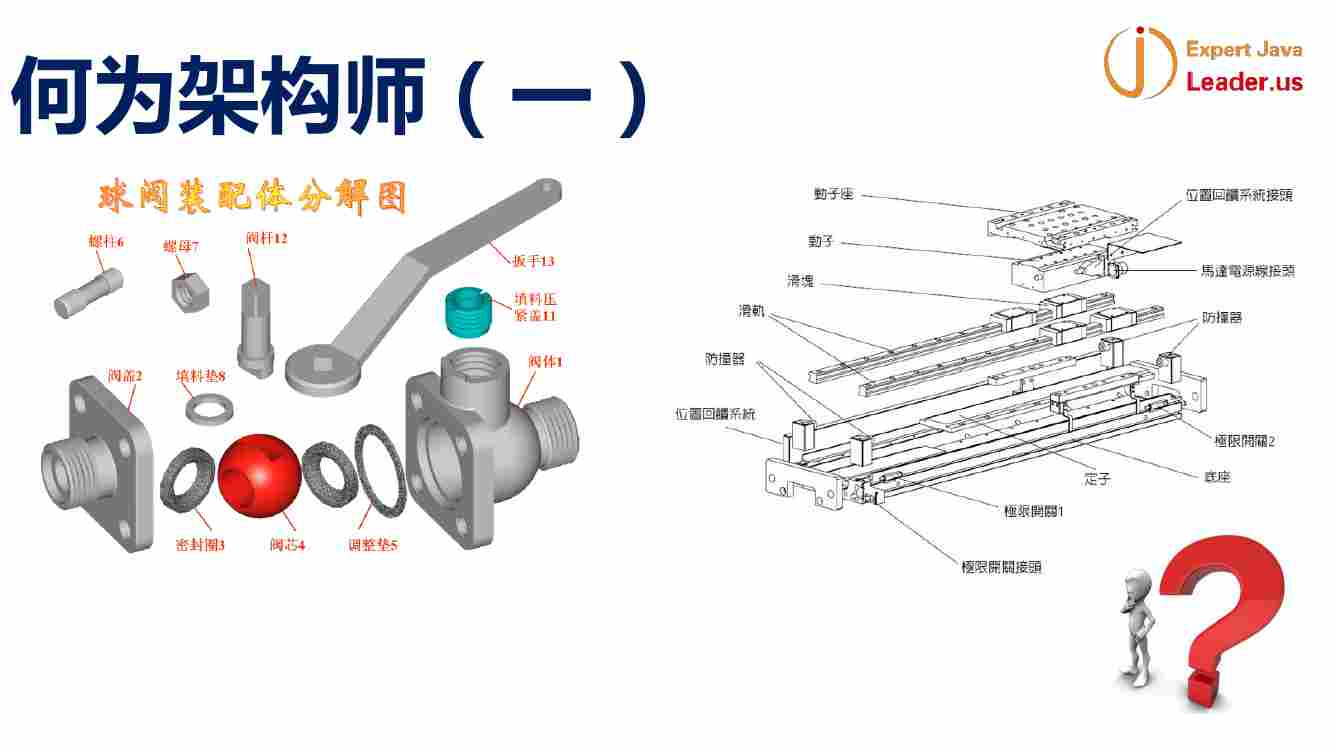




Spring整个是一个容器技术 里面的Component是组件 这个不是spring发明的 是一直存在的

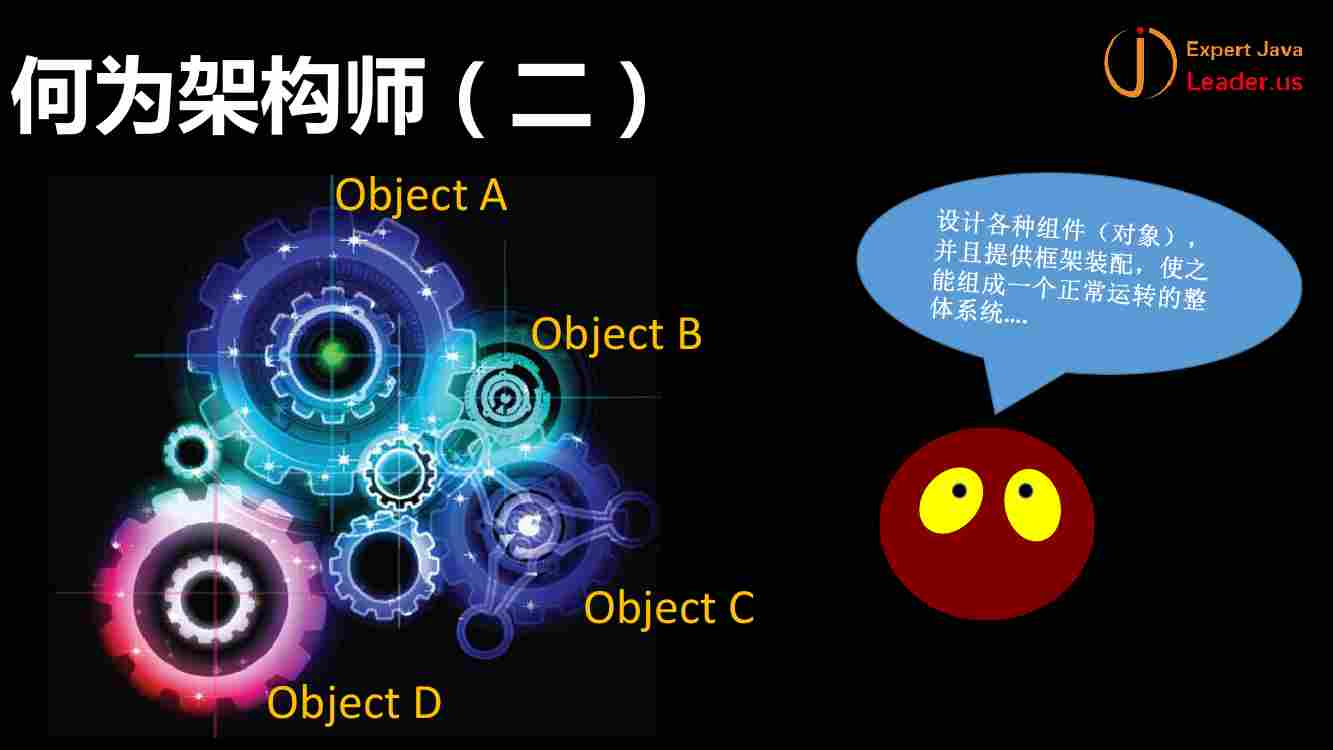


复杂机械系统有很多组件 有些组件是标准化的 ------ 这些组件是需要一个复杂的装配过程的

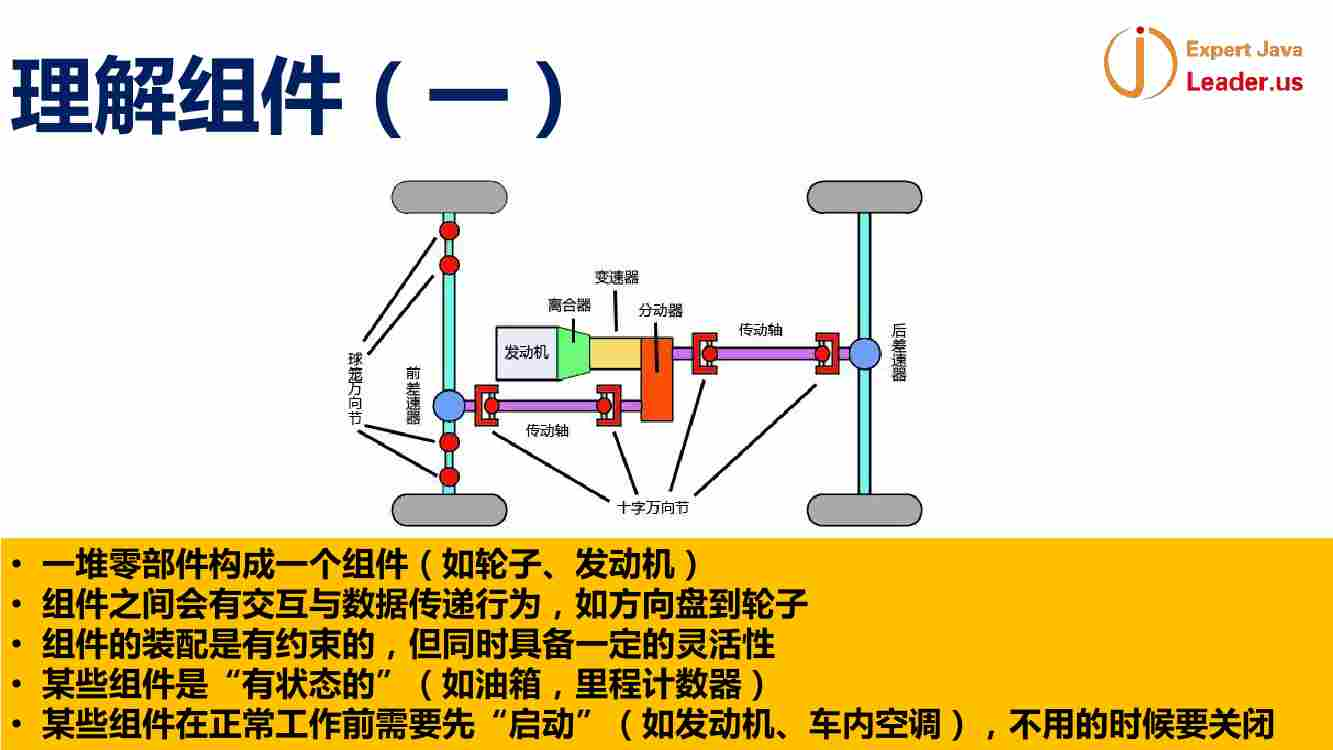
工人就是生产组件的（设计师的要求）

高级技工就是装配 画机械制图 然后知道别人去吧组件合理安装 装配组件

**设计组件 并给出装配指导图 就是架构师的任务**



架构是设计出各种组件的接口 然后提供装配框架 让系统运行起来 让组件发挥功能 这就是架构师要做的事情 ----- 正如图中所说



什么是Component？和机械里面一样 一堆零部件 构成一个组件 -- 一个组件由一系列相关的零部件组成 【一个发送机 可能由一系列的零部件构成 是不可分割的 整体代表一个共轭】

主键之间有交互和数据传递能力的：转动方向盘 最后会引发轮子的变相 这个过程就是把方向盘扭转 的信息 发送到轮子上的 ----- 数据传输的过程

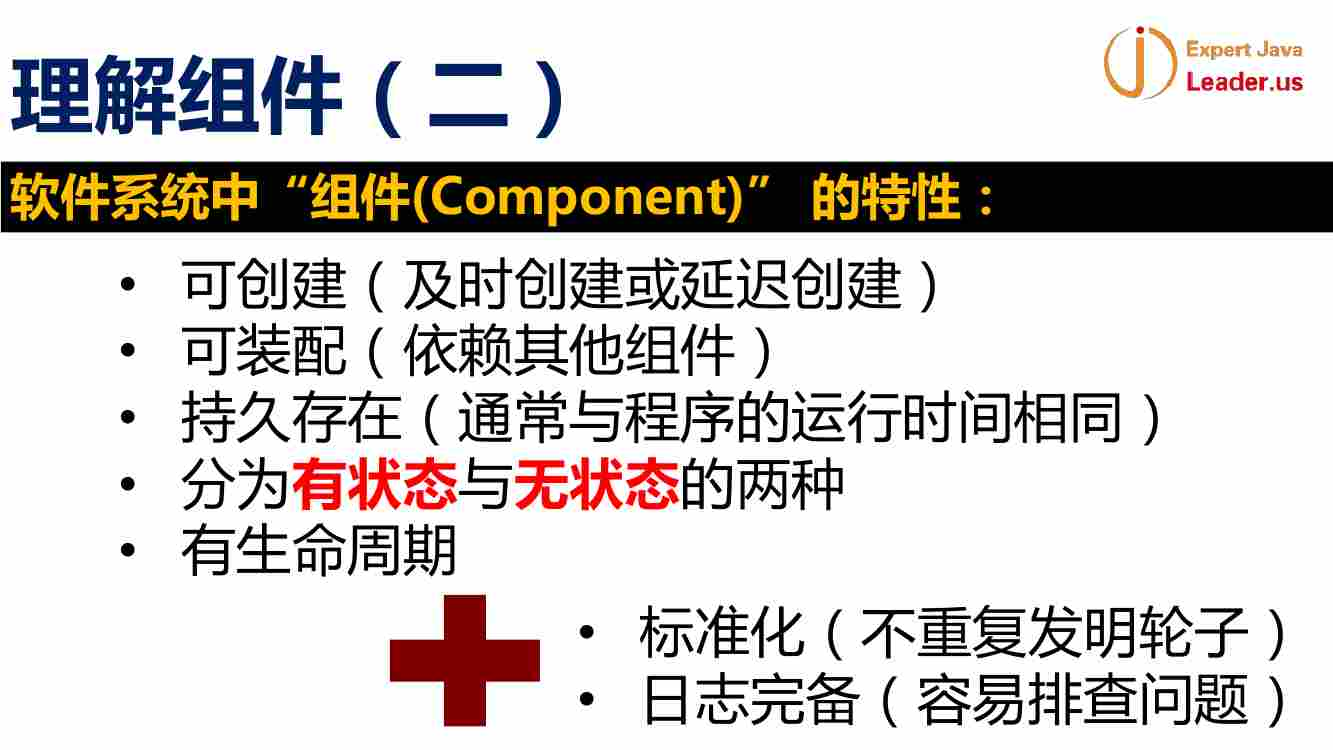
组件的装配是有约束的 不能随便放 每一个组件有自己存放位置 做自己的事情 ----- 对于软件来说 还有灵活性 可以被替换

组件本身有一个数据状态 比如油箱剩余多少油

某些主键正常工作之前 有一个启动的过程过程

不用的时候要关闭

这个是组件的生命周期 先要启动 然后正常运行 不使用之后 关闭 释放资源



Component特性

可创建 支持延迟创建和即时创建 --- 延迟创建需要的时候才创建 这个加速成的启动

可装配 工作的时候 依赖其他组件 或者为其他的组件提供服务

持久存在 一个组件创建之后 和程序的周期一样 不会临时创建 临时销毁 ---- 正常使用不会被销毁 不会被随意new

分成有状态和无状态两种

有生命周期

标准化 ----- 意味着复用能力很强 软件中 利用好的组件 不重复发明轮子 spring做得好

日志完备 ----- 组件运行的过程 有利于排查问题的过程 要进行日志完备 日志完备是非常重要 运行过程排查问题

Spring组件都具备这些特性和概念



每一个用户都是唯一的 用这个系统和你不一样的行为 不可能说 都用同样的数据 互联网领域 要求高 看起来向定制化的 感知用户的特征



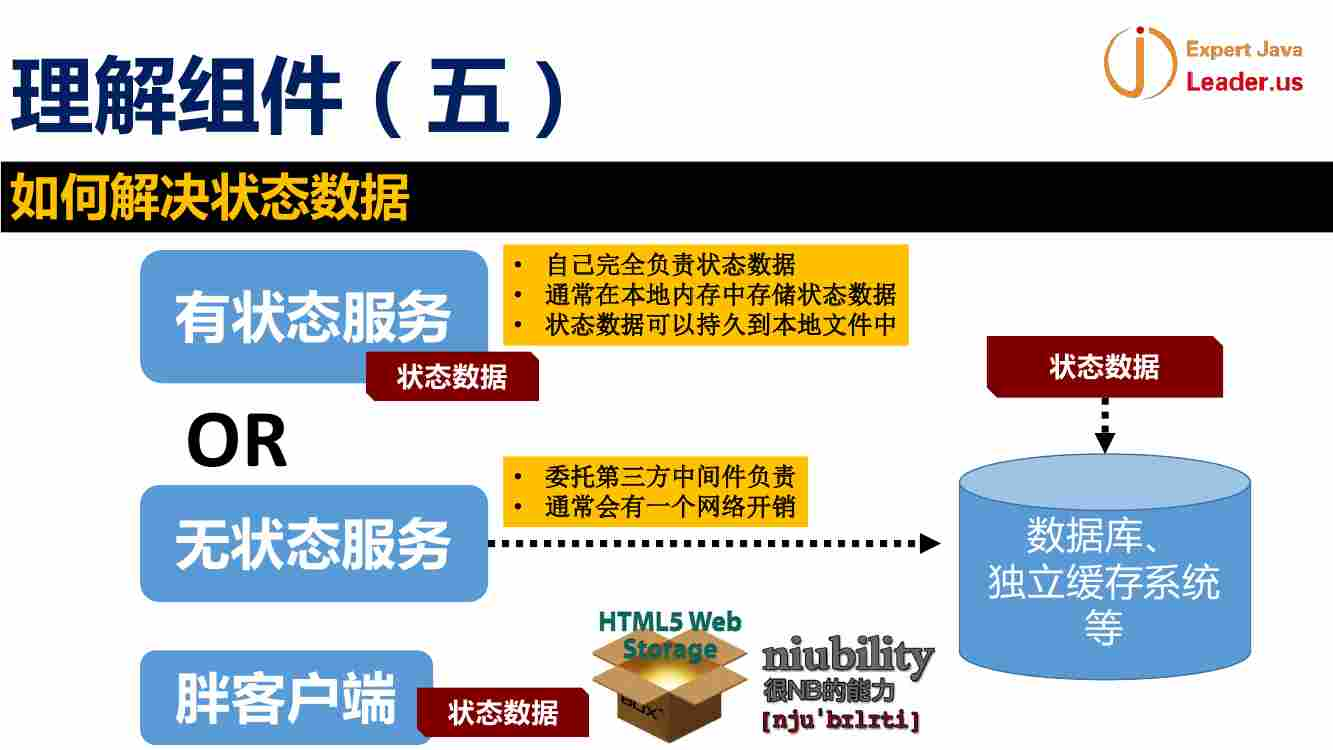
会话在软件中用的非常多 会话：一个代表用户的形成 Socket过程 HTTP服务周期 都是会话

上面是电商过程

用户说的历史表单数据 也是会话数据

购物车 浏览过的商品信息 和 分析出来的行为数据 ------ 为了更好地体验这样一个功能 及时数据和历史数据 都包含这里面

所以状态数据非常重要 组件有状态是必然的



有状态服务 服务本身保存了数据 就是表示本身就靠一个有状态服务去维持

状态数据怎么保存？

本地存储 或者HashMap 或者本地文件 存储不同用户的不同数据

有状态服务自己来维护本身的数据

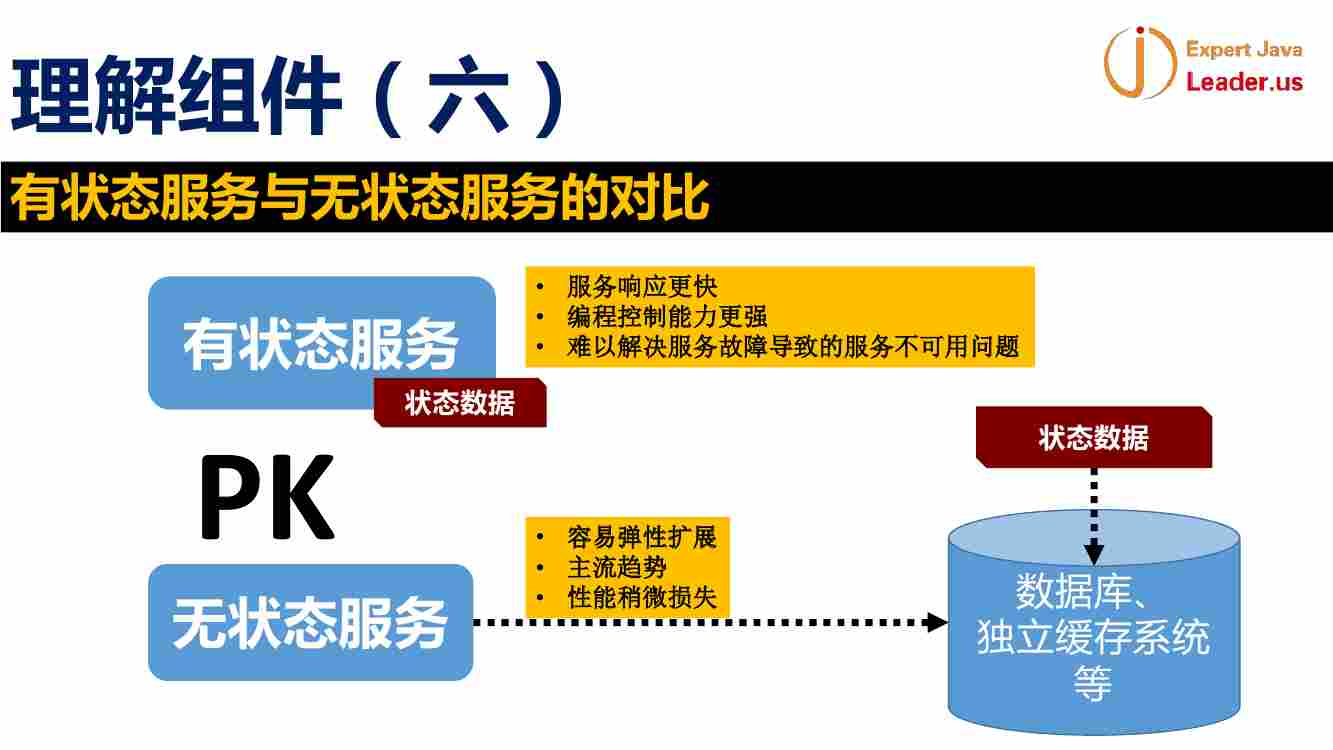
比如tomcatManager 保存了不同的在线用户的信息

无状态的组件 不包括状态信息 只提供业务的处理 数据第三方处理 --- 比如 数据库存储了用户的信息 用户需要访问数据的时候 给出一个表示用户ID的键值 去查数据库中的用户信息 委托了第三方中间件 不属于服务本身 是网络传输 有网络开销的

还有一种是胖客户端 包含了状态数据 也非常重要 –html5重视了这个问题 webstorage 比Cookie更强

很多数据需要在客户端展示 这样展示更快 并且离线可以做一些事情

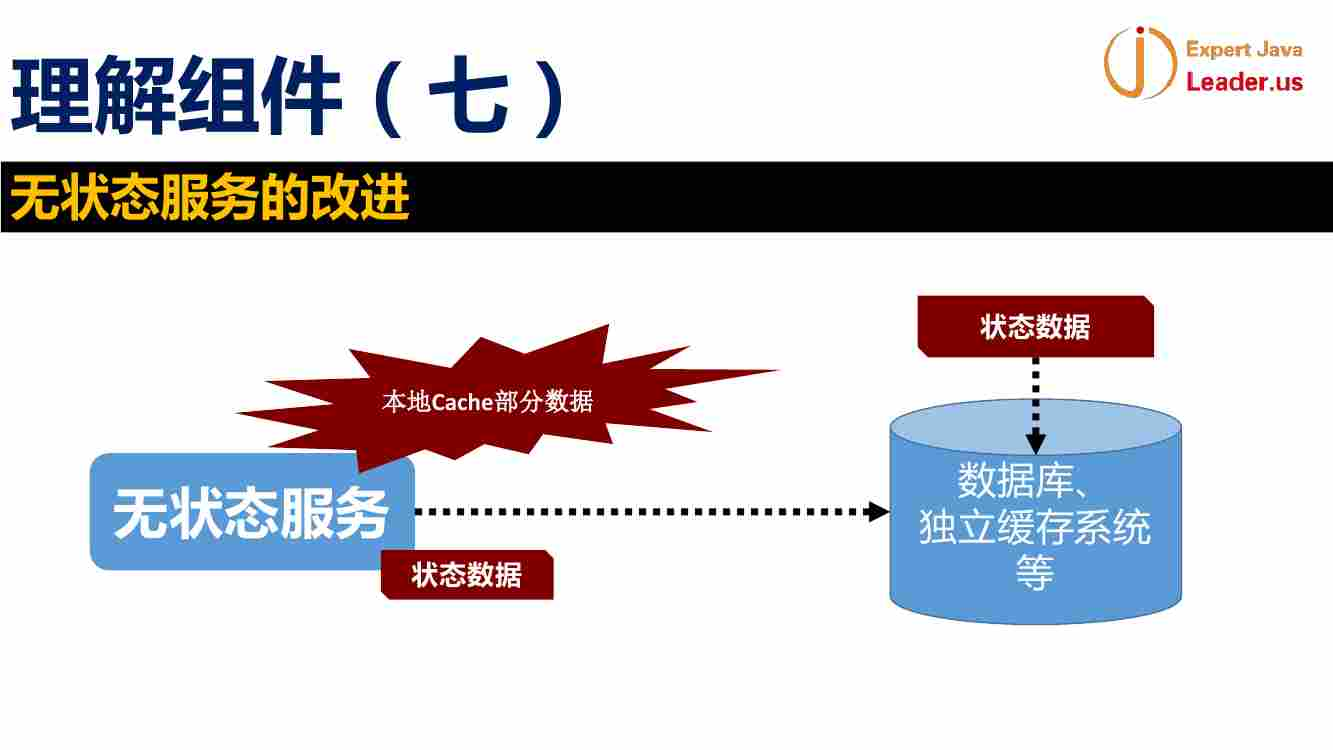
有了html的web storage 很多东西就可以直接在客户端完成



有状态服务 状态数据在内存中 和服务捆绑响应更快 ----- 对编程能力要求高 ---- 既要维护服务的状态数据 还要 提供服务

问题就是 服务有问题 就会产生单点问题 有问题 很难做水平扩展 这种情况 就产生了并行扩展的问题

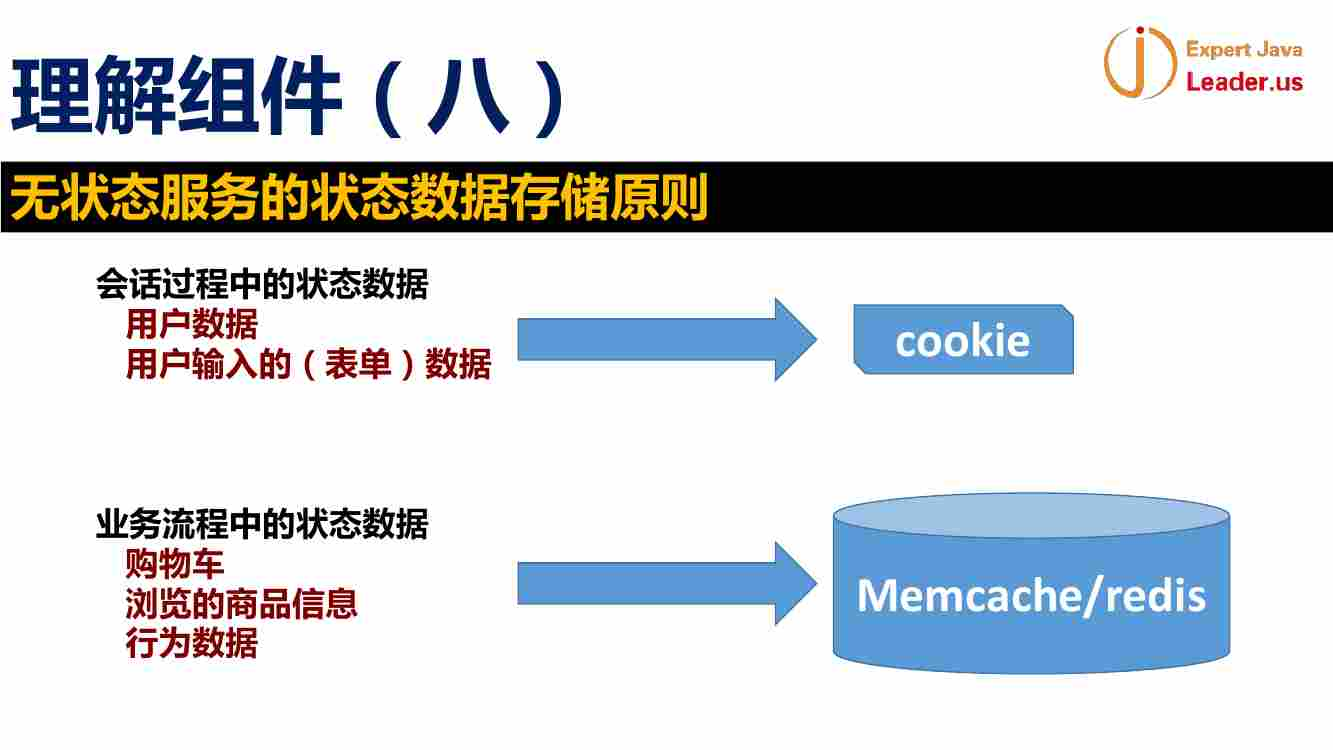
但是 无状态服务器 可以弹性扩展 是主流的方式 但是性能有损失 有网络开销



对性能要求高的无状态服务 ----- 本地Cache部分数据（这个就是有状态服务的一个特点） -----Cache完成之后 如果没找到数据 就去第三方数据库 或者缓存系统进行查找

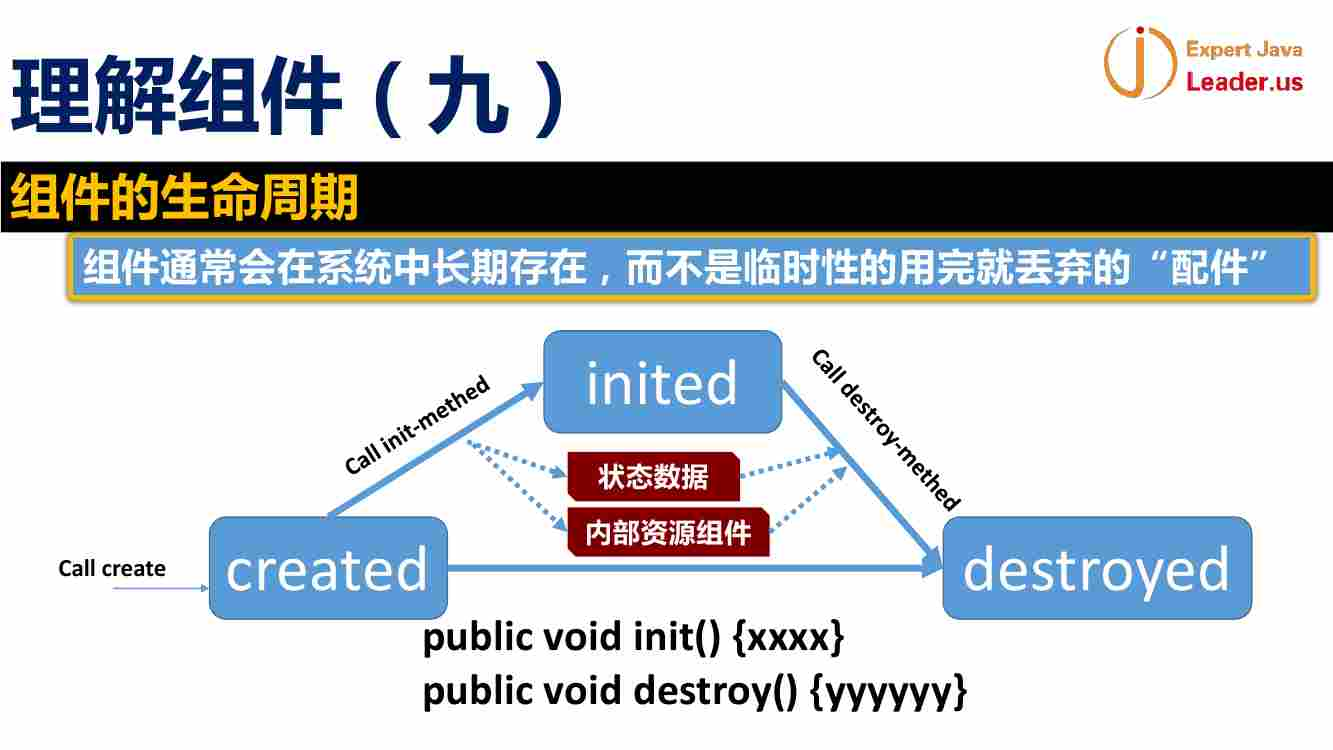
很多都这样做 这是一种柔和状态 柔和了有状态 和无服务的状态

本地无数据也没有影响 本地其实还是一个无状态的服务



用户中的信息在Cookie中 小数据能放cookie中

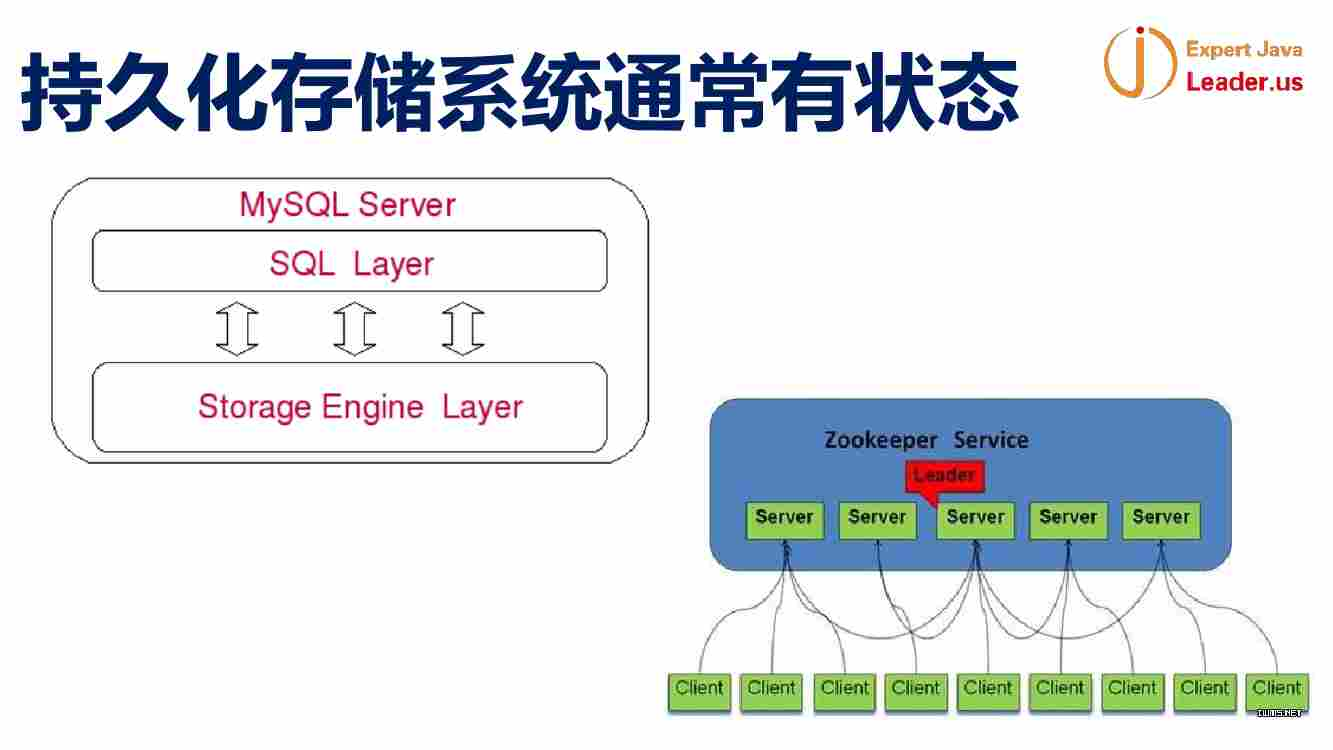
对于一些数据中可以放到缓存 持久化或者临时缓存



组件自服务之前 有一个初始化过程 在不提供服务之后 有一个destroyed的过程 所以 组件 有一个init方法 负责加载状态数据 内部资源组件的初始化 init就是生命周期中的重要方法 之后 提供服务 最后 就是destroy方法 销毁状态数据 ----- 有可能持久化到磁盘中 ---- 比如 有socket连接的 关闭socket链接 数据库链接 所以 组件生命周期 固化为 create inited desctroyed三个框框 完成一个生命周期

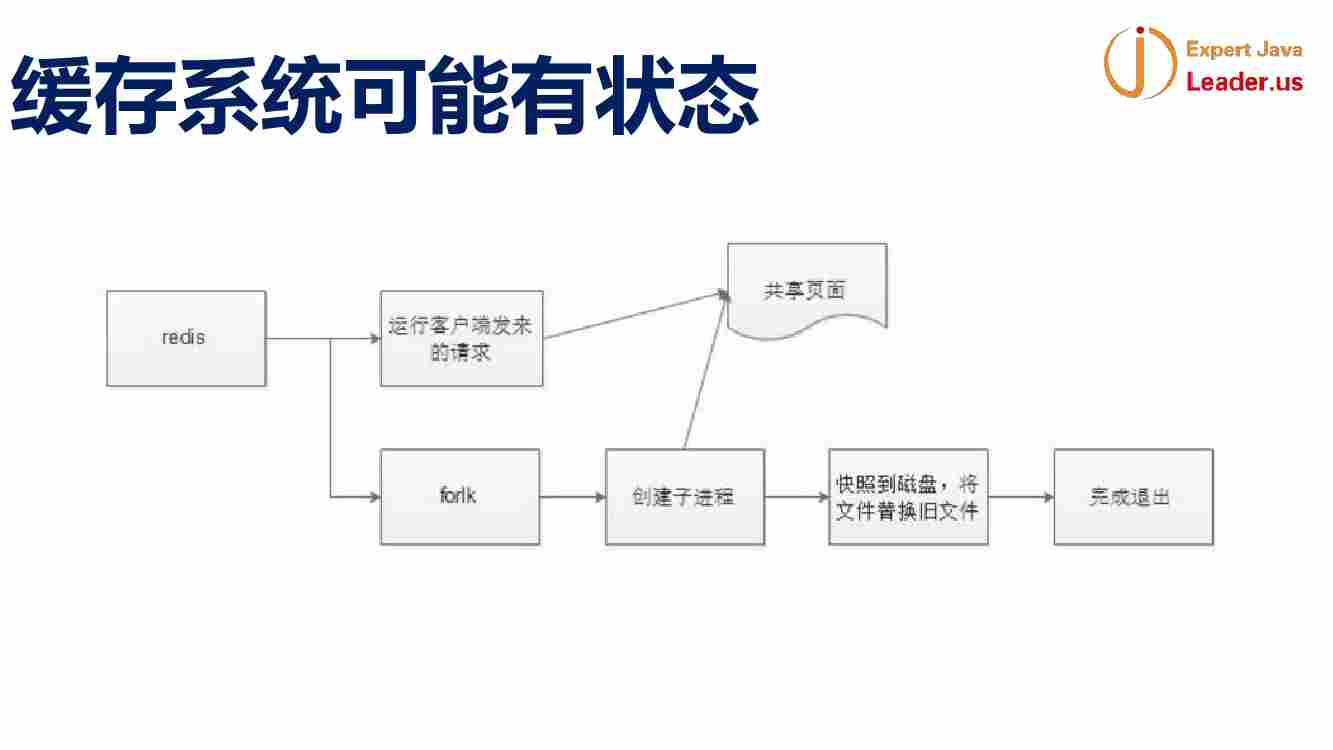
比如 tomcat下的web程序 一般可以重新加载 就是丢弃掉 再加载 ------ 这种情况 数据库连接不关闭 就导致资源消耗 分配了jvm内存 没回收 也会JVM就有一个消耗 ---- 分配了一些数据之后 不用了 就应该destroy掉

因为你的程序不可能都是单进程 一个劲的重启的 而是 要在容器中被不断重启

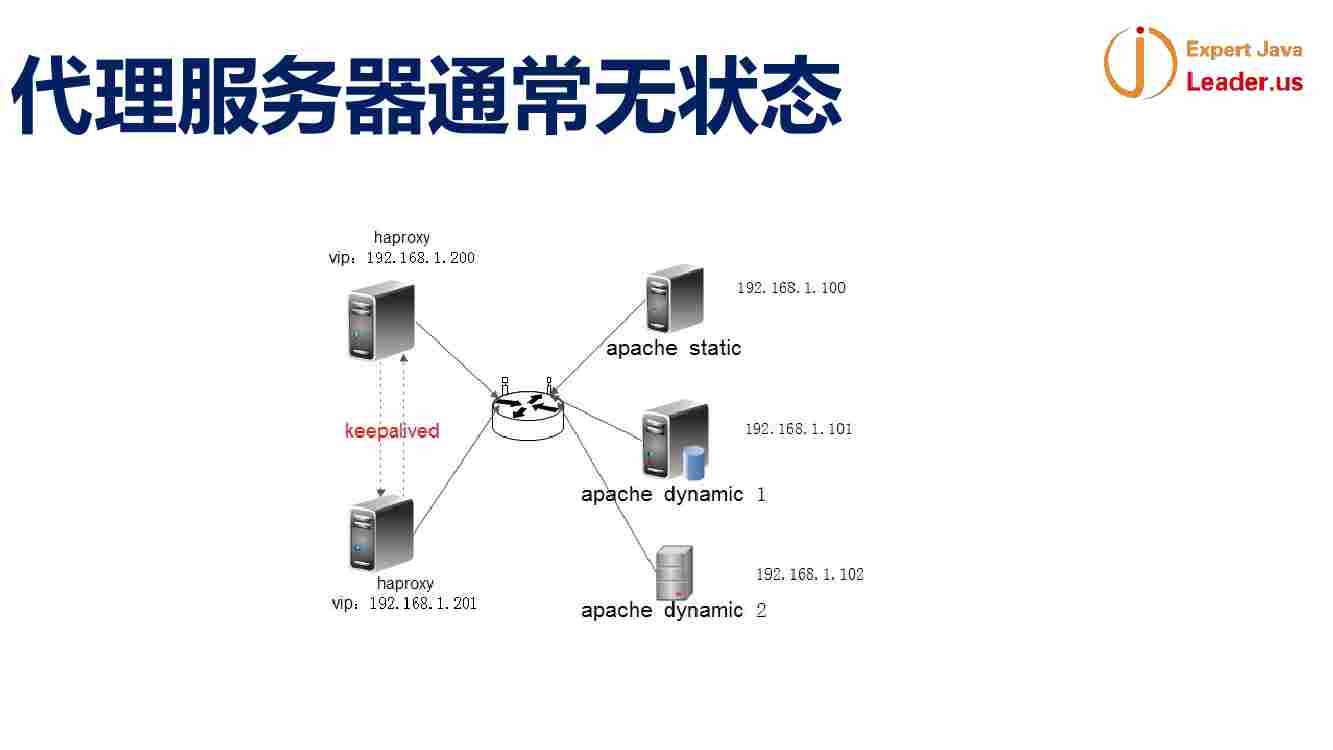


数据库就是有典型状态的系统 所以 数据库很难水平扩展

Zk有状态 做了很多简化 一般是一个集群 但是 规模也不是很大 因为一般情况下 要保持信息的同步 数据存储系统 持久换存储系统一定是有状态的



Redis是有状态的 ---- 可以把数据dump到磁盘中 ------ 缓存系统可能有状态



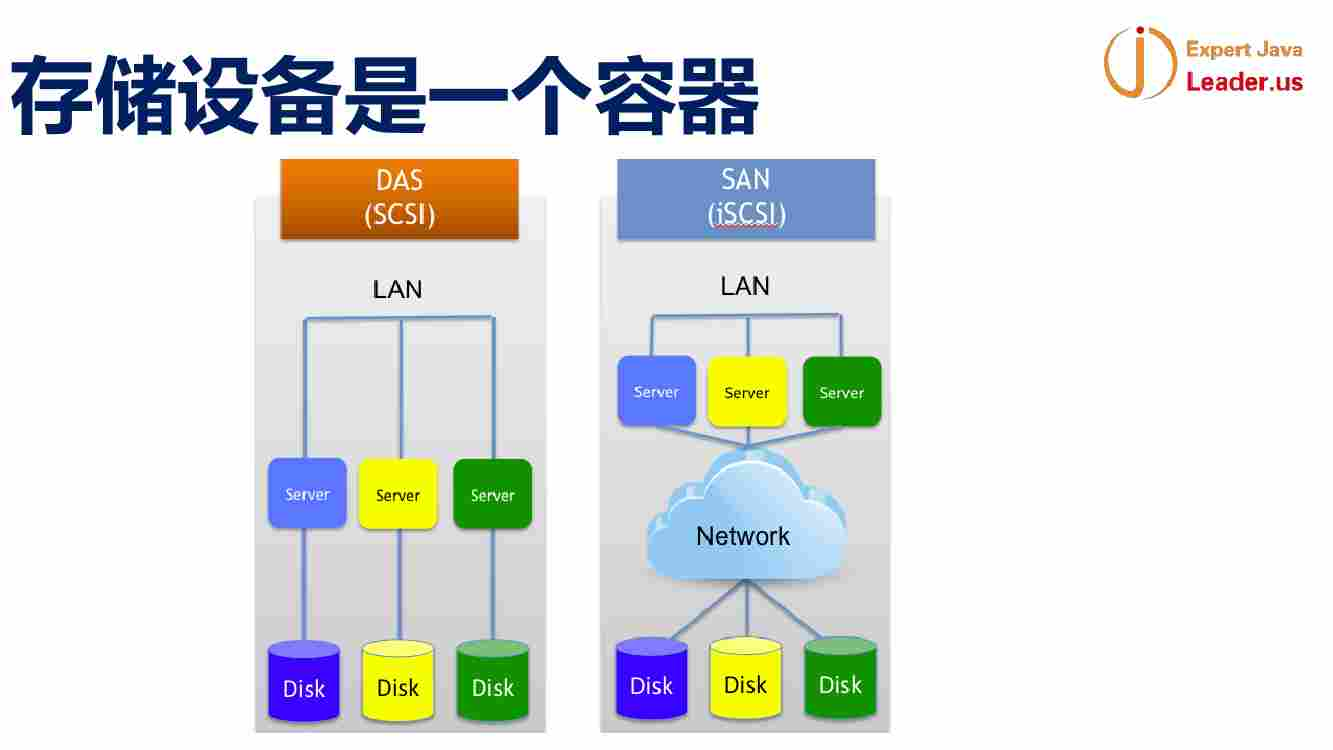
代理服务器一般是无状态的 只负责转发 ------ 不保存中间转发的数据

=========================================================================



负责装配组件的通用矿建   
创建组件 装配组件 负责组件的生命周期

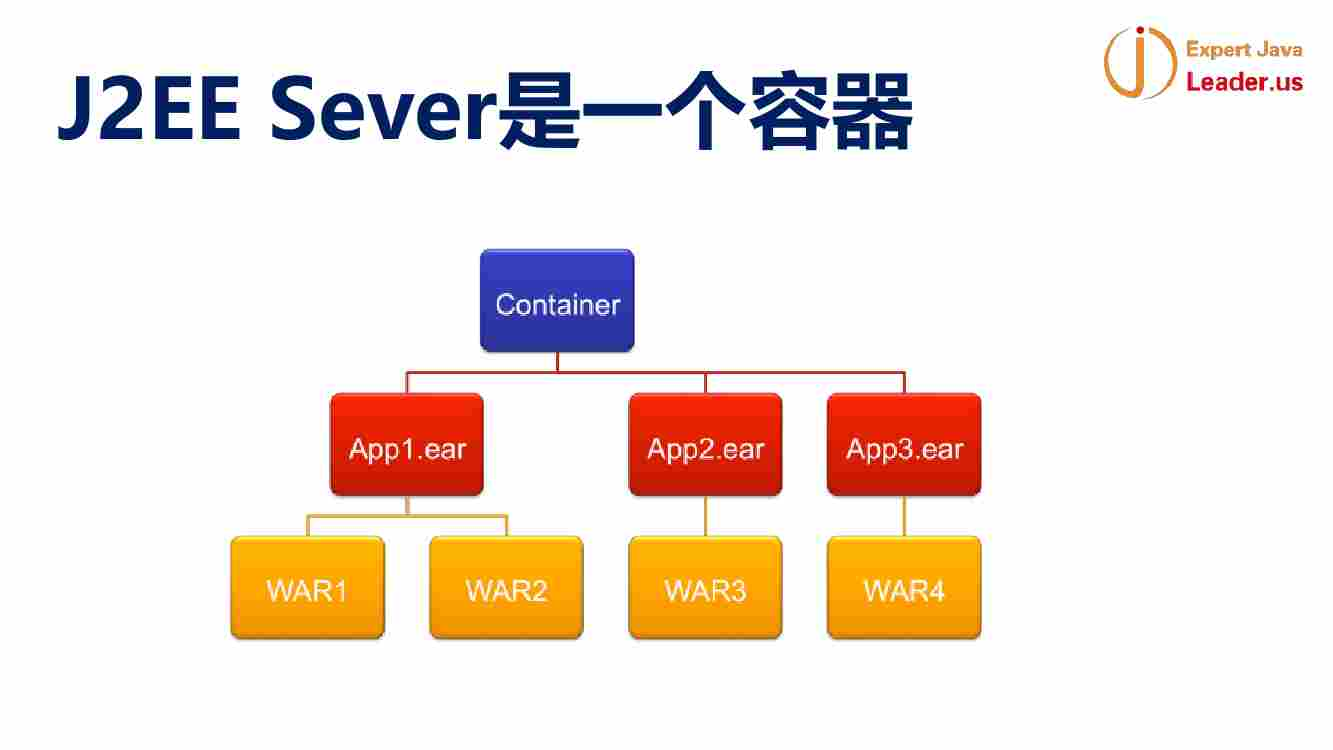
钝化组件在EJB提出来的 就是资源不够的情况下 可以把组件停用 保存到磁盘中 ---- 保存到磁盘中 就需要序列化和反序列化的过程 ---- 所以 组件通常要实现一个可序列化的接口 可以用来重新加载



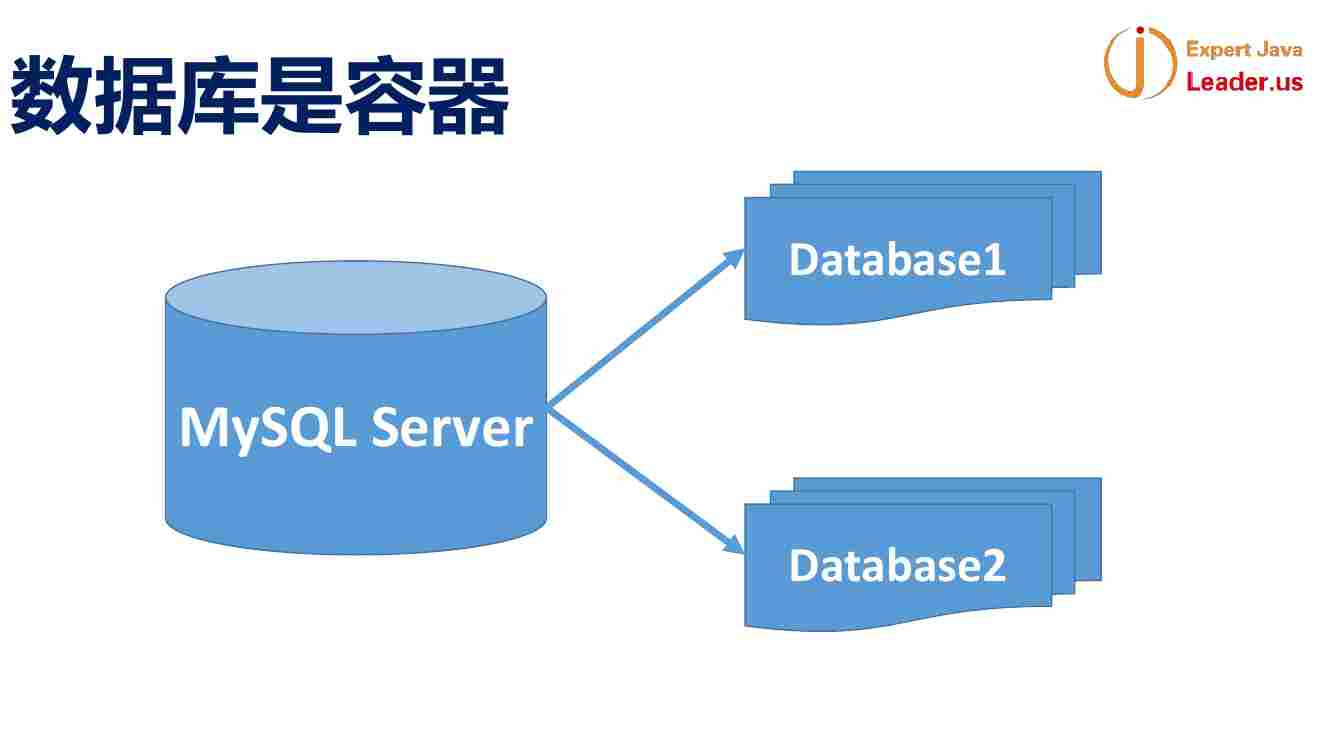
Server 就是容器 Disk就是一个组件 Server负责Disk的可视化 分配 管理



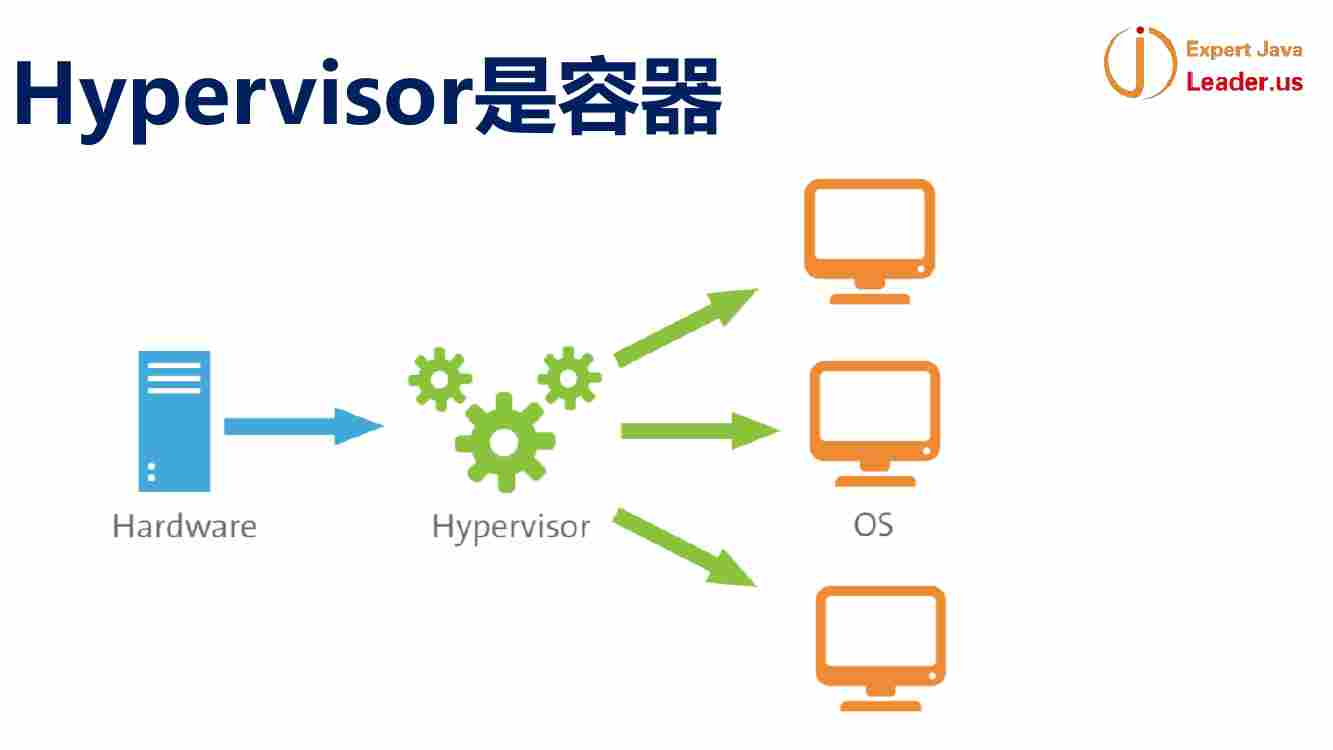
里面的进程 程序 资源都是组件 负责进程的创建 调度 销毁 所以OS也是一个容器



管理war ejb程序 等等 ---- 这样 tomcat服务器 j2ee服务器 都是容器 我们的war就是组件



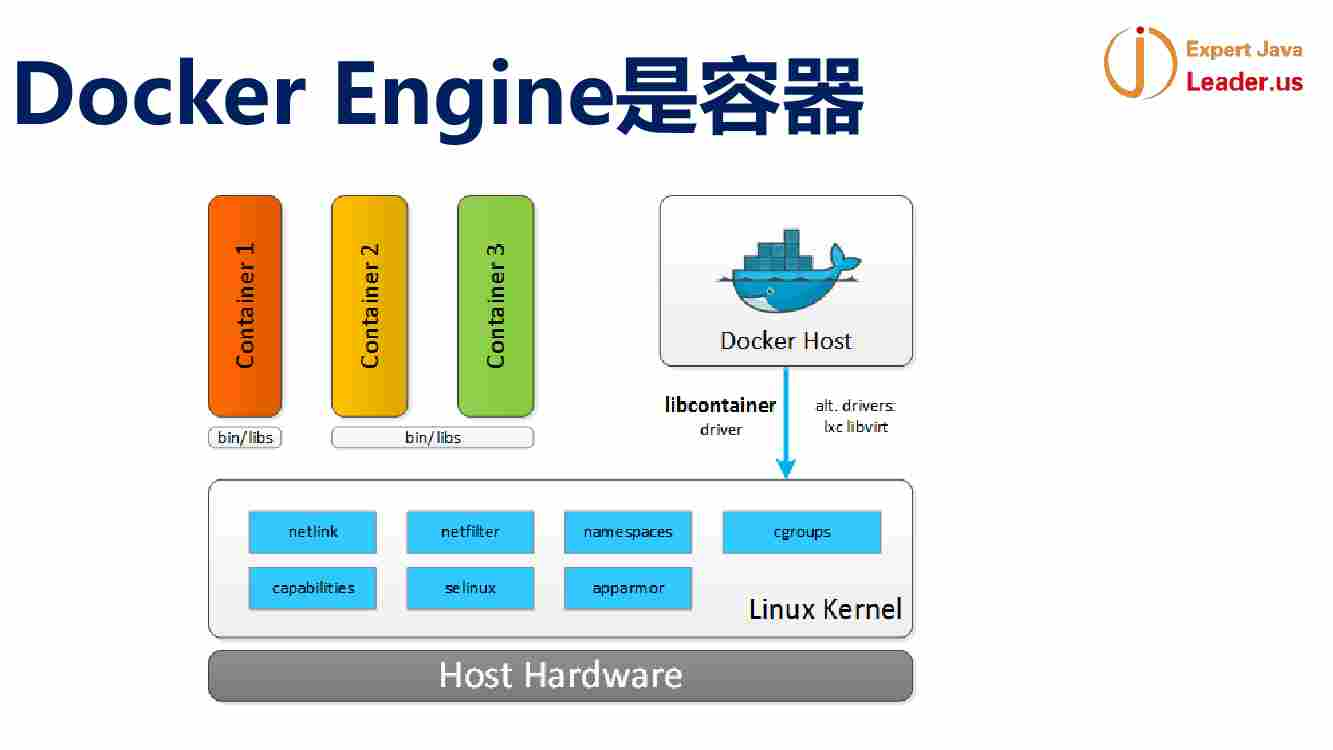
数据库每一个db就是单独的组件 数据负责db的创建 管理 和 维护 访问



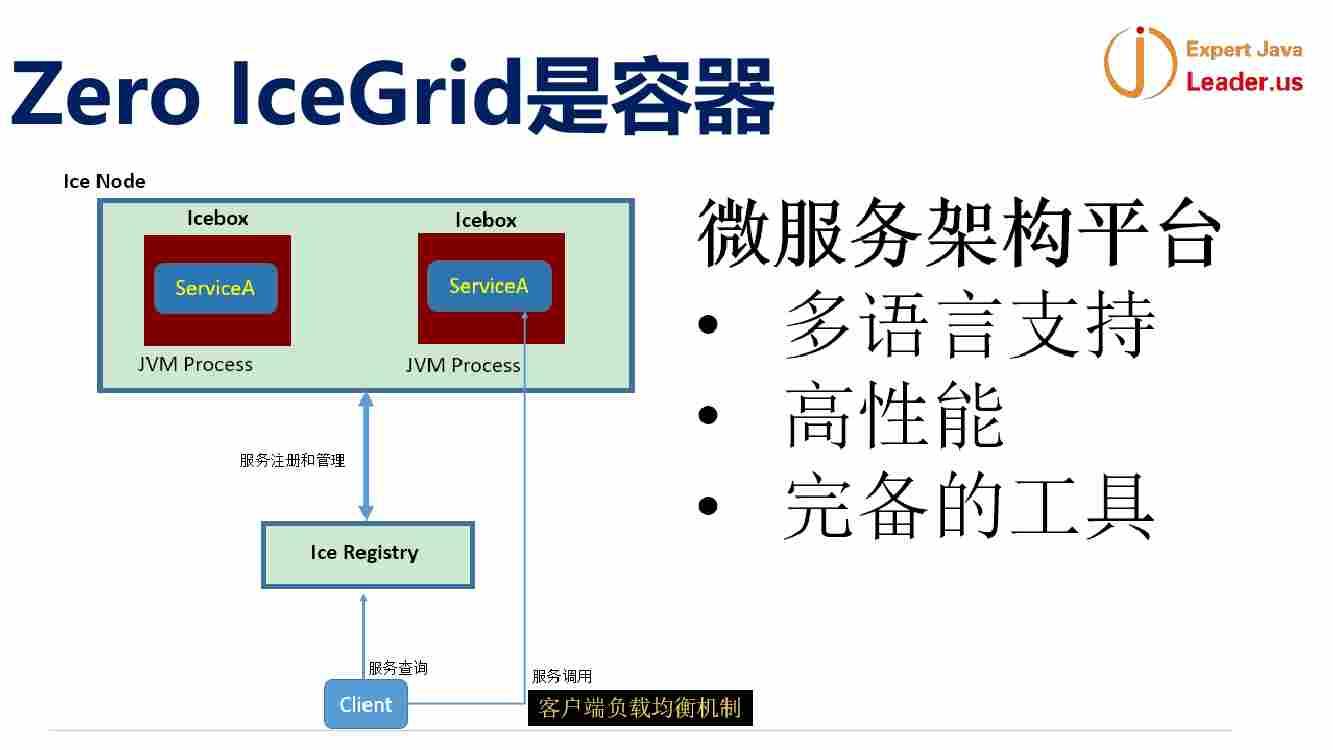
负责在物理机上提供虚拟化OS

组件就是虚拟的OS 给OS分配资源 包括动画 动画就是快照 Hyoervisor就是容器 每一个创建出来的OS就是组件

【说自己是容器 还是组件 都是相对的 容器对包含的组件进行管理 管理组件的创建 维护 和销毁 被管理的就是组件】

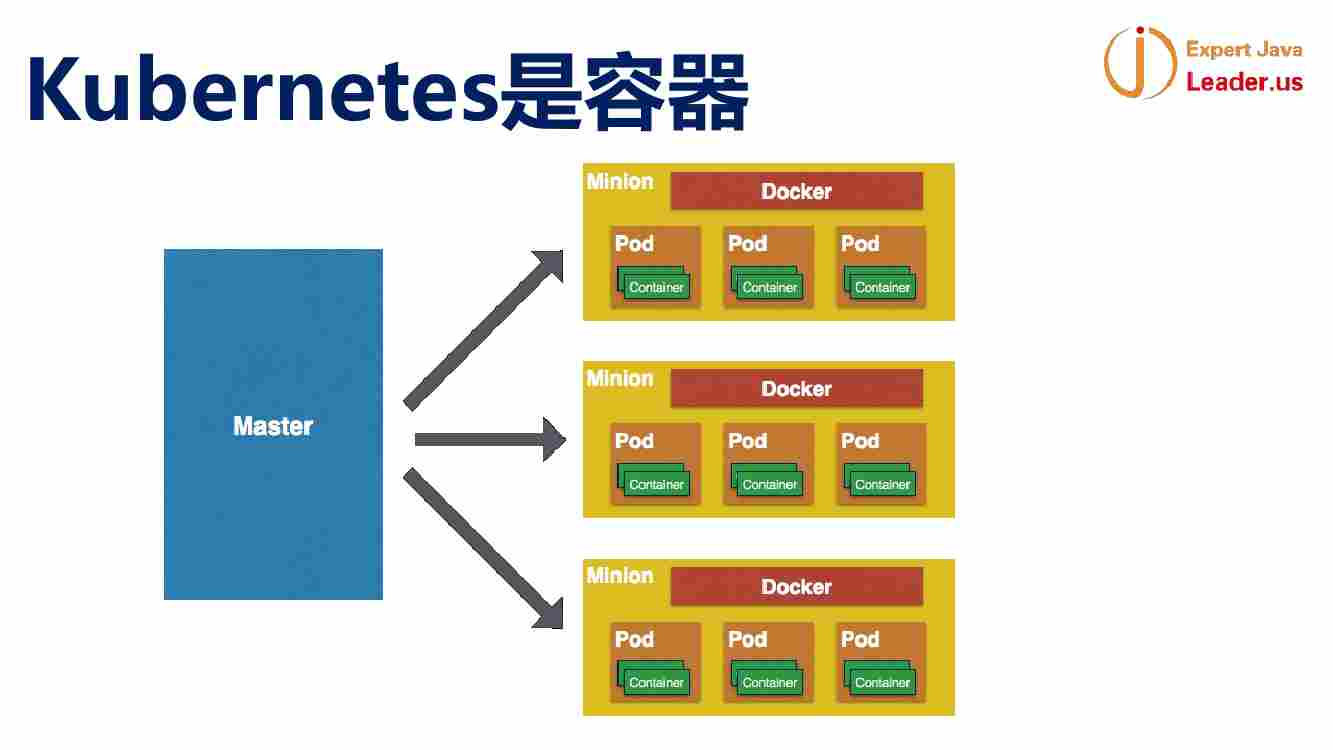


Docker也是一个容器 里面真正的是Container是组件 Docker引擎本身是容器 创建 启动 Container



ICeGrid是容器 整体上 ICEnode是一个容器 里面的每一个Icebox是一个组件

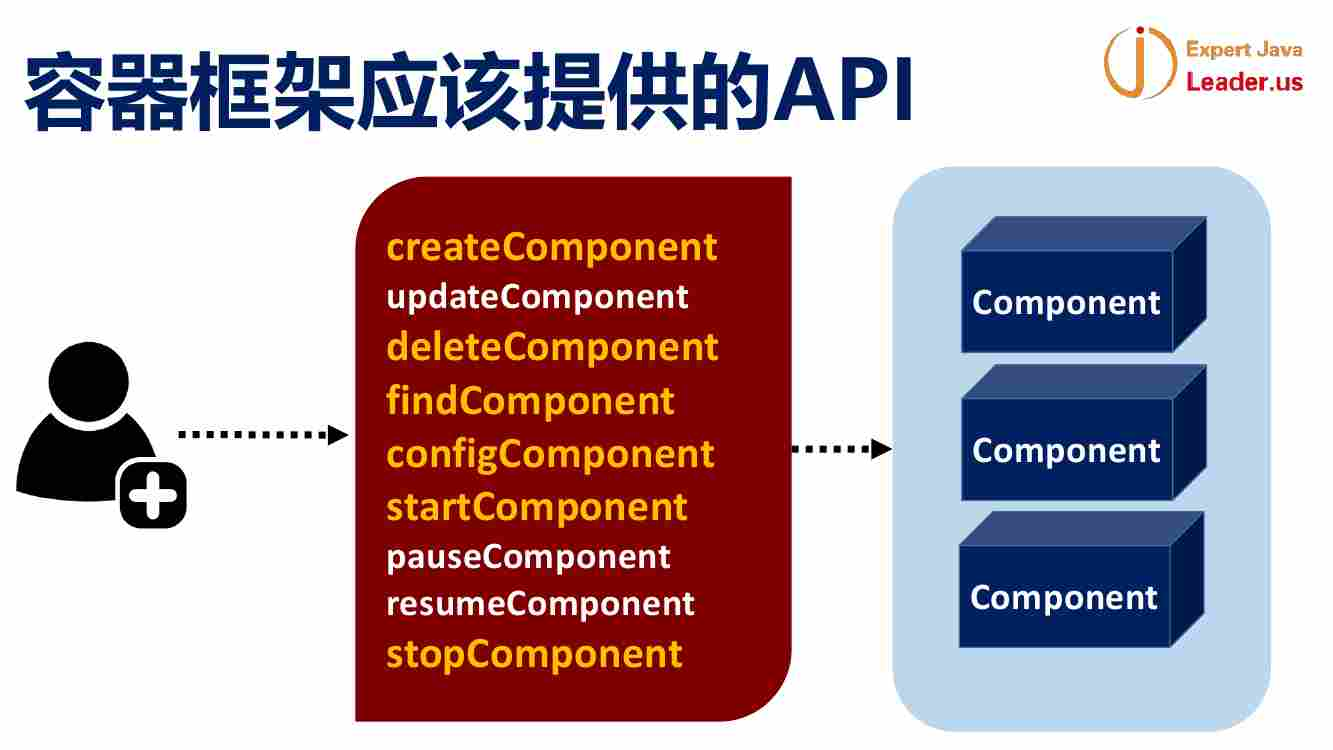
Icenode负责icebox的创建 启动 和 维护



这个也是容器 每一个组件上面 有一个pod pod就是里面的组件



总结 托管组件的容器 才是架构是的工作 也就是 你要设计这样的一个容器 很方便装配组件 调整组建 高技术含量



**设计一个容器** 容器应该提供哪些API?

有创建组件 createComponent ----- 创建组件

updateComponent ---- 让组件的配置重新生效

deleteComponet ---- 删除组件

findComponent --- 查找组件 按照名字去查找 或者按照类型去查找组件 --- 为了装配 A要依赖B ---查找组件 就是典型的命名服务 很多NamingService 很多就是查找组件的过程 找到需要的组件

configComponent --- 配置组件 要给容器配置哪些参数

StartComponent ---- 有生命周期的组件 要能够生命周期的原理

pauseComponent

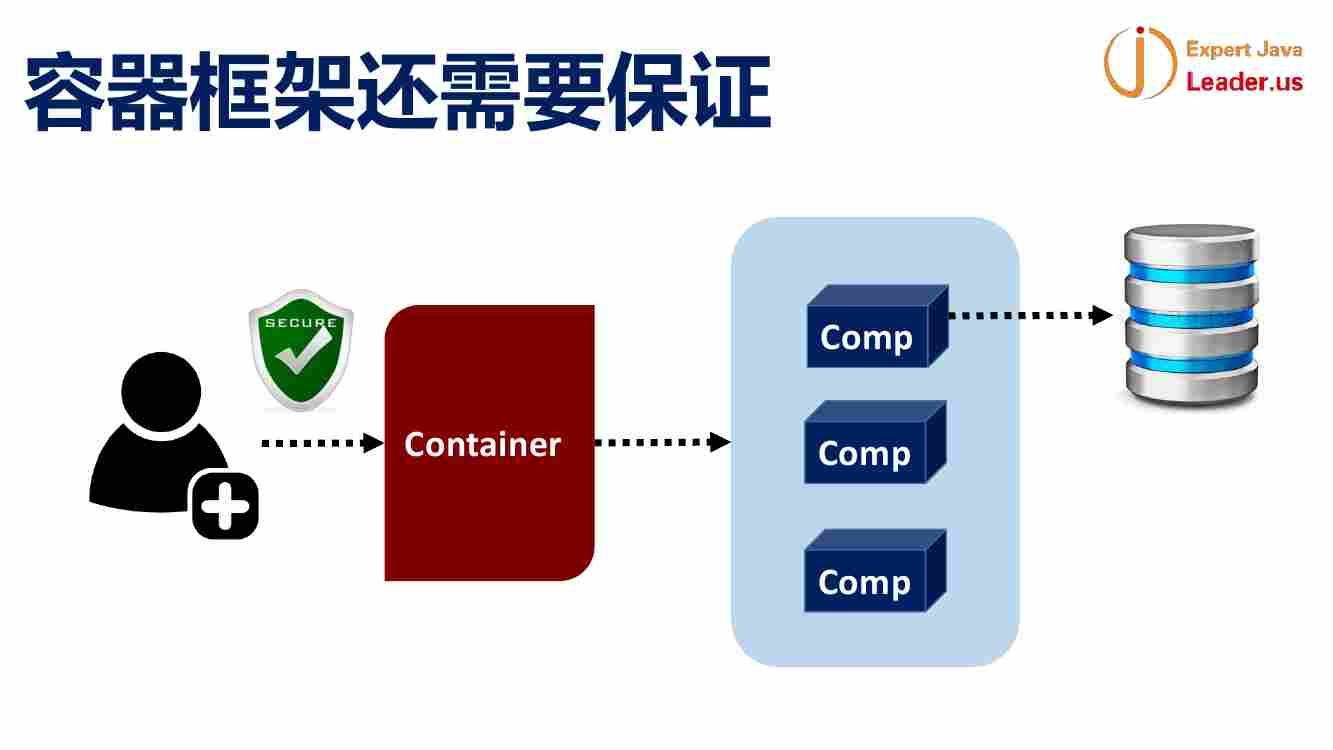
resumeComponent

stopComponent

容器本质就应该提供这些功能 所以 再学Docker的时候 就应该知道 创建Container Start Stop Find等等

OpenStack也是这样的

容器就是创建 并且管理Component的实体 spring就完成了这些功能



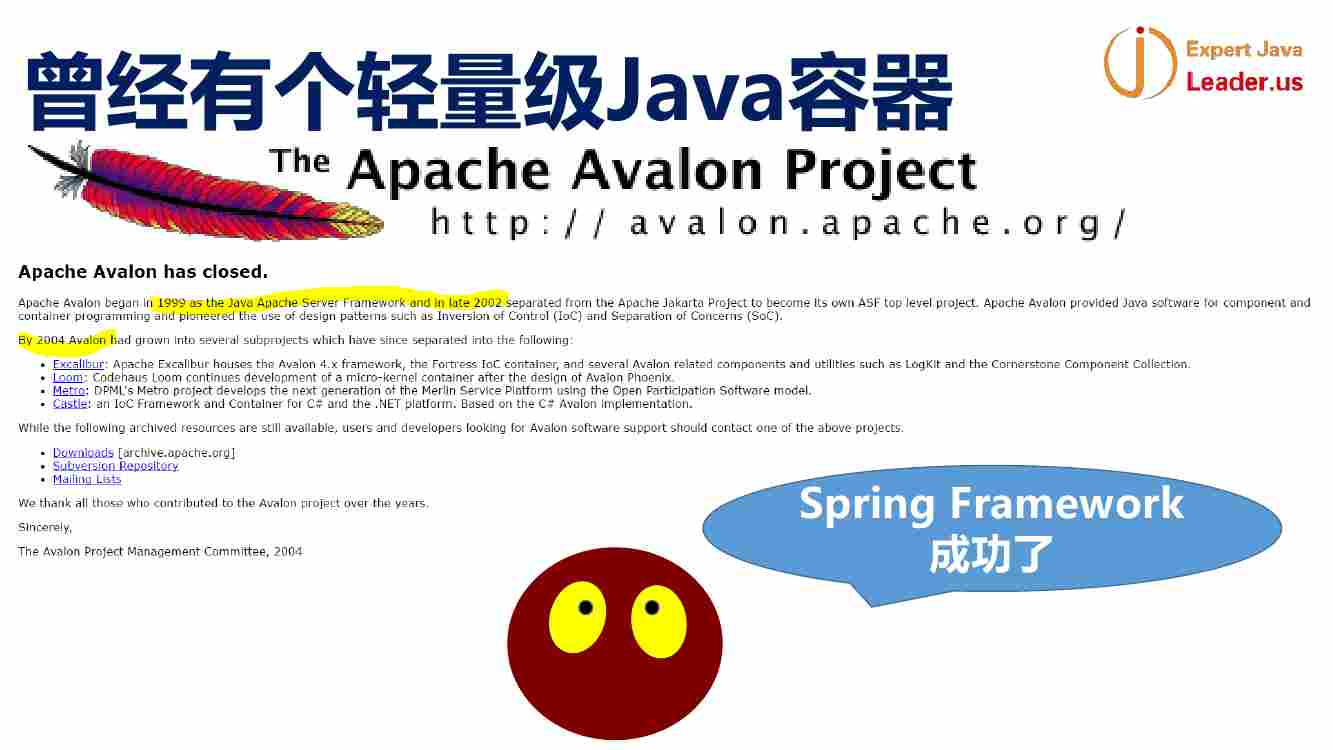
容器还提供了安全性 也就是 你访问虚机 创建虚机 有没有权限 但是 分成两种 一种是面向编程的 ---这个没什么要求

但是 如果你的容器是针对终端用户使用的 操作系统要登录 等等 有访问某个进程的权限

所以 整个容器有安全性要求

也就是说 组件不是自己创建出来的 是通过容器 创建管理的

组件的很多信息 放到持久化数据库中进行管理



Spring之前 有一个轻量级的Java容器 ---- avalon 这个历史很久 比spring早很多 99年就开始了 2002剥离出来 成立了顶级项目

做的就是后面spring做的事 但是 模式不同 模式是模仿j2ee --- 必须定义接口 才能成为容器 所以 就是 入侵式的编程 比j2ee简化

当时 Apache的项目到3.0都很少 这个项目到了4.0 但是最后还是被关闭了 04年关闭了

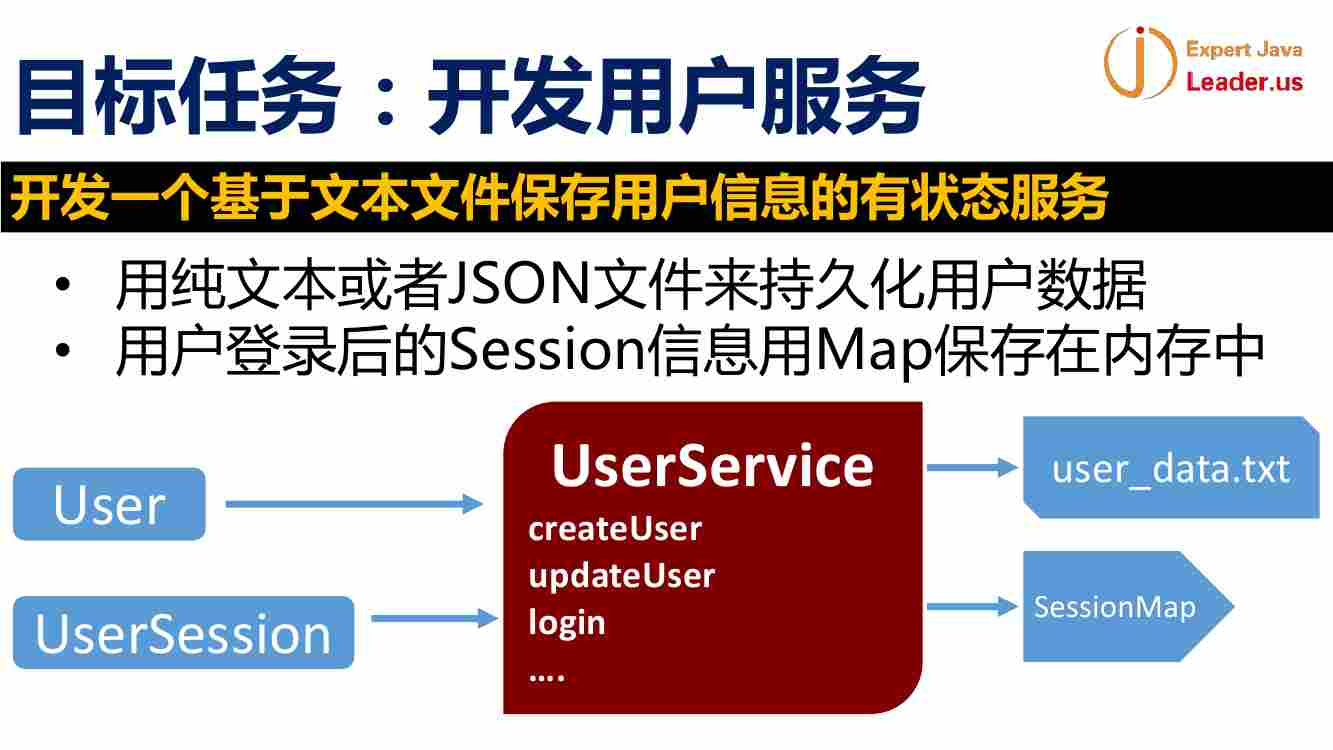
是因为spring成功了 avalon的设计过时了

技术落后了 再改进也很难有发展

Spring做了一个容器框架 里面没有任何的侵入性

没有容器的情况下 编程很麻烦





这个是作业 开发一个用户服务 ---- 一个文本文件的保存用户信息的有状态的服务

文本文件保存用户的信息

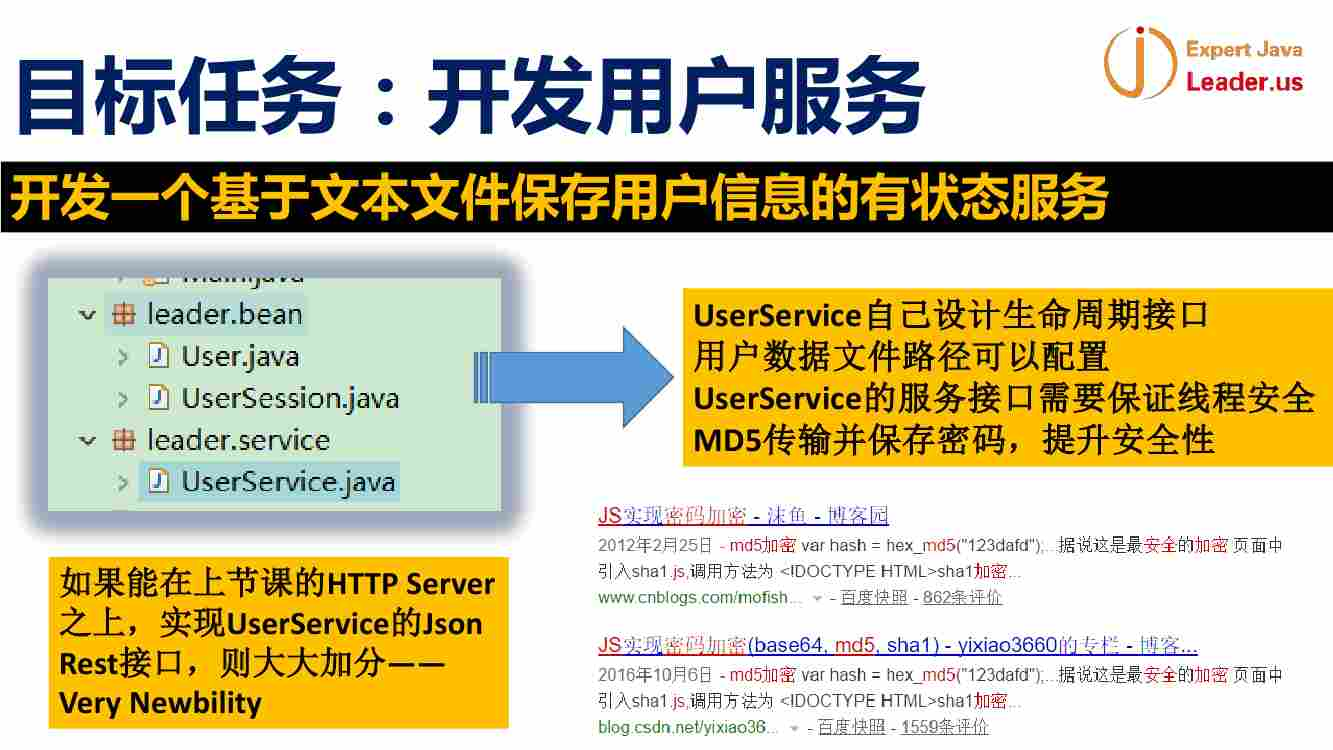
进程启动之后 从纯文本或者json中持久化用户数据 用哪个 看你的熟练程度

用户登陆的session信息持久化到map中 有User对象 UserSession

UserService有createUser updateUser login等方法 数据持久在user\_data.txt

Session信息方到SessionMap中 【考虑查查 http session存储的都是什么数据 作用是什么】

整个过程中 包括配置 装配 状态维护 这个过程 全部自己实现



这个里面是定义的class 代码中直接使用 ------

如果用在HTTPServer上面 就非常好 没有http server 就简单模拟就可以了

保存的密码 实际上是MD5加密的

