//[toc]

讨论课1：Servlet的并发机制

1.1 Servlet简介

Servlet是用来实现服务器端表现层的Java程序

Servlet的功能:

（1）产生客户端界面；

（2）数据转换，生成客户端能识别的数据（比如把JAVA的数组转成JSON或XML格式的数据）

1.2 Servlet接口

Servlet运行在Servlet容器中，其生命周期由容器来管理。Servlet的生命周期通过javax.servlet.Servlet接口中的以下三个方法来表示：

- Init()：初始化servlet（当客户端第一次请求某个Servlet时，Servlet容器将会实例化出一个servlet对象，并调用init方法初始化对象。当有新的客户端请求该Servlet时，不会再实例化该Servlet类。）
- Service()：执行实际任务的主要方法。每当请求Servlet时，Servlet容器就会调用这个方法
- Destory()：当要销毁Servlet时，Servlet容器就会调用这个方法

1.3 Servlet容器

Servlet容器是一个应用程序，主要负责管理servlet

Servlet容器的作用：

- 支持网络
- 管理Servlet的全生命周期（创建、运行、摧毁）
- 多线程的支持

1.4 Servlet容器工作流程

当Servlet容器接受到http请求信息时：

1.分析HTTP请求信息，组装成HttpServletRequest对象

2.创建新的HttpServletResponse对象

3.Servlet容器通过调度线程从线程池中调用一个线程

4.Servlet容器根据路由配置，搜索相应的Servlet实例，将HttpRequest对象和HttpResponse对象传给该Servlet实例

5.调用Servlet对象的service()方法

1.5 Servlet如何处理多用户多线程访问？（讨论课题目）

（1）Servlet收到客户端的具体请求，servlet首先会把请求发送给容器，由容器进行统一的请求派发，容器会从线程池中选取一个工作主线程；

（2）容器器把请求派发给该线程，由该线程的执行servlet的service方法；

（3）这个线程正在执行的时候，servlet容器又收到另一个请求，调度器会从线程池中选取另外一个工作主线程来服务这个新的请求，容器并不关心请求访问的是同一个servlet还是另外一个servlet；

（4）容器收到同一个servlet的多个请求时，这个servlet的service方法将会在多线程中并发执行；

（5）线程使用完之后，就会把线程放回线程池中，如果说线程池中的线程都在进行服务，有新的请求，一般情况下这些请求会做排队处理，servlet容器也能配置servlet的一个最大的请求排队数量，假如超过这个数量，servlet就会拒绝相应的servlet请求。

1.6 Servlet并发处理的优点

1. Servlet单实例，减少了产生servlet的开销
2. 通过线程池来响应多个请求，提高了请求的响应时间
3. 每一个请求由ServletRequest对象来接受请求，由ServletResponse对象来响应该请求；

1.7 Servlet并发处理的特点

单实例多线程：一个Servlet类只实例化一个对象（在第一次被调用时），多个线程共享这个对象。

线程池管理多线程：

1.8 Servlet并发机制的缺点：多线程同步问题

安全性问题：Servlet本身是单实例的，这样当有多个用户同时访问某个Servlet时，当访问该唯一的Servlet实例中的成员变量，如果对成员变量进行写入操作，那就会导致Servlet的多线程安全性问题，即数据不一致。

什么是ServletRequest

•ServletRequest代表来自客户端的请求。当Servlet容器接收到客户端的要求访问特定Servlet的请求时，容器先解析客户端的原始请求数据，把它包装成一个ServletRequest对象。

•HttpServletRequest继承自ServletRequest

1.9 为什么并发处理时使用局部变量不会有安全问题？(老师上课提的问题)

系统存在一个主内存，Java中所有实例变量都储存在主存中，对于所有线程都是共享的。
每个线程都有自己的工作内存，工作内存由缓存和**堆栈**两部分组成，缓存中保存的是主存中变量的拷贝，缓存可能并不总和主存同步，也就是缓存中变量的修改可能没有立刻写到主存中；堆栈中保存的是线程的**局部变量**，线程之间无法相互直接访问堆栈中的变量。

Servlet采用多线程来处理多个请求同时访问。依赖于一个线程池来服务请求。线程池实际上是一系列的工作者线程集合。Servlet使用一个调度线程来管理工作者线程。

当容器收到一个Servlet请求，调度线程从线程池中选出一个工作者线程,将请求传递给该工作者线程，该线程有自己的堆栈，因此多个请求同时处理，使用局部变量不会有安全问题。

讨论课2：Maven中多项目的管理方法

2.1 Maven是什么

Maven 是一个纯 Java 开发的开源项目工具，Maven 是专门用于构建和管理Java相关项目的工具。

- 使用标准目录结构和默认生命周期；
- 约定优于配置（开发者不需要创建构建过程本身)；
- 实现高内聚、低耦合(每一个模块独立，模块间相互联系)；
- 是一个构建工具,不仅帮我们自动化构建,还能抽象构建过程,提供构建任务实现.
- 是跨平台的，对外提供一致的操作接口
- 最大化的消除了构建的重复.
- 可以帮助我们标准化构建过程.所有的项目都是简单一致的,简化了学习成本.

不仅是构建工具,它还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.

2.2 Maven的用处

- 相同的项目结构：使用Maven管理的Java 项目都有着相同的项目结构
    1. 有一个pom.xml 用于维护当前项目都用了哪些jar包
    2. 所有的java代码都放在 src/main/java 下面
    3. 所有的测试代码都放在src/test/java 下面
- 统一维护jar包

2.4 POM的基本坐标与标签：

- project：根元素，声明了POM的命名空间
- modelVersion：当前POM模型的版本
- groupId：项目所在组的标识
- artifactId：项目唯一标识（名称）
- version：项目版本号
- packaging：项目的打包方式（默认jar）
- modules：标识子模块相对于父模块的相对路径
- dependencies：依赖

Maven POM

•父(Super)POM

•父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。

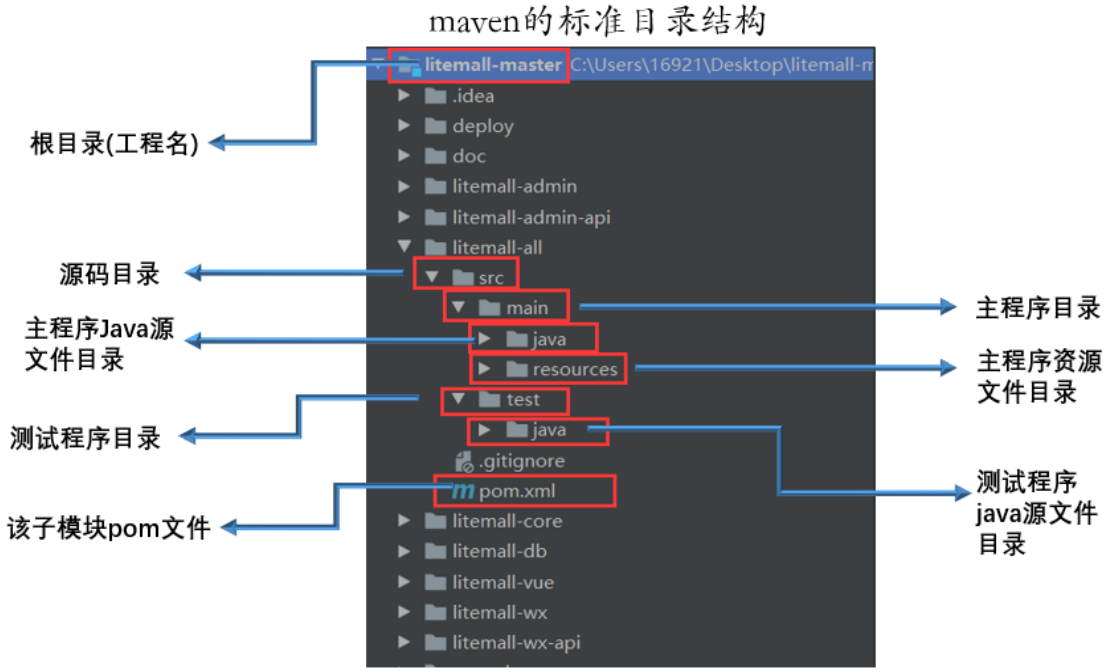
•子POM

•每一个子模块都有一个子POM

2.3 Maven多模块与依赖管理

因为Maven是基于项目对象模型（Project Object Model，POM）的概念运作的，所以Maven的项目都有一个pom.xml文件用来管理项目的依赖以及项目的编译等功能。

POM是Maven工程的基本工作单元，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。



> 执行任务或目标时，Maven 会在当前目录中查找 POM（父），通过读取 POM，获取所需的配置信息，然后执行目标。

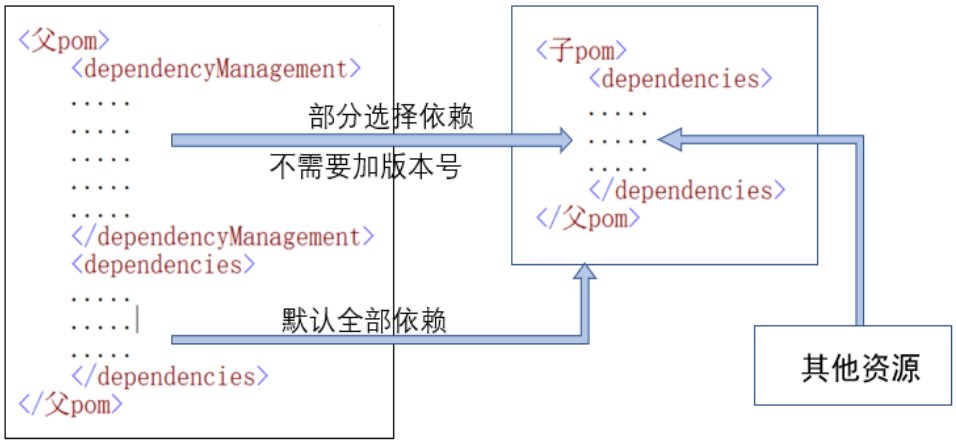
一个大型的项目，为了不显得臃肿而不好开发，常会划分成多个小项目工程。这些子系统并不是以一个工程的形式存在的，而是多个工程并行存在，从而满足项目管理和并行开发的需要。

最外层的依赖文件pom.xml被称为主控pom文件，而内层的结构也可以继续嵌套下去（即在子项目中再划分不同的子模块），子模块也有pom.xml，称为子pom。在文件中，使用标签对各子项目进行管理

```
<modules>
    <module>litemall-core</module>
    <module>litemall-db</module>
    <module>litemall-wx-api</module>
    <module>litemall-admin-api</module>
    <module>litemall-all</module>
</modules>
```

2.5 Maven的继承机制：

在父POM中声明一些配置供子POM继承来实现复用和消除复用；抽出重复的配置，统一依赖管理和插件管理；



标签的存在，是为了解决jar包依赖的版本冲突问题。

只是声明依赖，并不实现引入，因此子项目需要显式的声明需要用的依赖。如果不在子项目中声明依赖，是不会从父项目中继承下来的；子项目中无需指定具体版本

2.6 Maven的依赖传递机制：

依赖的传递取决于相关的**依赖范围**；

A->B->C/D，那么A是否依赖C/D取决于B依赖C/D的范围(compile or test)

Maven工程			依赖范围	对A的可见性
A	B	C	compile	✔
		D	test	✘

如果C对A可见，则当A的POM文件中写入了B的依赖，系统就会自动导入C的依赖

2.8 如何让各模块之间的依赖不复杂？如何管理各模块间的依赖关系？

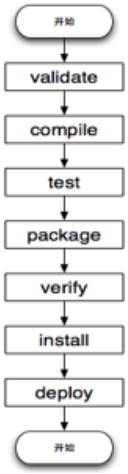
如果每个模块单独一个maven项目，单独构建每个maven项目，然后当你需要的时候在中添加相应项目的依赖。那么，每个模块的构建都要单独进行，如果你的项目有几十个模块，则每次构建项目将需要运行大量的命令。并且项目之间的依赖将影响构建的顺序

解决方案：创建一个总的项目，在父pom文件中定义所有的，并定义所有的版本号，从而一次构建litemall项目，就可以完成所有模块项目的构建。

然后在各子模块中用标签声明继承自父项目，通过module管理复杂的依赖关系

2.9 Maven 生命周期

定义了各个构建环节的执行顺序



阶段	处理	描述
验证validate	验证项目	验证项目是否正确且所有必须信息是可用的
编译 compile	执行编译	源代码编译在此阶段完成
测试 Test	测试	使用适当的单元测试框架(例如JUit)运行测试
包装 package	打包	创建JAR/WAR包，如在pom.xml中定义提及的包
检查 verify	检查	对集成测试的结果进行检查，以保证质量达标
安装 install	安装	安装打包的项目到本地仓库，以供其他项目使用
部署 deploy	部署	拷贝最终的工程包到原车跟仓库中，以共享给其他开发人员和工程

17

三个标准的相互独立的生命周期

- **Clean** 项目构建前的清理工作；
  - pre-clean 执行clean之前的工作
  - clean 移除上一次构建生成的文件
  - post-clean执行clean之后的工作
- **Default** 构建的核心部分,生命周期中最重要的部分，一共包含23个阶段；

  (build) (编译，测试，打包，安装，部署等等)；

  运行任何一个阶段的时候，它前面的所有阶段都会被运行；
- **Site** 生成项目报告，站点，发布站点；
  - pre-site: 执行一些需要在生成站点文档之前完成的工作；
  - site: 生成项目的站点文档；
  - site-delpoy: 将生成的站点文档部署到特定的服务器上（即发布maven站点）；
  -

2.10 生命周期与maven插件机制

- Maven的插件机制完全依赖Maven的生命周期
  - 生命周期的各个阶段仅仅定义了要执行的任务是什么
  - 各个阶段和插件的目标相互对应
  - 相似的目标由特定的插件来完成

可以将目标看作"调用插件功能的命令"

生命周期阶段	插件目标	插件
compile	Compile	Maven-compiler-plugin
test-compile	testCompile	Maven-compile-plugin

2.11 Maven 仓库

Maven 仓库是项目中依赖的第三方库，这个库所在的位置叫做仓库。maven仓库用于管理构件（任何一个依赖、插件、项目构建的输出）。

三种类型

1. 本地仓库；

不会在安装maven后就被创建，在第一次执行maven命令时才被创建；

2. 中央仓库；

Maven社区提供的仓库，包含大量常用的库；包含大多数Java项目依赖的构件；

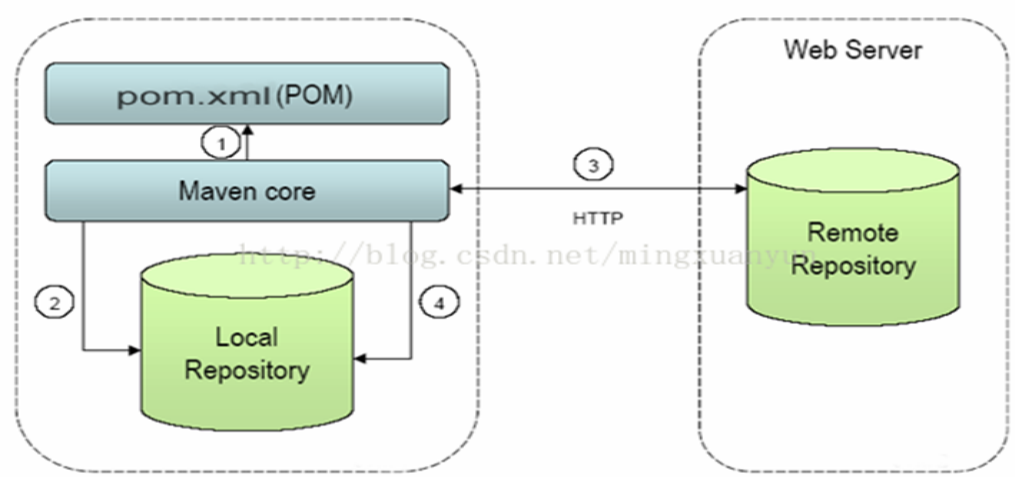
由maven社区管理，无需配置，通过网络访问；

3. 远程仓库；

开发人员自己定制仓库，包含了从中央仓库获取不到的文件

Maven执行构建命令时按以下顺序查找依赖的库：

1. 在本地仓库搜索；
2. 在中央仓库搜索；
3. 在远程仓库中查找并下载到本地仓库；
4. 如果没有设置远程仓库，简单停滞处理并抛出错误；

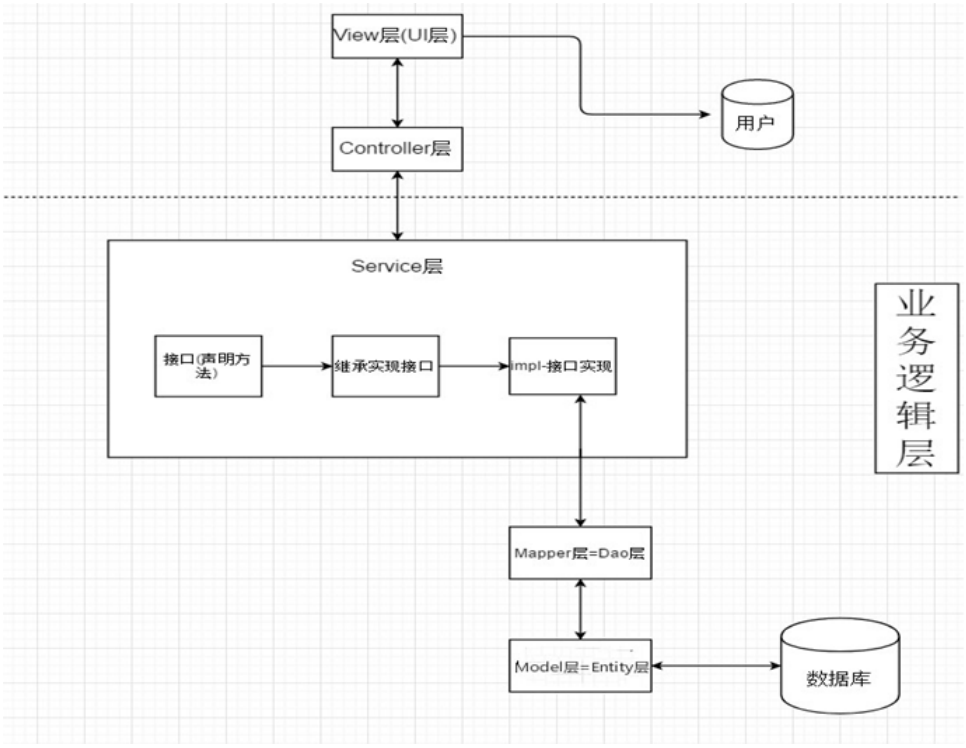


2.11 Maven 多模块项目管理的优点

1. 借助Maven将一个项目拆分成多个工程，相比于Package更便于管理多模块项目，有利于团队间分工合作，模块间可以互通信息。
2. 依赖——继承机制可以降低jar包重复率，节约项目空间。
3. 所有知名框架或第三方工具jar包已经按照统一规范放在了Maven的中央仓库中，jar包不需要去官网下载，不会下载到其他不好的版本，项目更规范。
4. Maven自动将被依赖的jar包导入项目。

讨论课3：讨论Litemall的后端体系结构和模块划分

3.1 springBoot分层：controller、service、dao、entity层



dao层：数据持久层，也称为mapper层。作用：访问数据库，向数据库发送sql语句，完成数据的增删改查。它只是个接口，只有方法名字，具体实现在mapper.xml中。

service层：业务逻辑层。作用：完成功能设计。service层调用dao层接口，接收dao层返回的数据，完成项目的基本功能设计。

> Service层建立在DAO层之上的，建立了DAO层后才可以建立Service层，而Service层是在Controller层之下的，因而Service层应该既调用DAO层的接口，又要提供接口给Controller层的类来进行调用，刚好处于中间层的位置。

controller层：控制层。作用：请求和响应控制。负责前后端交互，接受前端请求，调用service层，接收service层返回的数据，最后返回具体的页面和数据到客户端。

> 面向对象和面向功能的分界线；面向对象到面向功能的数据格式转换；做一些比较繁杂的工作，比如权限。

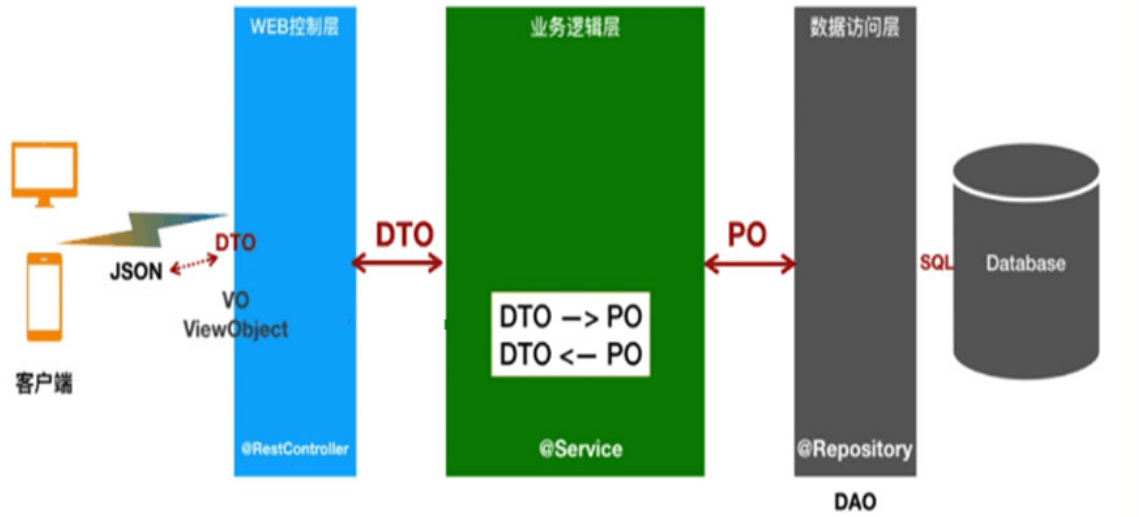
litemall中层与层之间的调用关系：

controller层中类的方法调用service层中类的方法，service层中类的方法调用dao层中类的方法，dao层中的接口方法的实现在mapper.xml中实现。

3.2 实体类：

- **VO**：视图对象，用于view层。作用：把某个指定页面（或组件）用于展示的数据封装起来。
- **DTO**：数据传输对象。将数据封装成普通的POJO，在J2EE多个层次之间传输。
- **PO**：持久化对象：跟持久层的数据结构形成一一对应的映射关系，若持久层是关系型数据库，那么，数据表中的每个字段就对应PO的一个属性。

  成员变量+getter/setter方法→把数据库中字段映射成为对象的属性



3.3 面向接口开发：（将service层分为接口和实现类）

多人分模块开发时，写service层的人将接口定义好提交到Git，其它层的人直接可以调用接口方法，而写service层的人也可以通过实现类写具体方法逻辑。达到多人同时开发。

讨论课4：NGINX服务器

4.1 什么是Nginx

1. Nginx是一个web服务器和反向代理服务器，及电子邮件代理服务器，用于HTTP、HTTPS、SMTP、POP3和IMAP协议。
2. 基于REST架构风格，以统一资源描述符URI或者统一资源定位符URL作为沟通依据.

4.2 为什么要用Nginx

1. 跨平台、配置简单，非阻塞、高并发连接；
2. Nginx具有占用内存少，稳定性高等优势；
3. 内置的健康检查功能, 稳定性高：宕机的概率非常小
4. 节省宽带：可以添加浏览器本地缓存
5. 使用异步逻辑：（从而削减了进程上下文切换的开销，因此并发服务能力更强。）
6. Nginx整体采用模块化设计，有丰富的模块库和第三方模块库，配置灵活。
7. 可以有多台nginx服务器 使用dns做负载均衡,事件驱动
8. Nginx使用epoll事件模型，在Linux系统上运行效率非常高。

> **Nginx性能这么高的原因**：得益于它的事件处理机制：异步非阻塞事件处理机制：运用了epoll模型，提供了一个队列，排队解决。

4.3 Nginx 的采用异步逻辑和一客户一线程有什么区别?

- 异步逻辑：浏览器将请求发送到nginx服务器，它先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了web服务器的压力,一边接收web服务器的返回数据，一边发送给浏览器客户端, 网络依赖性比较低，只要ping通就可以负载均衡。
- 一客户一线程：服务器持续侦听，每接收到一个连接请求，就创建一个新线程进行处理。

异步逻辑削减了进程上下文切换的开销，因此并发服务能力更强。

4.4 什么是epoll事件模型?

假设进程有10万个TCP连接，且只有几百个连接是有事件需要处理的。那么在每一个时刻进程只需要处理这几百个有事件需要处理的连接即可。

> 1、调用epoll_creat函数建立一个epoll对象（一颗红黑树，一个准备就绪list链表）。

> 2、调用epoll_ctl函数把socket放到红黑树上，给内核中断处理程序注册一个回调函数，告诉内核，如果这个句柄的中断到了，就把这个socket放到准备就绪list链表里。

> 3、调用epoll_wait到准备就绪list链表中处理socket，并把数据返回给用户。）

> Note:不需要把全部的连接处理一遍，只需要去list链表里处理socket。

> 获取事件时，操作系统不需要遍历所有连接，提高了程序在高并发中只有少量连接活跃情况下CPU的利用率。

4.5 为什么会需要类似Nginx这一类代理服务器呢?

web应用服务器往往达不到理想中的并发（其原因可能是带宽、系统IO等），进而自然想到通过分布式架构提升并发能力。但又因为公网IP的稀缺性，所以只能采用Nginx反向代理来做类似负载均衡的一些事情。

4.6 Nignx如何做负载均衡调度算法?

1. weight
2. 轮询(默认，常用)：每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除
3. ip_hash（常用）：每个请求中客户端的ip的hash进行匹配，访问到后端服务器。
4. fair：智能调整调度算法，动态的根据后端服务器的请求处理到响应的时间进行均衡分配，响应时间短处理效率高的服务器分配到请求的概率高。
5. url_hash：每个请求中客户端的url的hash进行匹配，访问到后端服务器。

优点：负载均衡主要解决网络拥塞问题，提高服务器响应速度，服务就近提供，达到更好的访问质量，减少后台服务器大并发压力.

4.7 采用ip_hash有什么优点(存在的价值。。。)?

把登录信息保存到了session中，那么跳转到另外一台服务器的时候就需要重新登录了，所以很多时候我们需要一个客户只访问一个服务器，那么就需要用iphash了，iphash的每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

4.8 使用“反向代理服务器”的优点是什么?（具体做法和正向代理区别）+

1. 保证内网的安全，通常将反向代理作为公网访问地址，Web服务器是内网
2. 反向代理+负载均衡，通过反向代理服务器来优化服务器的负载

4.9 Nginx是如何处理一个请求的

1. nginx在启动时，会解析配置文件，得到需要监听的端口与ip地址 （解析文件获ip端口）
2. 在nginx的master进程里面先初始化好这个监控的socket，再进行listen （初始化）
3. fork出多个子进程，子进程会竞争accept新的连接。
4. 客户端向nginx发起连接。
5. 三次握手，建立连接后
6. 子进程accept成功，创建nginx对连接的封装，根据事件调用相应的事件处理模块
7. nginx或客户端来主动关掉连接

4.10 动态资源、静态资源分离

动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来。

根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路。

动态资源、静态资源分离简单的概括是：动态文件与静态文件的分离

4.11 为什么要做动、静分离

1. 后台处理忽略静态文件
2. 对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决（减少后台请求次数），动、静分离将网站静态资源
3. 提高用户访问静态代码的速度，降低对后台应用访问，可以将静态资源放到nginx中，动态资源转发到tomcat服务器中

4.12 列举Nginx和Apache 之间的不同点。

Nginx	Apache
• Nginx是一个基于事件的web服务器 • 所有请求都由一个线程处理 • Nginx避免子进程的概念 • Nginx类似于速度 • Nginx在内存消耗和连接方面比较好 • Nginx在负载均衡方面表现较好 • 对于PHP来说，Nginx可能更可取，因为它支持PHP • Nginx不支持像IBMi和OpenVMS一样的OS。 • Nginx只具有核心功能 • Nginx的性能和可伸缩性不依赖于硬件	• Apache是一个基于流程的服务器 • 单个线程处理单个请求 • Apache是基于子进程的 • Apache类似于功率 • Apache在内存消耗和连接上并没有提高 • 当流量到达进程的极限时，Apache将拒绝新的连接 • Apache支持的PHP、Python、Perl和其他语言使用插件，当应用程序基于Python或Ruby时，它非常有用 • Apache支持更多的OS • Apache提供了比Nginx更多的功能 • Apache依赖于CPU和内存等硬件组件

两者最核心的区别在于apache是同步多进程模型，一个连接对应一个进程，而nginx是异步的，多个连接（万级别）可以对应一个进程。

更为通用的方案是，前端nginx抗并发，后端apache集群，配合起来会更好。

4.13 本次课程设计采用nginx的主要目的和优点?

加载zuul网关前，做负载均衡

1. 反向代理，保证内网的安全，通常将反向代理作为公网访问地址，Web服务器是内网
2. 反向代理+负载均衡，通过反向代理服务器来优化服务器的负载

讨论课5：Docker服务

5.1 什么是Docker

- Docker是一个容器化平台，它将应用程序及其所有依赖项以容器的形式打包在一起，以确保应用程序在任何环境（无论是开发环境、测试环境还是生产环境）中无缝运行。
- Docker容器，将一个软件包在一个完整的文件系统中，其中包含运行所需的一切：代码、运行时、系统工具、系统库等任何可以安装在服务器上的东西。
- 它都将始终运行相同的程序，无论软件的环境如何。

5.2 什么是Docker镜像

- Docker镜像是Docker容器的源代码。
- Docker镜像用于创建容器。使用build命令创建镜像，使用run启动时它们将生成容器。
- 镜像存储在Docker注册表中，registry.hub.docker.com因为它们可能变得非常大，镜像被设计为由其他镜像层组成，允许在通过网络传输镜像时发送最少量的数据

5.3 获得Docker镜像的方式

- 通过仓库、拷贝等方式直接获取
- 使用 Dockerfile文件定义并构建镜像
- 使用maven等工具构建镜像

5.4 描述如何利用Maven来构建Docker镜像？（老师的docker的讨论课标题）

docker-maven-plugin插件构建docker镜像有两种方式：

- 指定参数，由docker-maven-plugin插件根据这些参数来制作镜像
    - 工程目录下新建文件pom_docker.xml，内容和pom.xml一样，然后我们再去节点添加以下内容，放在原有的节点后面
    - build命令构建镜像
- 指定Dockerfile，这和我们用docker build命令来构建镜像的过程一样，只是插件帮我们在创建项目的同时构建镜像。
    - 和第一种方式相比有如下变动：
        - 新增节点用来表示自定义Dockerfile文件的位置
        - 和节点用不上了，在此删掉
        - 过程类似前面说的通过Dockerfile构建镜像

5.5 什么是Docker容器?

- Docker容器包括应用程序及其所有依赖项，但与其他容器共享内核，在主机操作系统的用户空间中作为独立进程运行。
- Docker容器不依赖于任何特定的基础架构：它们可以在任何计算机，任何基础架构和任何云中运行。

5.6 Docker 容器分为哪几类?

仓库又可以分为两种形式：

- public(公有仓库)
    - 是开放给用户使用、允许用户管理镜像的 Registry 服务
- private(私有仓库)
    - 官方提供了 Docker Registry镜像，可以直接使用做为私有 Registry 服务

5.7 什么是Docker Hub 【Repository（仓库）？】?

- Docker hub是一个基于云的注册表服务，允许您链接到代码存储库，构建映像并测试它们，便您可以将镜像部署到主机。
- 开发流程中的容器发现，分发和变更管理，用户和团队协作以及工作流自动化提供了集中资源。

5.8 概要描述一下docker架构?

1. Docker使用的是C/S结构，即客户端/服务器体系结构
2. Docker客户端与服务端可以运行在一台机器上，也可以通过RESTful、stock或网络接口进行远程通信
3. Docker Client：是Docker命令行界面工具，是许多Docker用户与Docker交互的主要方式
4. Docker daemon：是服务器组件，以Linux后台服务的方式运行，是Docker最核心的后台进程，也叫守护进程。它负责响应来自Docker Client的请求，然后将其转化为系统调用来完成容器管理操作

5.9 概要描述一下Docker daemon组成及其内部原理?

Docker daemon内部大致可以分为三部分

- Docker Server
- Engine
- Job

Server负责接收请求，在Engine中处理请求，然后根据请求的类型，创建出指定的Job并运行

5.10 Docker容器在任何给定时间点可以处于什么状态?

- 运行
- 已暂停
- 重新启动
- 已退出

5.11 如何使用Docker?

用户可以通过push、pull命令完成镜像的上传与下载。

用run命令执行镜像。

可以用docker images查看当前存在多少镜像

用docker ps –a查看当前存在的所有容器。

可以使用docker start/stop/restart container-name来管理容器

用rm来删除容器、rmi删除镜像

- 若存在的容器中有使用待删除镜像的，则需要先停止容器运行，再删容器，最后再删镜像。

5.12 Docker优点?

1. 隔离强：Docker可以将应用程序打包封装到一个容器中，该容器包含了应用程序的代码、运行环境、依赖库、配置文件等必需的资源。容器之间达到进程级别的隔离，在容器中的操作，不会影响道宿主机和其他容器，这样就不会出现应用之间相互影响的情形！
2. 可移植性：使用docker容器后，可以实现开发、测试和生产环境的统一化和标准化。镜像作为标准的交付件，可在开发、测试和生产环境上以容器来运行，最终实现三套环境上的应用以及运行所依赖内容的完全一致。（在现在微服务的架构中，一个应用拆成几十个微服务，每个微服务都对应有开发、测试、生产三套环境需要搭建。如果不使用docker。。。呵呵呵）
3. 轻量和高效：和虚拟机相比，容器仅需要封装应用和应用需要的依赖文件，实现轻量的应用运行环境，且拥有比虚拟机更高的硬件资源利用率。（在微服务架构中，有些服务负载压力大，需要以集群部署，可能要部署几十台机器上，对于某些中小型公司来说，使用虚拟机，代价太大。如果用容器，同样的物理机则能支持上千个容器，对中小型公司来说，省钱！）

> 微服务为什么要使用**docker**？（上课提问的一个问题）见**docker**优点系列 （楼上）

讨论课6：基于Redis的商城库存量并发设计

6.1 Redis的介绍

Redis 是完全开源免费的高性能的内存key-value数据库，可用作数据库、缓存和消息中间件。

- 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- 不仅支持简单的key-value（string）类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
- 支持数据的备份，即master-slave模式的数据备份。
- 性能极高 – 读的速度是110000次/s, 写的速度是81000次/s 。
- Redis的所有操作都是原子性的。多个操作也支持事务。

6.2 阐述Redis和传统数据的不同?

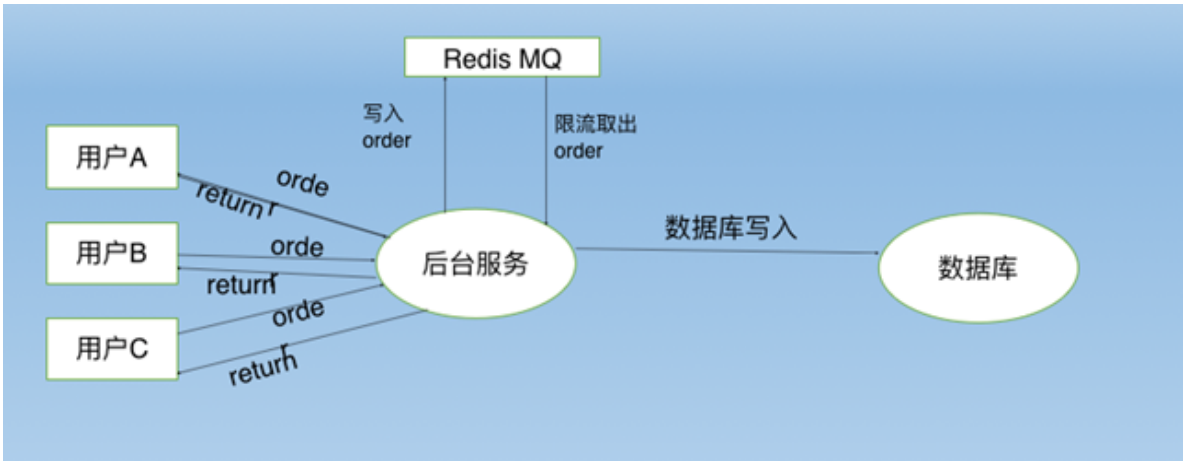
Redis和传统数据的不同及Redis优点：

与传统数据库不同的是 Redis 的数据是存在内存中的，所以存写速度非常快，因此 Redis 被广泛应用于缓存方向。Redis为分布式缓存，在多实例的情况下，各实例共用一份缓存数据，缓存具有一致性。

6.3 为什么使用Redis?

秒杀活动中用户大量的提交创建订单的请求，订单的写入需要操作数据库，而对数据库的写入数据库的操作受硬件的影响很大，硬件设备性能较低时只能支持每秒几十次的操作，大量订单写入数据库很可能会导致数据库的崩溃（缓存雪崩），因此我们需要限制服务发起数据库操作的请求的速度

解决方案：消息队列削峰



6.4 阐述电子商城缓存雪崩的原因?

用户操作 +

1）缓存中大量的key设置相同的过期时间，在某一时刻同时失效

2）缓存数据库因某种原因（缓存穿透， 缓存击穿等）发生宕机或断网，此时对数据库造成的压力不可预计，所造成的雪崩较为致命

6.5 阐述Redis功能?

- 数据缓冲

  (1)Redis是将数据存到内存中，而Mysql存在磁盘中，从内存中读取数据更快

  (2)访问率高但更新频率较低的数据，可以添加缓存，减少Mysql压力

  (3)查询时先从Redis中查，没有再去Mysql中查，将查到的数据放缓存中一份。

- 数据持久化

  redis中内部提供了良好的持久化的策略，保证内存中的数据不丢失，这样redis服务器重启后，依然可以获取其中的数据。

- 分布式锁

  控制分布式系统或不同系统之间共同访问共享资源的一种锁实现，如果不同的系统或同一个系统的不同主机之间共享了某个资源时，往往需要互斥来防止彼此干扰来保证一致性。

- 消息队列

6.6 具体描述用redis解决电子商城库存并发控制的问题?

方案一：基于消息队列和分布式锁的方案:

1. 基于发布/订阅的消息队列模式，将各个客户端设置为订阅状态，库存量的变化将实时推送给相应客户端。将高并发的用户请求写入消息队列， 通过异步处理提高系统处理的性能，进行流量平峰。
2. Redis实现分布式锁。用户通过前端发起请求，每个请求执行前对Redis中的库存加锁，判读库存量是否充足，若充足则扣去相应库存量，进行解锁；若用户请求对库存量加锁失败，则后端拒绝该请求。
3. 预扣库存+Redis集群（主从复制）

将mysql中的库存量提前提取一部分进入Redis缓存，设置预警值，每当Redis中的库存量接近预警值，即从mysql中再提库存进入redis

使用Redis集群（主从复制）的架构，防止主redis中的数据意外丢失。

方案二：

1. 获取商品的锁 → 若未获取成功，判断是否超时，超时则释放锁，再尝试获取锁
2. 获取商品锁成功，则去判断redis中是否有库存。
3. redis中若是没有，则去商品服务扣去预扣库存。如果实际库存为0则释放锁，订单失败。如果不为0，则设置redis。
4. 如果redis有，则去判断是否库存小于阈值。小于则跳转3。
5. 判断redis中商品a的数量是否小于要买的数量。
6. 数量充足，则购买，然后释放锁，购买成功。
7. 数量不充足则根据差值去商品服务上查询实际库存值，然后得出订单结果。

6.7 如何防止两个线程请求同时访问redis造成的超卖问题（1件商品，两个请求）？

为每一个商品设置一个在redis的key,如果key存在，则线程正在访问redis,如果key不存在，则可以进行访问。

讨论课7：Zuul网关

讨论课8：Eureka

讨论课9：Ribbon

###