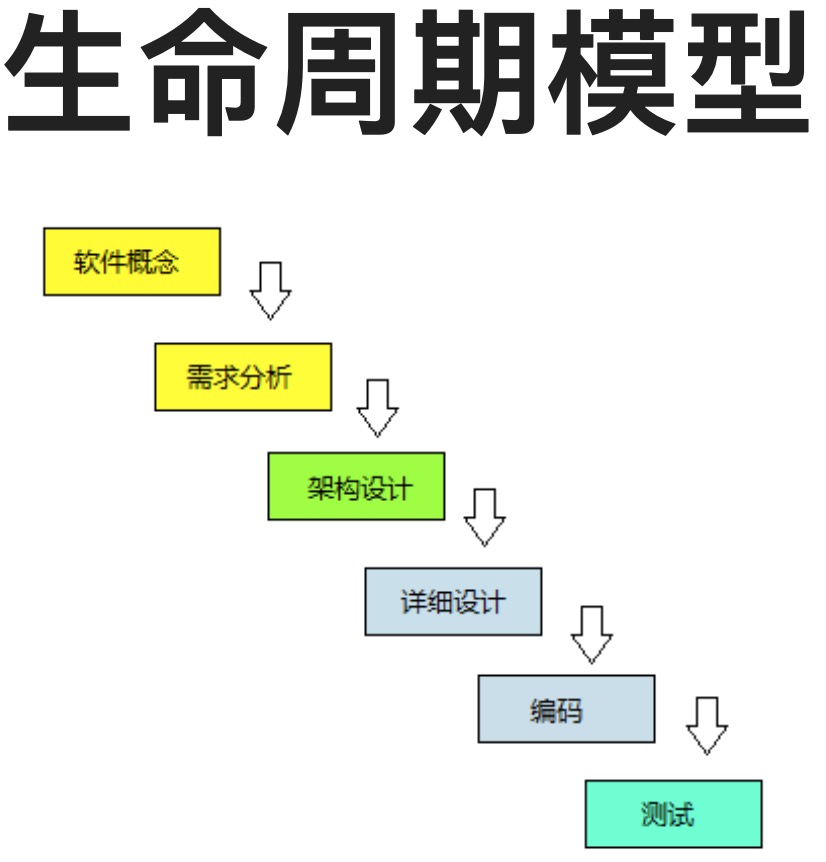
# 软件体系结构-期末复习

## Chapter1

1. 生命周期模型：软件概念 -> 需求分析 -> 架构设计 -> 详细设计 -> 编码 -> 测试



## Chapter2

1. 体系结构视图分类：
   * 模块（Module）视图，刻画模块（实现单元）的关系
   * 构件连接件（Component-and-connecto）视图，刻画运行时刻计算部件（Component）及其间交互关系（Connector）
   * 分配（Allocation）视图， 刻画软件系统与外界环境间的关系
2. 4+1视图分类：
   * 逻辑视图（LOGICAL VIEW）
     + 应用相关核心概念的抽象
       - 元素（类）
       - 关系（泛化，依赖，聚合等等）
       - 主要关注功能需求
   * 过程视图(PROCESS VIEW)
     + 刻划：并发 、同步、分布等特性，专注于性能、容错、可用性、系统一致性等非功能质量属性方面的考虑
   * 实现(开发)视图(IMPLEMENTATION/DEVELOPMENT VIEW)
     + 软件的静态组成：模块划分，相互联系
     + 软件的开发方便程度、复用、模块化等内部质量因素和工程组织管理因素
   * 部署视图(DEPLOYMENT VIEW
     + 软件模块的物理分布
     + 关注可用性、可靠性、性能、扩展性等非功能指标
   * 用例视图(USE CASE VIEW)
     + 用若干用例来串接上述四个视图
     + 两方面作用：
       - 各个视图的设计，实际上均是从用例出发来发现其中的元素、关系等
       - 通过用例场景的推演，来展示并验证体系结构。

## Chpater3

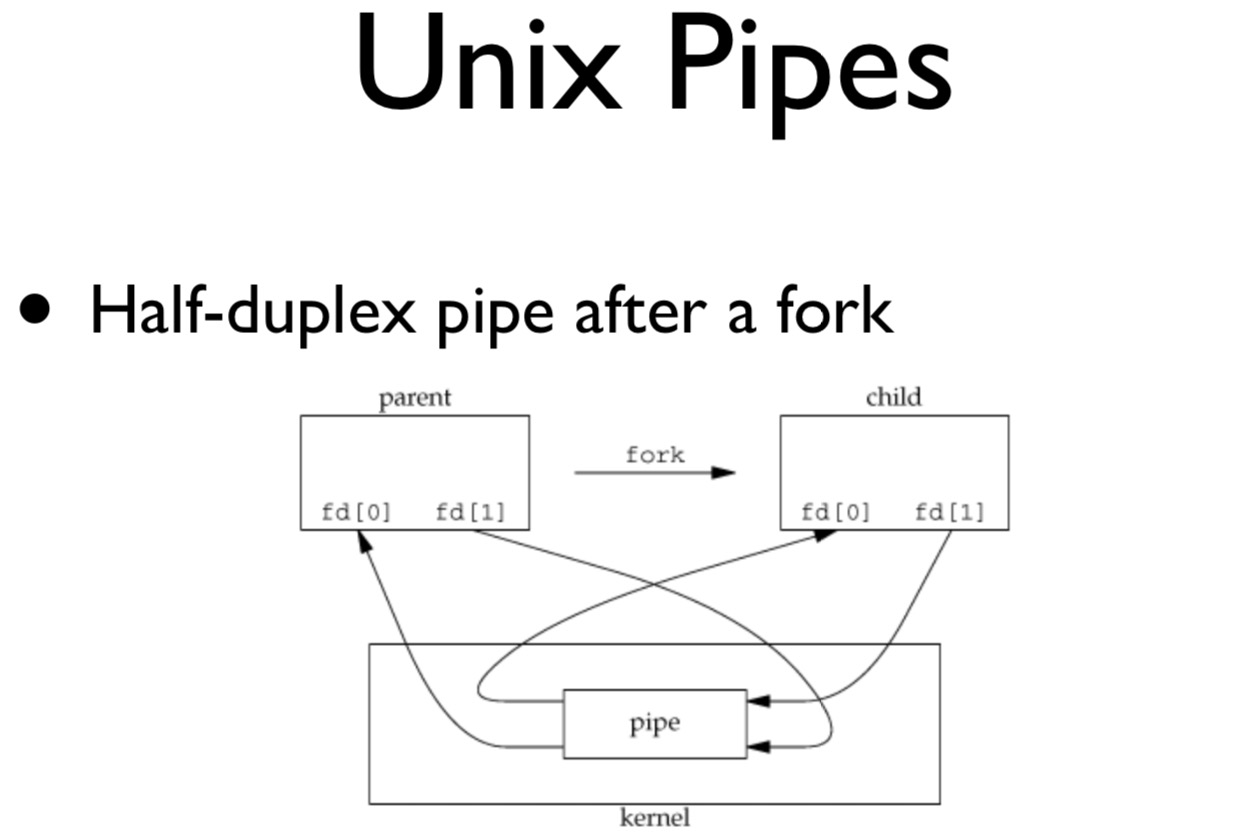
1. MVC设计模式：
   * 系统分割为processing, output和input三个部分
     + “Model”: processing，核心数据和功能
     + “View”: output，从Model获得数据并显示给用户
     + “Controller”: input，处理事件操作模型

## Chapter4

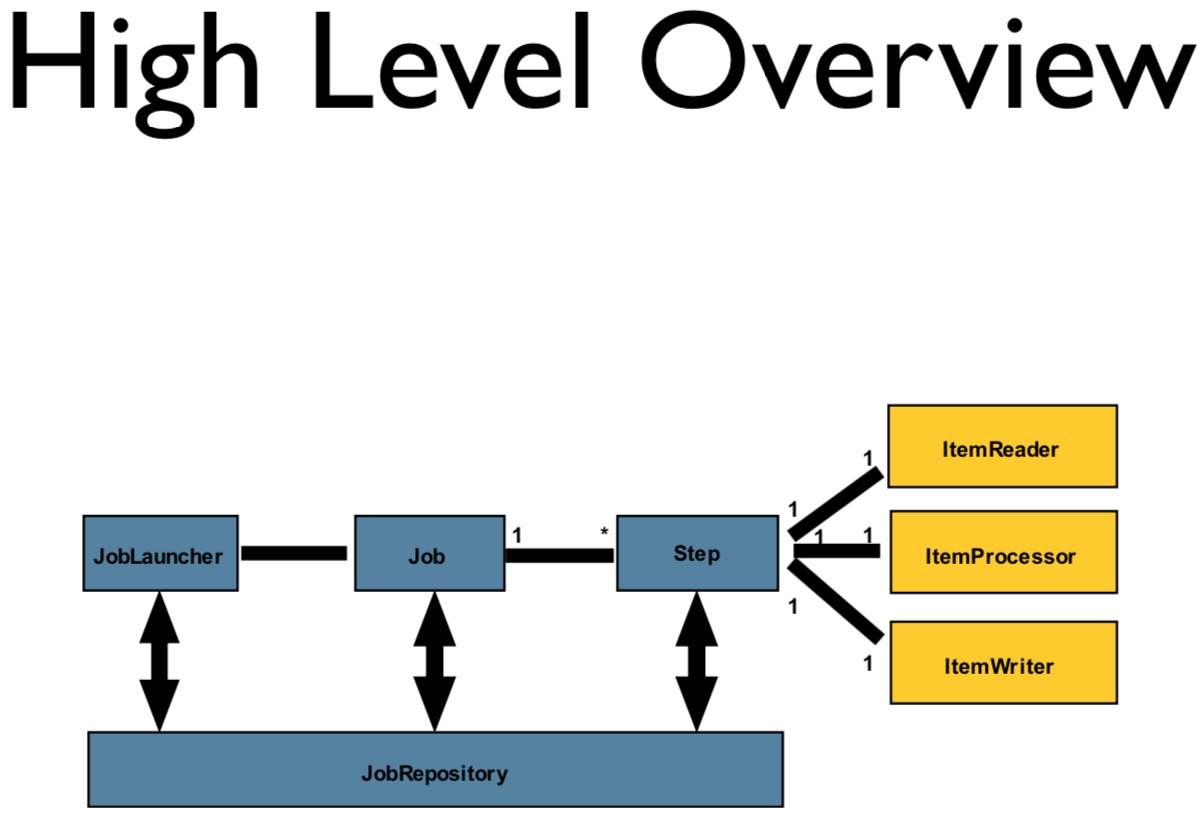
1. 层次设计模式：
   * 分层网络，TCP/IP协议设计
   * 优点：
     + 实现了对复杂问题的分步求解
     + 自顶向下看:不断的假设过程
     + 自底向上看:不断的抽象过程
     + 各层语义良好
       - 具有明确的使⽤场景
       - 提供良好的复⽤条件

## Chapter5

1. 管道-过滤器模式：
   * 组成方式: Pipes + Filters
     + Data Source -> Filter -> Data Sink
     + Filter完成单步数据处理功能
     + Data Source/Data Sink/Filter以Pipe连接
     + Pipe连接相邻模块，前⼀模块的输出为后⼀模块的输⼊
   * 优点：
     + 过滤器
       - 重⽤
       - 重组合
       - 可替换
     + 不需保存中间结果
     + 高效的并⾏处理理(多active部件)
   * 缺点：
     + 数据传输开销较⼤
     + 数据转换开销较⼤
     + 错误处理较为复杂
2. 过滤器工作方式
   * 过滤器是流水线的处理单元，负责丰富、提炼或转换其输⼊数据。 它以下⾯的三种⽅式⼯作:
     + 随后的流⽔线单元从过滤器中拉出数据。
     + 前面的流水线单元把新的输入数据压⼊过滤器。
     + 过滤器以循环的方式工作，从流水线中拉出输入数据并且将其输出数据压入流水线。
   * 前两种是被动过滤器，后⾯的是主动过滤器。多主动过滤器需要设置同步和缓冲。



1. SOAP协议：简单理解：以XML形式表示服务调⽤过程中的消息
   * 二进制消息PlainText消息
   * 平台无关性
2. SpringBatch：
   * 批处理框架

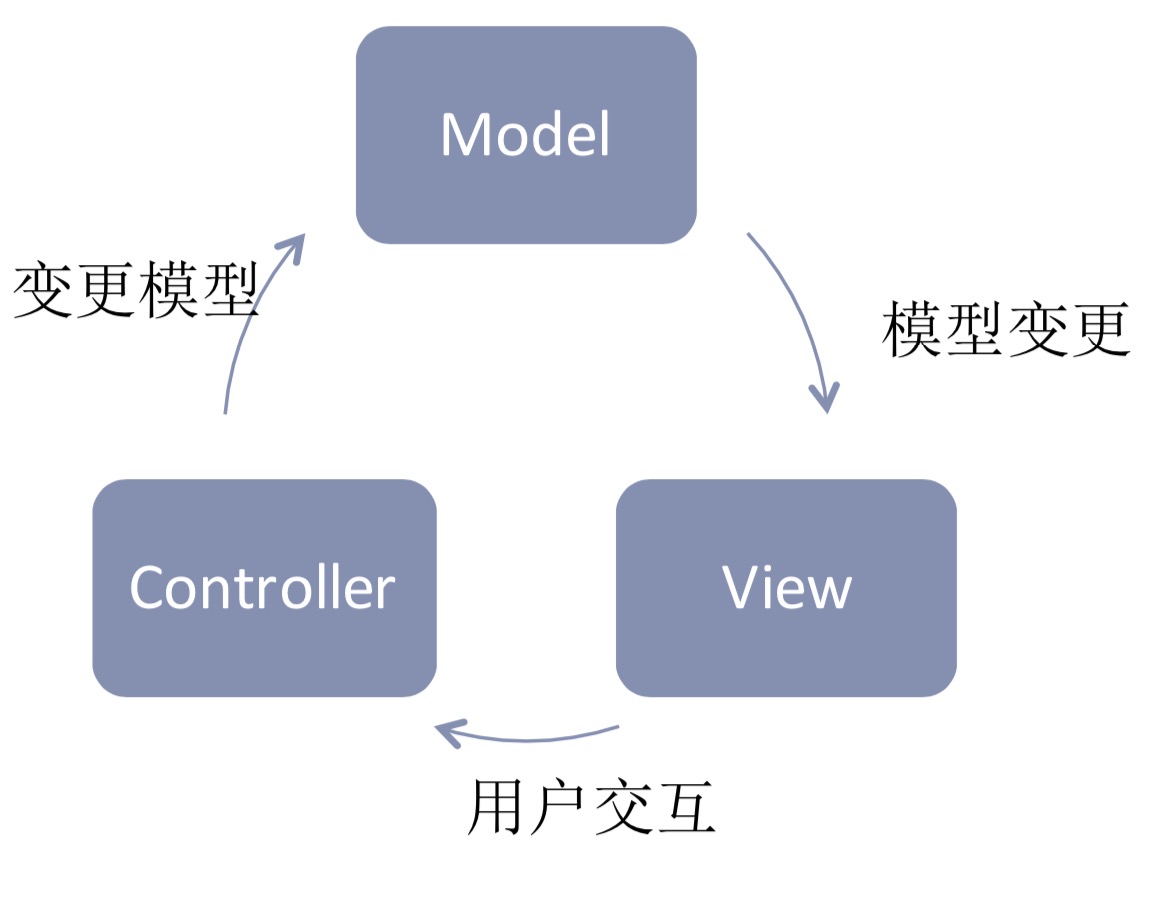


## Chapter6

1. 反转控制（IoC）设计模式：
   * 组件（Component）无需创建其依赖对象的实例，但是有其他模块或方式为它们创建这些实例。
   * 降低耦合度
2. 依赖注入（DI）设计模式：
   * 三种方式：
     + 构造注入
     + 接口注入
     + 属性设置注入

## Chapter7

1. MVC设计模式图例：

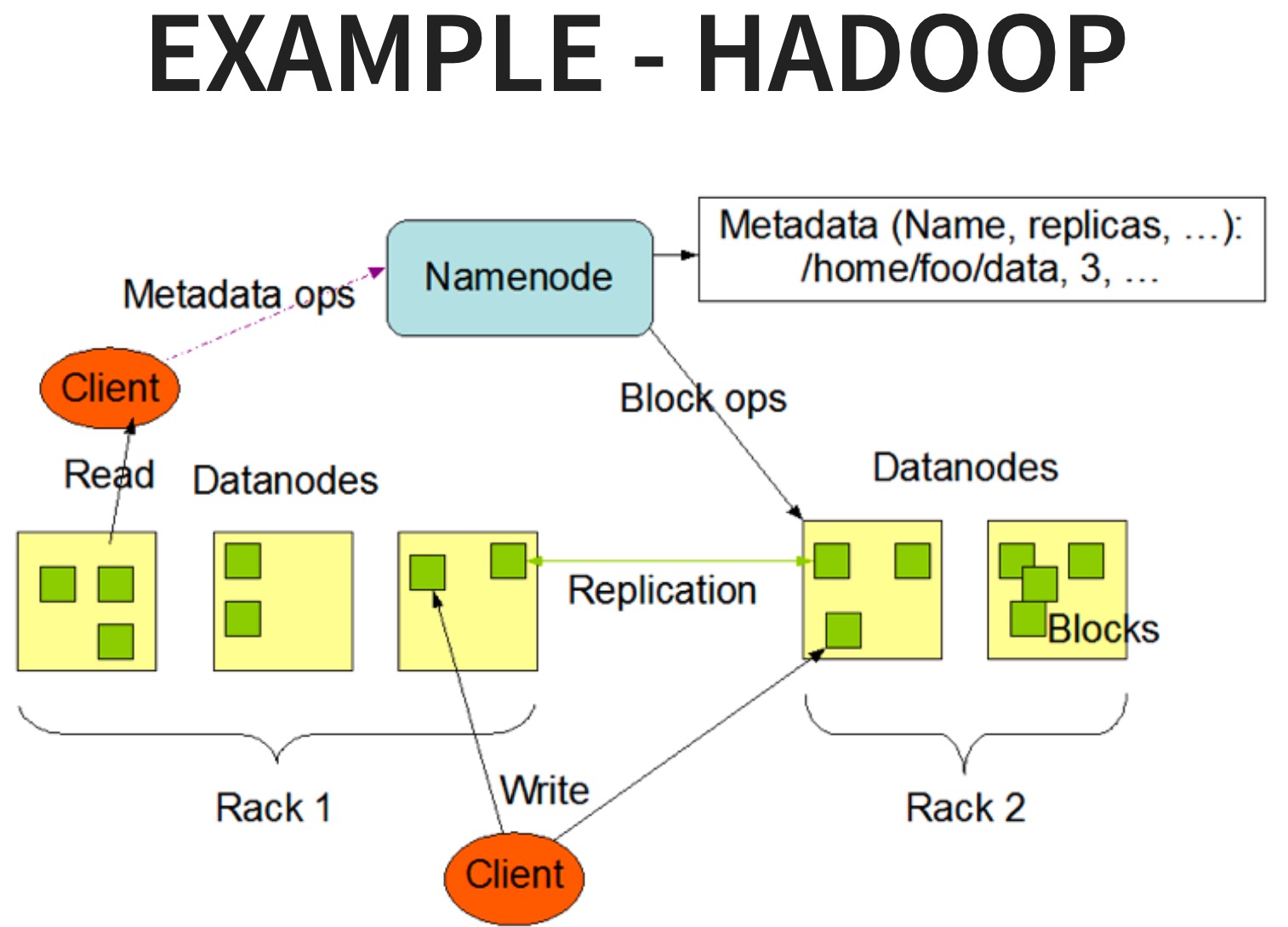


## Chapter8

1. C/S设计模式：
   * 优点
     + 资源共享
     + 集中化管理理
     + 性能分布
     + 安全
   * Fat Client：
     + 特征：
       - 应⽤系统在Client端运⾏
       - Client知道Server上的数据、⽂件等如何组织和存储
       - 是⼀种C/S模式的传统形式
       - 一般用户个⼈软件系统
     + 优点：
       - 为⽤户端程序的设计和开发提供较⼤的灵活性和便利性
   * Fat Server：
     + 特征：
       - Server通过一组确定的过程(或方法)提供资源的访问，⽽不是提供对资源的直接操作
       - Client提供GUI界⾯供⽤户进行操作，并通过远程⽅法调用与Server通信，获得服务
     + 优点：
       - 应⽤代码集中于Server端，便于部署和管理
       - 减少⽹络通信开销
2. 可扩展性：
   * 纵向扩展
   * 横向扩展
     + 传输端负载均衡
       - 基于DNS的负载均衡
       - TCP/IP负载均衡
     + 分布式缓存
     + 一致性哈希
     + 虚拟结点
     + 应用端负载均衡
       - 服务器/客户机
       - 会话

## Chapter9

1. **HDFS结构及相关模型**



## Chapter10

1. REST设计模式
   * REST指的是⼀组架构约束条件和原则
     + 为设计一个功能强、性能好、适宜通信的Web应用
     + 如果一个架构符合REST的约束条件和原则，我们就称它为RESTful架构
     + 原则和条件如下：
       - 统一接口原则
       - “资源”的具体呈现形式叫做“表现”
       - 状态转移（无状态通信原则）：
         * 状态应该区分应用状态和资源状态，客户端负责维护应⽤状态，⽽服务端维护资源状态
         * 客户端与服务端的交互必须是⽆状态的，并在每一次请求中包含处理该请求所需的一切信息
         * 服务端不需要在请求间保留应⽤状态，只有在接受到实际请求的时候，服务端才会关注应用状态
         * 这种⽆状态通信原则，使得服务端和中介能够理解独立的请求和响应
   * 核⼼概念
     + 资源(Resources)
     + 表现层(Representation)
     + 状态转化(State Transfer)

## Chapter11

1. P2P架构设计：即对等网络结构
   * P2P体系结构取消了服务器的中心地位，各个系统内计算机可以通过交换直接共享计算机资源和服务
   * 在这种体系结构中，计算机可对其他计算机的要求进行响应，请求响应范围和方式都根据具体应用程序不同而有不同的选择

## Chapter12

1. Microservices架构设计
   * **概念**：把一个大型的单个应用程序和服务拆分为数个甚至数十个的支持微服务，它可扩展单个组件而不是整个的应用程序堆栈，从而满足服务等级协议
   * **定义**：围绕业务领域组件来创建应用，这些应用可独立地进行开发、管理和迭代。在分散的组件中使用云架构和平台式部署、管理和服务功能，使产品交付变得更加简单
   * **本质**：用一些功能比较明确、业务比较精练的服务去解决更大、更实际的问题