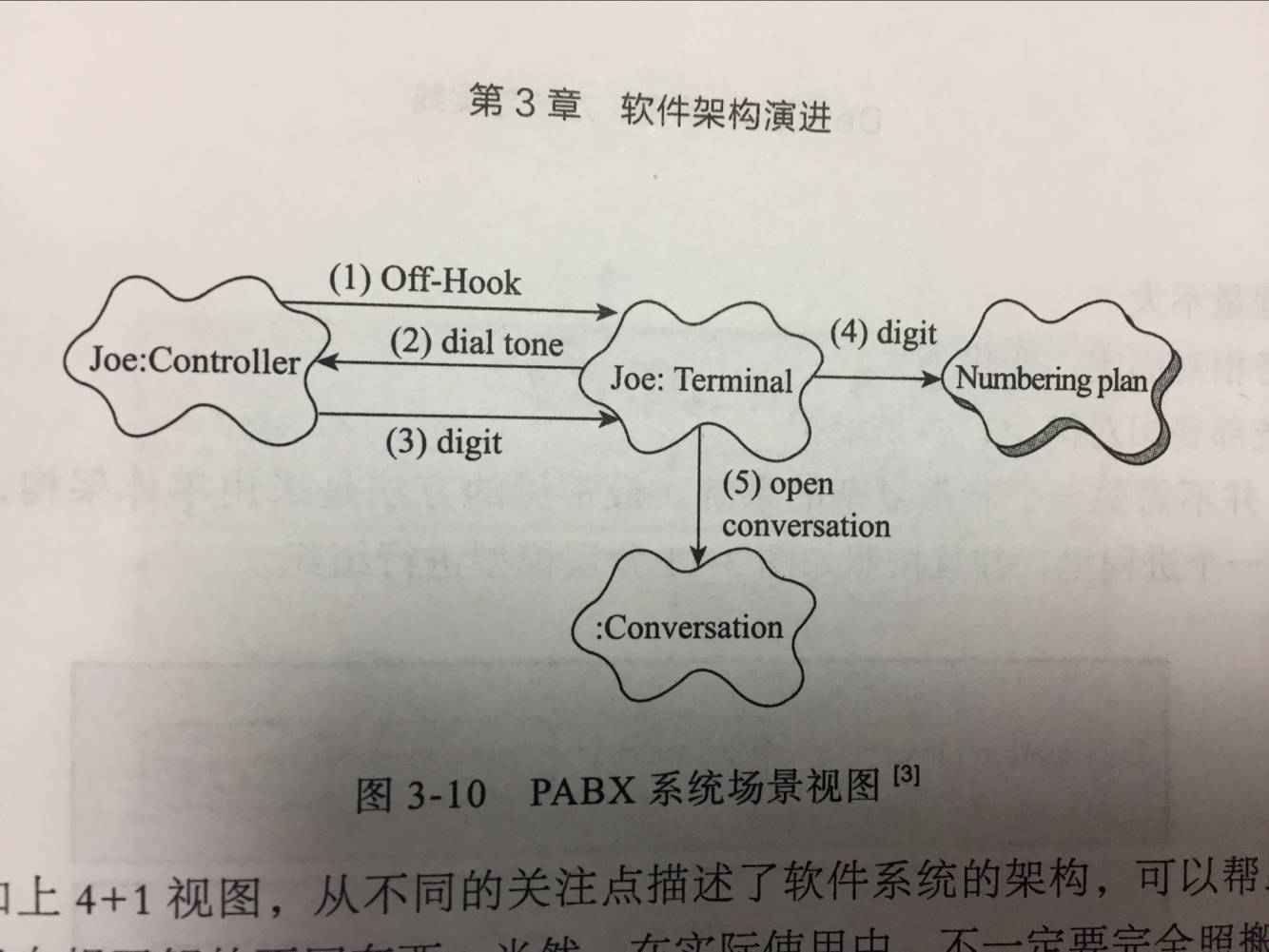
1. 开发视图



1. 物理视图



1. 场景视图



## Devops第7章

1.Linux namespace 有几种，分别是什么？

6种

1. UTS namespace：提供主机名与域名的隔离
2. IPC namespace：提供信号量、消息队列和共享内存的隔离
3. PID namespace：对进程PID进行重新编号，提供对进程编号的隔离
4. network namespace：提供对网络资源的隔离，包括网络设备，IPv4和IPv6协议栈，路由表，防火墙等
5. mount namespace：通过隔离文件系统挂载点对隔离文件系统提供支持
6. user namespace：隔离了安全相关的标识符和属性，包括用户ID，用户组ID，root目录，key和特殊权限

2.Dockerfile的CMD与Entrypoint有什么异同？

同：

1. 都是指定容器启动的命令和相关参数，都可以指定shell或exec函数调用的方式执行命令；
2. 每个 Dockerfile 只能有一条 CMD/ENTRYPOINT命令。如果指定了多条命令，只有最后一条会被执行。

异：

1. CMD指令指定的容器启动时的命令可以被docker run指定的命令覆盖，ENTRYPOINT在运行时也可以替代，但更繁琐，需要通过docker run 的参数 --entrypoint来指定
2. CMD指令可以作为ENTRYPOINT指令的默认参数，而且可以被docker run指定的参数覆盖；

3.编写一个运行Java应用的Dockfile

略

4.Docker网络模式有几种，分别是什么

4种

1. host 模式：和宿主机共用一个network namespace
2. container模式：和已存在的容器共享一个network namespace
3. none模式：容器拥有自己的network namespace，但并进行任何的网络配置
4. bridge模式：默认的网络设置，为每一个容器分配network namespace ，进行网络设置，并将一个主机上的Docker容器连接到一个虚拟网桥上。

5.Doker Storage Driver 有几种？分别是什么？使用场景有哪些

6种

1. AUFS：被使用在宿主机fs格式为：ext4 或 xfs 上
2. Overlay/Overlay2：被使用在宿主机fs格式为：ext4 或 xfs 上
3. Btrfs：被使用在宿主机fs格式为：btrfs 上
4. Device mapper：被使用在宿主机fs格式为：direct-lvm 上
5. VFS：被使用在宿主机fs格式为：debugging 上
6. ZFS：被使用在宿主机fs格式为：zfs 上

6.Doker编排引擎有几种，分别是什么？

4种

1. Docker Swarm
2. Kubernetes
3. Marathon
4. Nomad

补：

1.Dockerfile、Docker 镜像和Docker容器的关系

Dockerfile是软件的原材料，Docker 镜像是 软件的交付品，Docker容器是软件的运行态，从应用软件的角度看，分别代表软件的三个阶段，Dockerfile面向开发，Docker 镜像成为交付标准，Docker容器则涉及到部署和运维，三者缺一不可。

简单来说，Dockerfile构建出Docker 镜像，通过Docker 镜像运行Docker容器。

2.Dockerfile常用指令

FROM ：指定基础镜像

RUN：执行命令行（shell/exec格式）

COPY：复制文件/目录

ADD：增加文件/目录+自动解压

CMD：指定默认的容器主进程的启动命令的

ENTRYPOINT: 同CMD一样，指定容器启动程序和参数

EXPOSE：声明运行时容器提供服务端口

3.CNM - Container Network Model - 容器网络模型 ： 一套容器网络接口的标准