JavaEE平台技术
预备知识

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

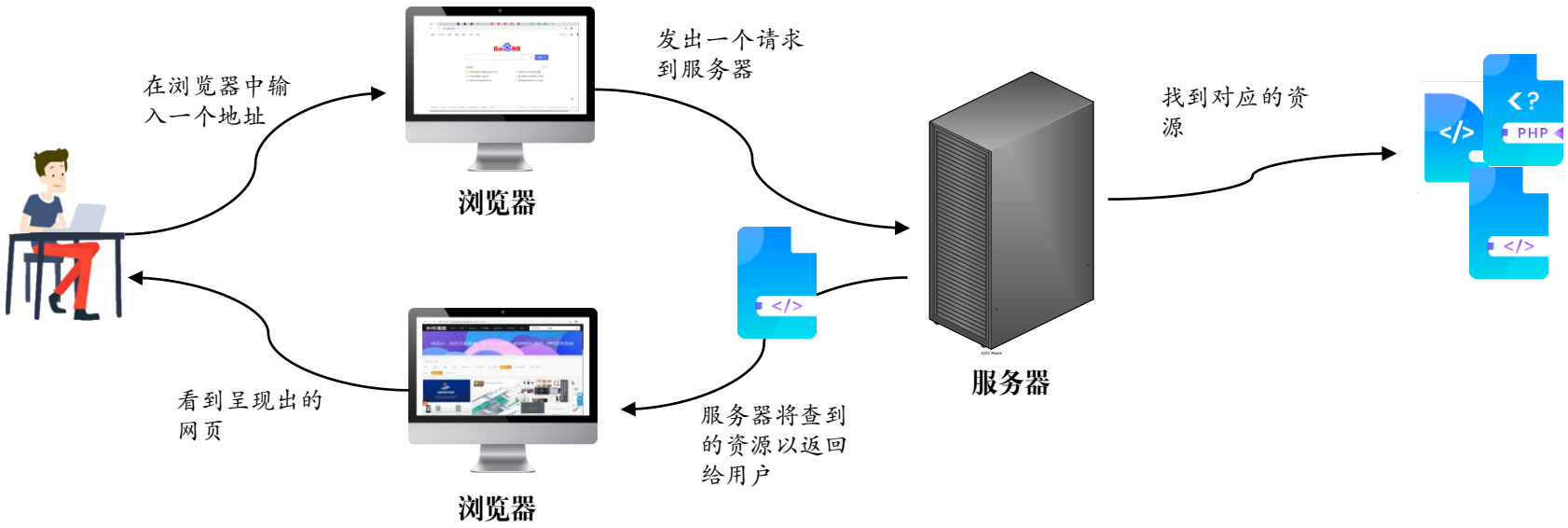
1. Web访问过程

- 统一资源定位系统（Uniform Resource Locator）是互联网上用于指定信息位置的表示方法。

网络协议　服务器名称　端口号　路径　资源

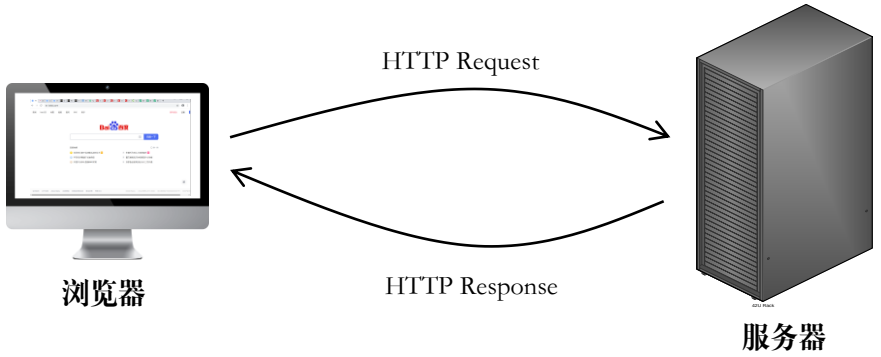
http://www.xmu.edu.cn:8080/deptment/teacher/info.html

1. Web访问过程


在浏览器中输入一个地址
浏览器
发出一个请求到服务器
服务器
找到对应的资源
PHP
服务器将查到的资源以返回给用户
浏览器
看到呈现出的网页

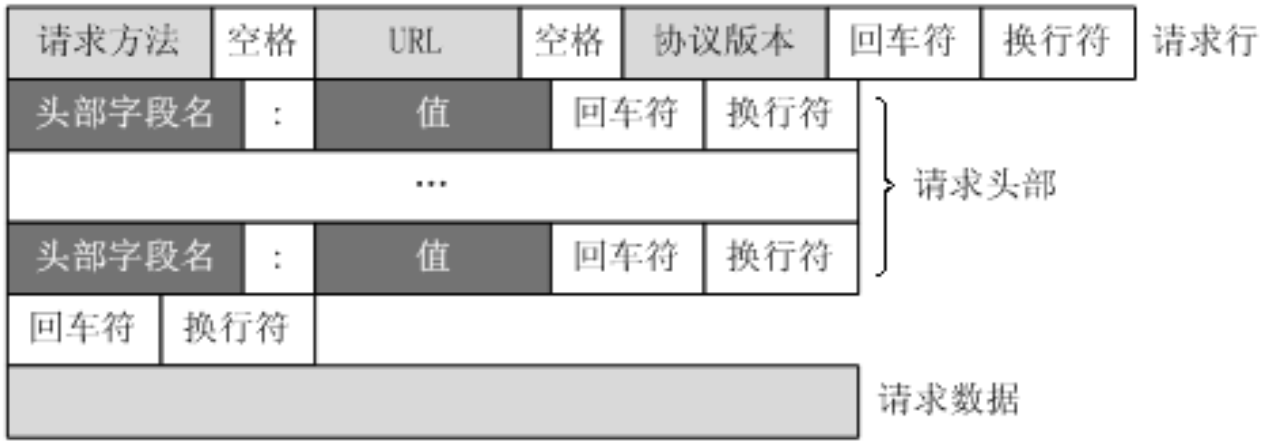
2. HTTP协议

- HTTP协议
  - 超文本传输协议（英文：HyperText Transfer Protocol，缩写：HTTP)是一种用于分布式、协作式和超媒体信息系统的应用层协议。



2. HTTP协议

- 协议格式
  - HTTP Request




```
GET /mix/76.html?name=kelvin&password=123456 HTTP/1.1
 Host: www.fishbay.cn
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/56.0.2924.87 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
```

2. HTTP协议

- 请求方法

请求方法	描述
OPTIONS	返回服务器针对特定资源所支持的HTTP请求方法，
GET	向特定的资源发出请求。请求体中无内容
POST	向指定资源提交数据进行处理请求。数据被包含在请求体中。
PUT	从客户端向服务器传送的数据取代指定的文档的内容。
DELETE	请求服务器删除指定的页面。
CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
TRACE	回显服务器收到的请求，主要用于测试或诊断。

2. HTTP协议

- 协议格式
  - HTTP Response

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 20 Feb 2017 09:13:59 GMT
Content-Type: text/plain;charset=UTF-8
Vary: Accept-Encoding
Cache-Control: no-store
Pragrma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Cache-Control: no-cache
Content-Encoding: gzip
Transfer-Encoding: chunked
Proxy-Connection: Keep-alive

{"code":200,"notice":0,"follow":0,"forward":0,"msg":0,"comment":0,"pushMsg":null,"friend":{"snsCount":0,"count":0,"celebrityCount":0}}

2. HTTP协议

- 协议格式
  - HTTP Response 状态码

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

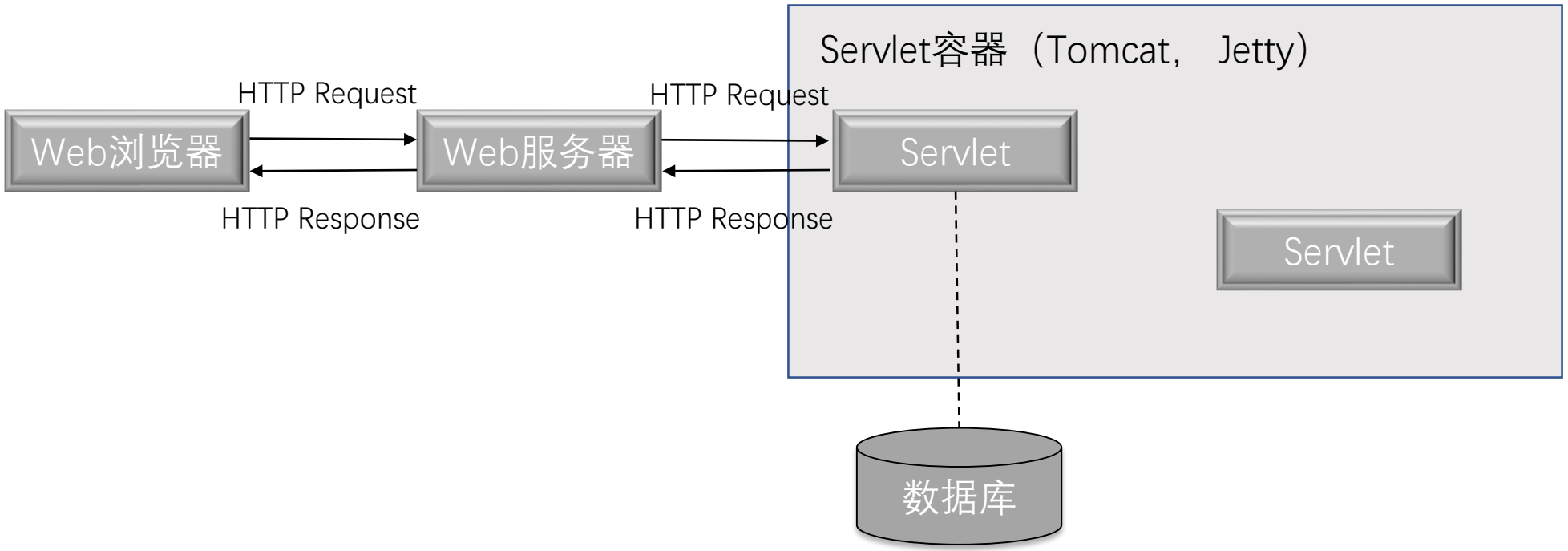
404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

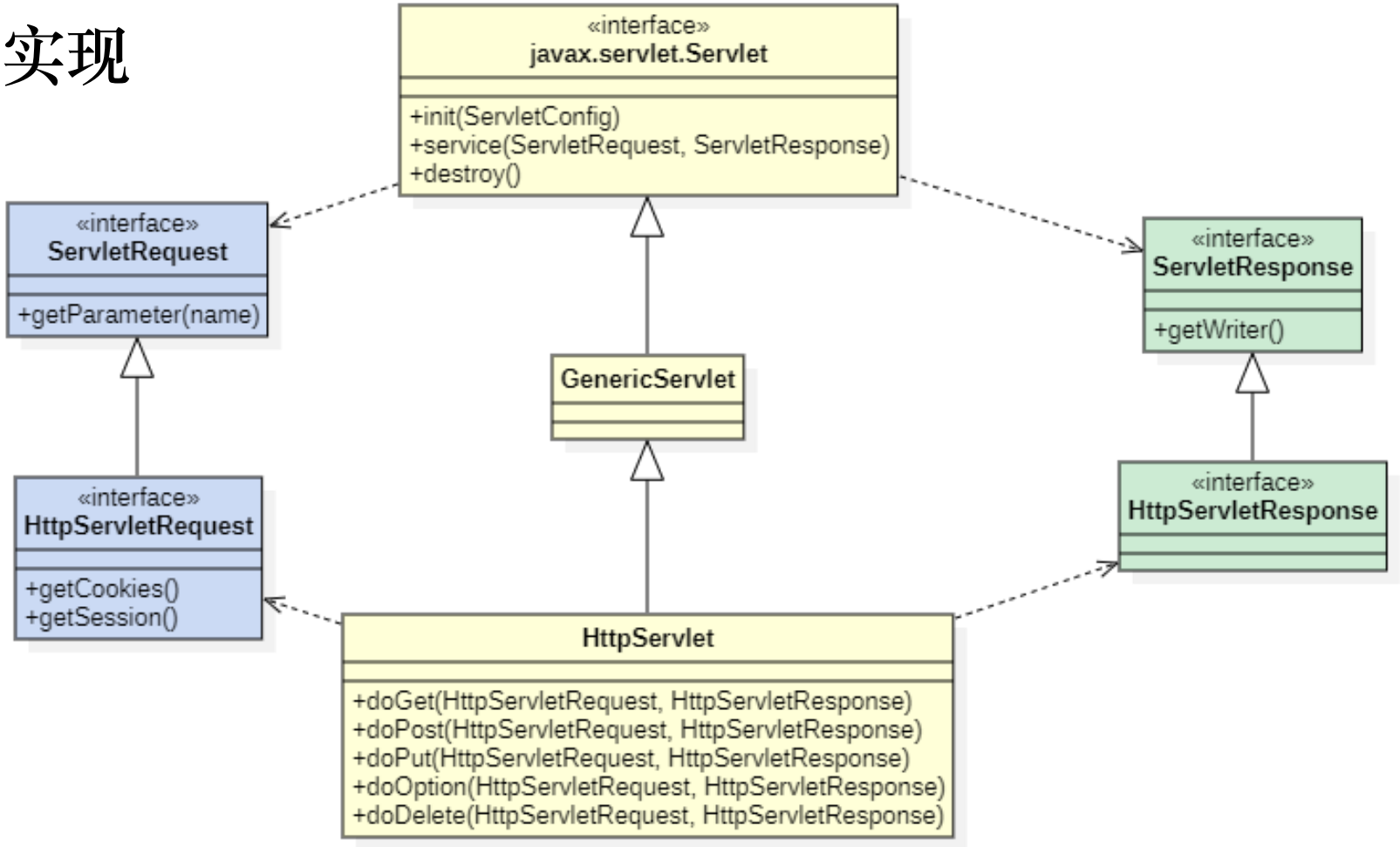
3. Servlet

- 运行在 Web 服务器或应用服务器上的程序，它可以收集来自网页表单的用户输入，呈现来自数据库的记录，动态生成网页。



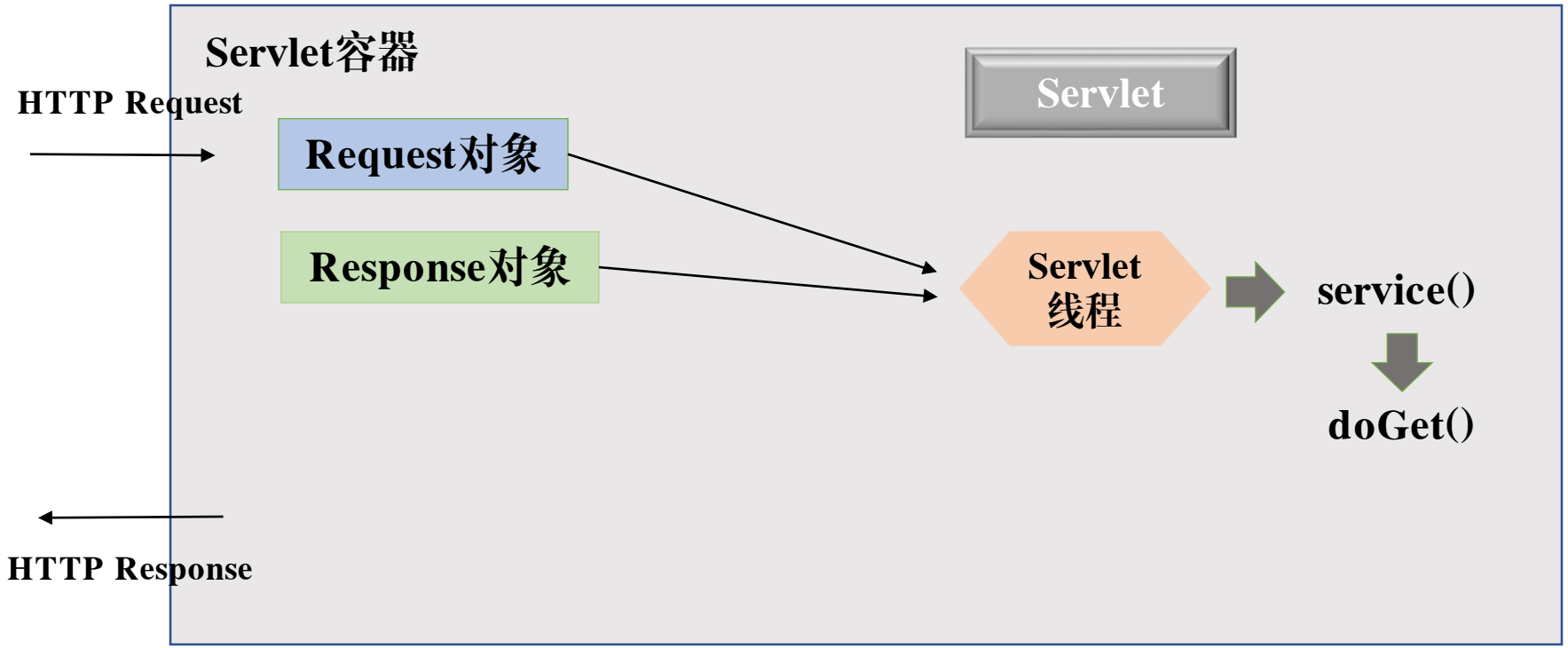
3. Servlet

- Servlet的实现

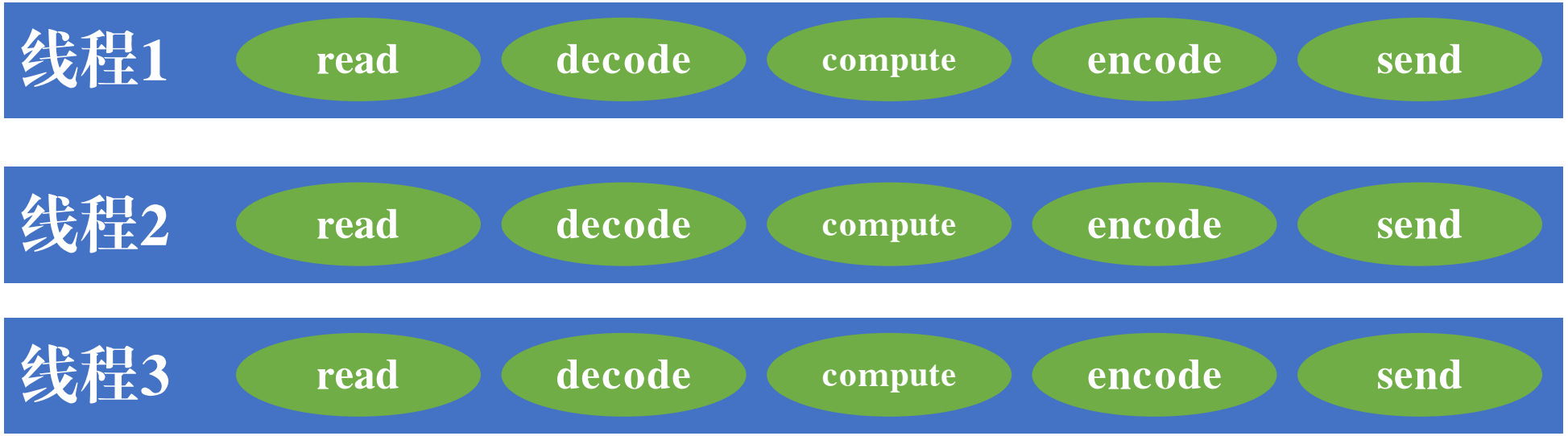


1.3. Servlet

- Servlet的工作机制



1.2.Spring技术栈



3. Servlet

- 通过线程池管理多线程
  - 降低资源消耗-->重复利用已经创建的线程
  - 提高响应速度-->当请求到达时而线程池中有空闲线程时候，可以直接从线程池获取线程,而不需要创建新的线程
  - 提高线程的可管理性-->使用线程池可以统一分配,调优和监控,例如可以根据系统的承受能力,调增线程池中工作线程的数目

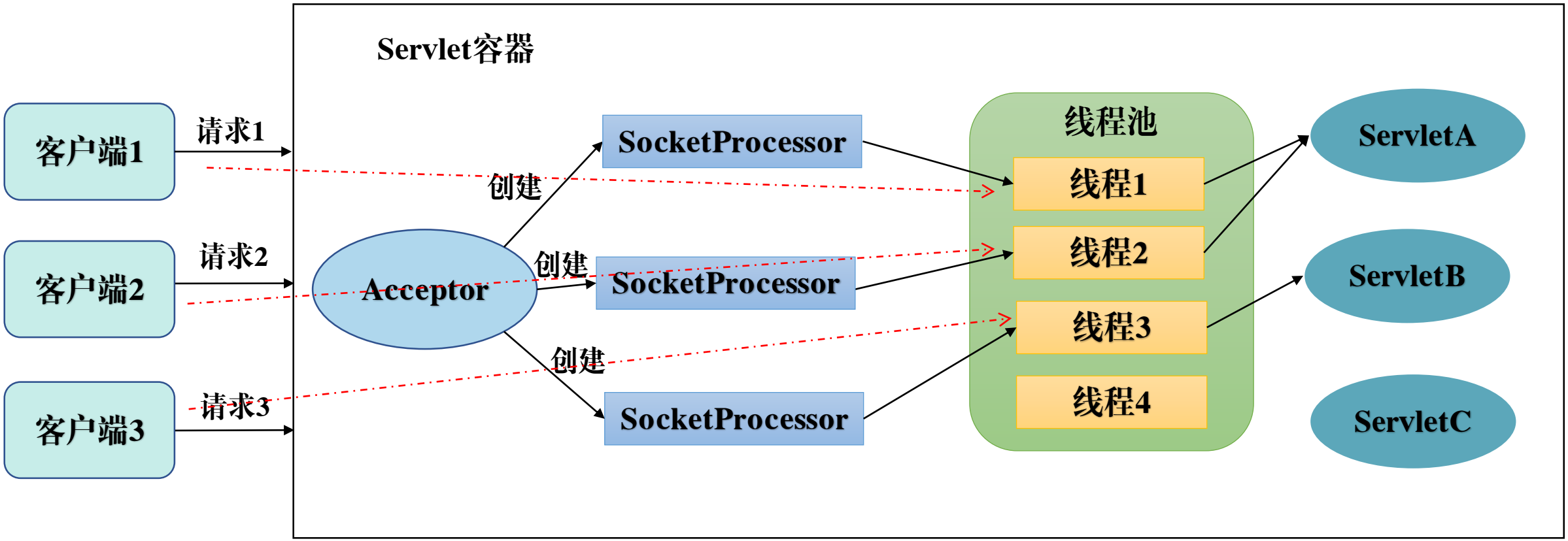
```
#最大工作线程数，默认200。
server.tomcat.max-threads=200

#最大连接数默认是10000
server.tomcat.max-connections=10000

#等待队列长度，默认100。
server.tomcat.accept-count=100

#最小工作空闲线程数，默认10。
server.tomcat.min-spare-threads=100
```

3. Servlet

- Tomcat的三种运行模式
  - BIO: 同步阻塞I/O模式
  - NIO： 异步I/O模式
  - APR：调用Apache HTTP服务器的核心动态链接库来处理文件读取或网络传输操作

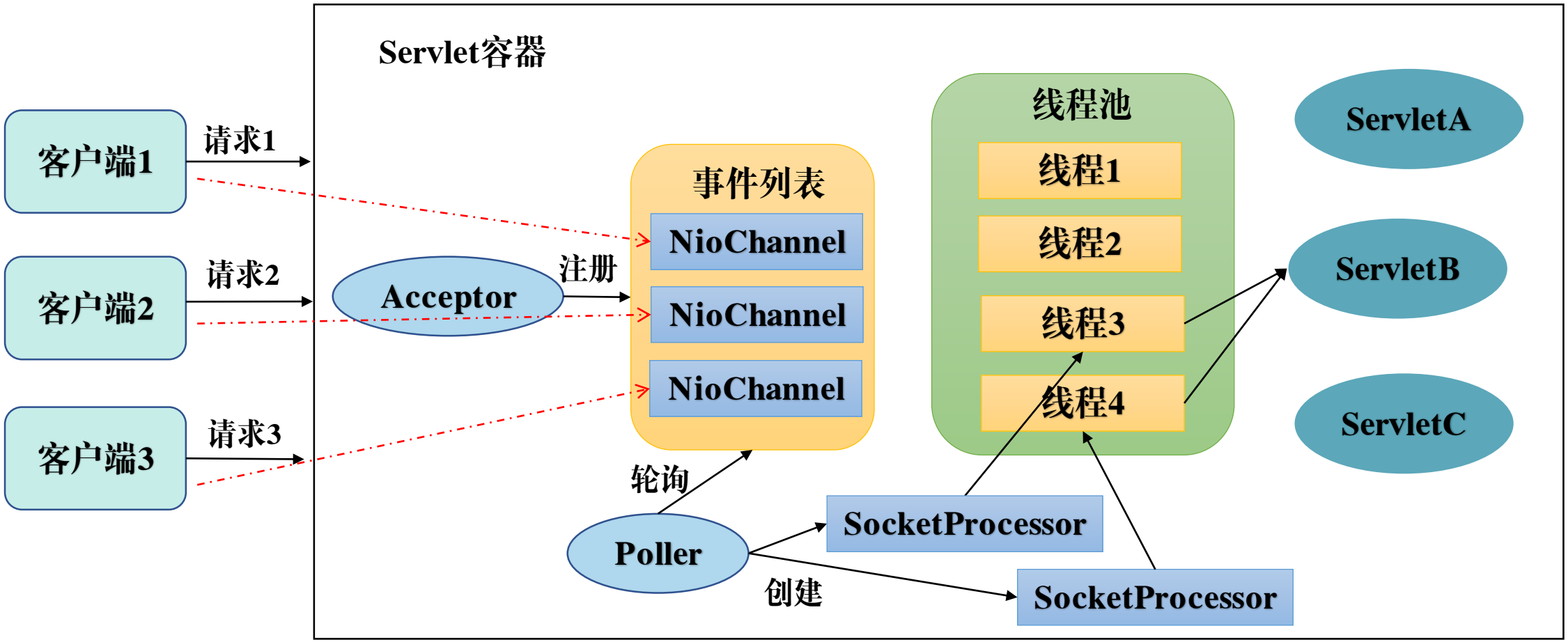
3. Servlet

- 线程模型 - BIO



3. Servlet

- 线程模型 - NIO



4. Maven

- Apache 下的一个纯 Java 开发的开源项目。基于项目对象模型（缩写：POM）管理一个项目的构建、依赖、报告和文档等步骤。
  - Maven是一个构建工具，实现自动化构建。
  - Maven是跨平台的，对外提供一致的操作接口。
  - Maven是一个依赖管理工具和项目信息管理工具。它还提供了中央仓库，能帮我们自动下载构件。
  - Maven是一个标准，对于项目目录结构、测试用例命名方式等内容都有既定的规则。

4. Maven

- Maven提倡使用一个共同的标准目录结构

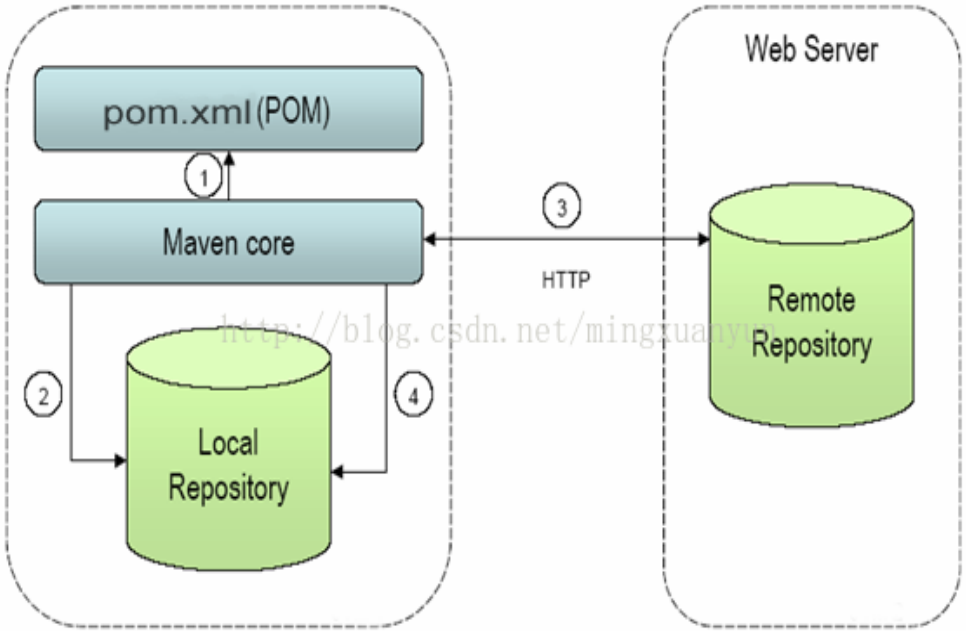
目录	用途
${basedir}	存放pom.xml和所有的子目录
${basedir}/src/main/java	项目的java源代码
${basedir}/src/main/resources	项目的资源，比如说property文件，springmvc.xml
${basedir}/src/test/java	项目的测试类，比如说Junit代码
${basedir}/src/test/resources	测试用的资源
${basedir}/src/main/webapp/WEB-INF	web应用文件目录，web项目的信息，比如存放web.xml、本地图片、jsp视图页面
${basedir}/target	打包输出目录
${basedir}/target/classes	编译输出目录
${basedir}/target/test-classes	测试编译输出目录
~/.m2/repository	Maven默认的本地仓库目录位置

4. Maven

- Maven POM
  - POM(Project Object Model，项目对象模型) 是一个XML文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖，等等。
  - Maven 会在当前目录中查找 并读取POM文件，获取所需的配置信息，然后执行目标。

4. Maven

- Maven会自动根据dependency中的依赖配置，直接通过互联网在Maven中央仓库（https://mvnrepository.com/）下载相关依赖包到本地Maven库，本地Maven库默认是用户目录的.m2目录



5. Maven

- Maven 的三个相互独立的生命周期：
  - clean：项目构建前的清理工作，删除前一次构建在target文件夹下生成的各个Jar包等
  - default：构建，包括项目的编译，测试，打包，安装，部署等等
  - site：生成项目报告，发布站点，Maven可以根据pom所包含的信息，生成一个站点，方便团队交流和发布项目信息，

4. Maven

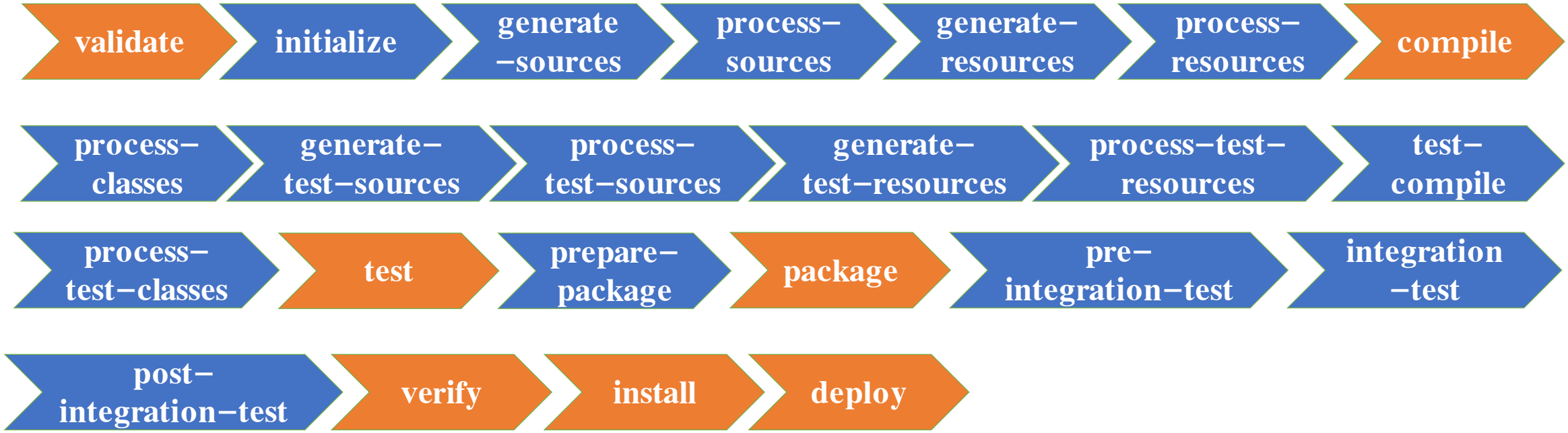
- Maven的清理(Clean)生命周期
  - pre-clean：执行一些清理前需要完成的工作
  - clean：清理上一次构建生成的文件
  - post-clean：执行一些清理后需要完成的工作

4. Maven

- Maven的构建(Default)生命周期
  - validate: 验证项目是否正确且所有必需资源是否可用
  - initialize:初始化编译的状态，例如：设置一些properties属性，或者创建一些目录
  - generate-sources:通常是通过插件支持创建额外的源代码。
  - process-sources:处理源代码，例如：替换值（filter any values)
  - generate-resources:生成这个项目包所有需要包含的资源文件
  - process-resources:复制并处理资源文件到目标目录，为packaging 打包阶段做好准备
  - compile：编译项目的源码
  - process-classes:后置处理编译阶段生成的文件，例如：做java字节码的增强操作
  - generate-test-sources:生成编译阶段需要的test源代码
  - process-test-sources:处理test源代码，
  - generate-test-resources:生成test测试需要的资源文件
  - process-test-resources:复制并处理资源文件到test测试目标目录
  - test-compile:编译项目单元测试代码
  - process-test-classed:后置处理test编译阶段生成的文件，例如：做java字节码的增强操作
  - test：使用单元测试框架运行测试，测试代码不会被打包或部署
  - prepare-package:处理任何需要在正式打包之前要完成的必须的准备工作。这一步的通常结果是解压，处理包版本等
  - package：创建JAR/WAR包如在 pom.xml 中定义提及的包
  - pre-integration-test:完成一些在集成测试之前需要做的预处理操作，这通常包括建立需要的环境。
  - integration-test:处理并部署（deploy）包到集成测试可以运行的环境中执行系统集成测试。
  - post-integration-test:处理一些集成测试之后的事情，通常包括一些环境的清理工作
  - verify：对集成测试的结果进行检查，以保证质量达标
  - install：安装打包的项目到本地仓库，以供其他项目使用
  - deploy：拷贝最终的工程包到远程仓库中，以共享给其他开发人员和工程

4. Maven

- Maven的构建(Default)生命周期



4. Maven

- Maven的站点(Site)生命周期
  - pre-site: 执行一些实际站点生成之前的预处理操作
  - site：生成项目站点文档
  - post-site: 执行一些后置操作并完成最终生成站点操作，并为最后站点发布做好准备
  - site-deploy：将生成的项目站点发布到服务器上

4. Maven

- Maven 插件
  - Maven 实际上是一个依赖插件执行的框架，每个任务实际上是由插件完成。每个插件可以完成多个功能，每个功能称为插件目标（Plugin Goal）。

插件目标	描述
maven-compile-plugin:compile	编译位于src/main/java/目录下的主源码
maven-compile-plugin:testCompile	编译位于src/test/java/目录下的测试源码。

4. Maven

- 插件与阶段的内置绑定
  - Clean生命周期

阶段	内置绑定的插件目标
clean	maven-clean-plugin:clean

4. Maven

- 插件与阶段的内置绑定
  - Default生命周期（当packaging的值是jar/war）

阶段	内置绑定的插件目标
process-resource	maven-resources-plugin:resources
compiler	maven-compiler-plugin:compile
process-test-resources	maven-resources-plugin:testResources
test-compile	maven-compiler-plugin:testCompile
test	maven-surefire-plugin:test
package	maven-jar-plugin:jar/maven-war-plugin:war
install	maven-install-plugin:install
deploy	maven-deploy-plugin:deploy

4. Maven

- 插件与阶段的内置绑定
  - Default生命周期（当packaging的值是pom）

阶段	内置绑定的插件目标
install	maven-install-plugin:install
deploy	maven-deploy-plugin:deploy

4. Maven

- 插件与阶段的内置绑定
  - site生命周期

阶段	内置绑定的插件目标
site	maven-site-plugin:site
site-deploy	maven-site-plugin:deploy

4. Maven

- 父（Super）POM
  - 父（Super）POM是 Maven 默认的 POM。所有的 POM 都继承自一个父 POM（无论是否显式定义了这个父 POM）。父 POM 包含了一些可以被继承的默认设置
  - 可以用Show Effective POM看到最终有效的POM定义

4. Maven

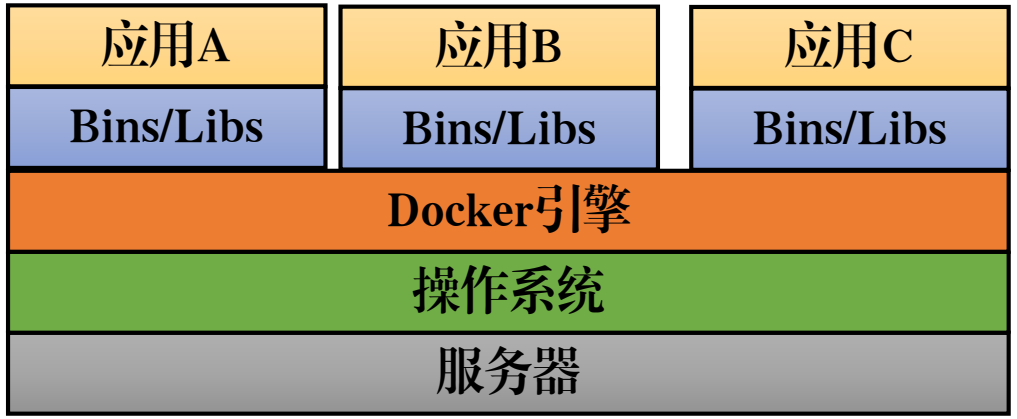
- 父子项目

```
1. +- pom.xml
2. +- my-app
3. | +- pom.xml
4. | +- src
5. |   +- main
6. |     +- java
7. +- my-webapp
8. | +- pom.xml
9. | +- src
10.|   +- main
11.|     +- webapp
```

```
1.<project xmlns="http://maven.apache.org/POM/4.0.0"
2.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4.                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.  <modelVersion>4.0.0</modelVersion>
6.
7.  <groupId>com.mycompany.app</groupId>
8.  <artifactId>app</artifactId>
9.  <version>1.0-SNAPSHOT</version>
10. <packaging>pom</packaging>
11.
12.  <modules>
13.    <module>my-app</module>
14.    <module>my-webapp</module>
15.  </modules>
16.</project
```

5. Docker

- Docker是一款以容器虚拟化技术为基础的软件
- 容器是指通过操作系统自身支持一些接口，让应用程序间可以互不干扰的独立运行，并且能够对其在运行中所使用的资源进行干预。



5. Docker

- Docker 镜像（Image）
  - 类似于虚拟机镜像，可以理解为一个面向Docker引擎的只读模板，
  - 一个镜像可以包含一个完整的ubuntu，里面仅安装apache或用户的其他应用
- Docker 容器（Container）
  - 容器类似于一个轻量级的沙箱，Docker利用容器来运行和隔离应用。
  - 容器是从镜像创建的应用运行实例，可以将其启动、开始、停止、删除，
  - 容器Container之间都是相互隔离、互不可见的。

5. Docker

- Docker的常用命令
  - docker run： 创建一个新的容器并运行一个命令
  - docker start/stop/restart :启动/停止/重启一个容器
  - docker rm： 删除一个或多个容器。
  - docker ps：列出容器
  - docker images：列出本地镜像。
  - docker rmi：删除本地一个或多个镜像。

5. Docker

- 用Dockerfile创建镜像
  - Dockerfile由一系列指令和参数组成，基于DSL语法。每条指令都必须为大写字母，后面要跟随一个参数

5. Docker

- Dockerfile常用命令
  - FROM：每个Dockerfile的第一条指令都应该是FROM。FROM指令指定一个已经存在的镜像，后续指令都是将基于该镜像进行，这个镜像被称为基础镜像（base image）
  - MAINTAINER：这条指令会告诉Docker该镜像的作者是谁，以及作者的邮箱地址。这有助于表示镜像的所有者和联系方式。
  - RUN：RUN指令会在当前镜像中运行指定的命令。每条RUN指令都会创建一个新的镜像层，如果该指令执行成功，就会将此镜像层提交，
  - EXPOSE：EXPOSE指令是告诉Docker该容器内的应用程序将会使用容器的指定端口。
  - ADD：复制本机文件或目录或远程文件，添加到指定的容器目录
  - ARGS: 添加Dockerfile编译时参数

5. Docker

- ENTRYPOINT和CMD
  - ENTRYPOINT与CMD都可以让容器启动时执行一条命令
  - 在启动容器时，CMD会被docker run命令给出参数所覆盖， 而ENTRYPOINT不会

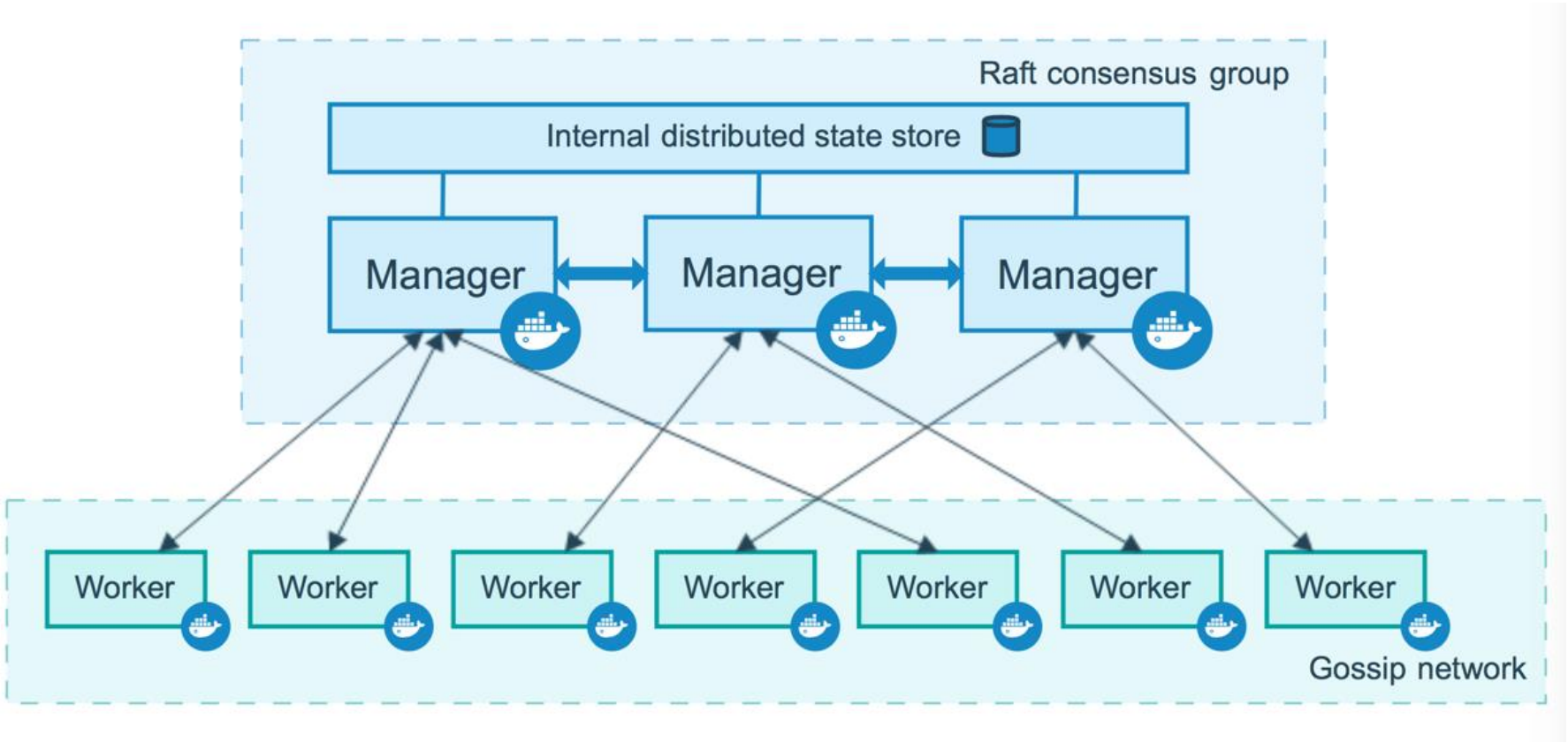
5. Docker

```
FROM openjdk:11-jre
MAINTAINER mingqiu mingqiu@xmu.edu.cn
WORKDIR /app
ARG JAR_FILE
ADD ${JAR_FILE} /app/app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar"]
CMD ["app.jar"]
```

5. Docker

- Docker的集群—Swarm
  - 主要作用是把若干台运行在Swarm模式的Docker主机抽象为一个整体，并且通过一个入口统一管理这些Docker主机上的各种Docker资源

5. Docker


Raft consensus group
Internal distributed state store
Manager
Manager
Manager
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Gossip network

5. Docker

- Swarm的特点
  - 去中心化设计——所有的节点都平等的参与任务
  - 多主机网络——建立了覆盖多主机多容器的Overlay网络，所有容器自动分配地址
  - 服务发现——为每个节点分配了DNS名称
  - 负载平衡——可以将服务端口暴力给外部的负载平衡服务器，Swarm负责各节点的服务分配

5. Docker

- Manager节点
  - 负责管理集群的状态
  - 调度服务
- Worker节点
  - 接收和执行任务
- Service
  - 运行在Manage节点和Worker节点上的任务

Question & Answer