

尚硅谷-谷粒商城



一、项目简介

1、项目背景

1) 、电商模式

市面上有 5 种常见的电商模式 B2B、B2C、C2B、C2C、O2O；

1、**B2B 模式**

B2B (Business to Business), 是指商家与商家建立的商业关系。如：阿里巴巴

2、**B2C 模式**

B2C (Business to Consumer), 就是我们经常看到的供应商直接把商品卖给用户，即“商对客”模式，也就是通常说的商业零售，直接面向消费者销售产品和服务。如：苏宁易购、京东、天猫、小米商城

3、**C2B 模式**

C2B (Customer to Business), 即消费者对企业。先有消费者需求产生而后有企业生产，即先有消费者提出需求，后有生产企业按需求组织生产

4、**C2C 模式**

C2C (Customer to Consumer) , 客户之间自己把东西放上网去卖，如：淘宝，闲鱼

5、**O2O 模式**

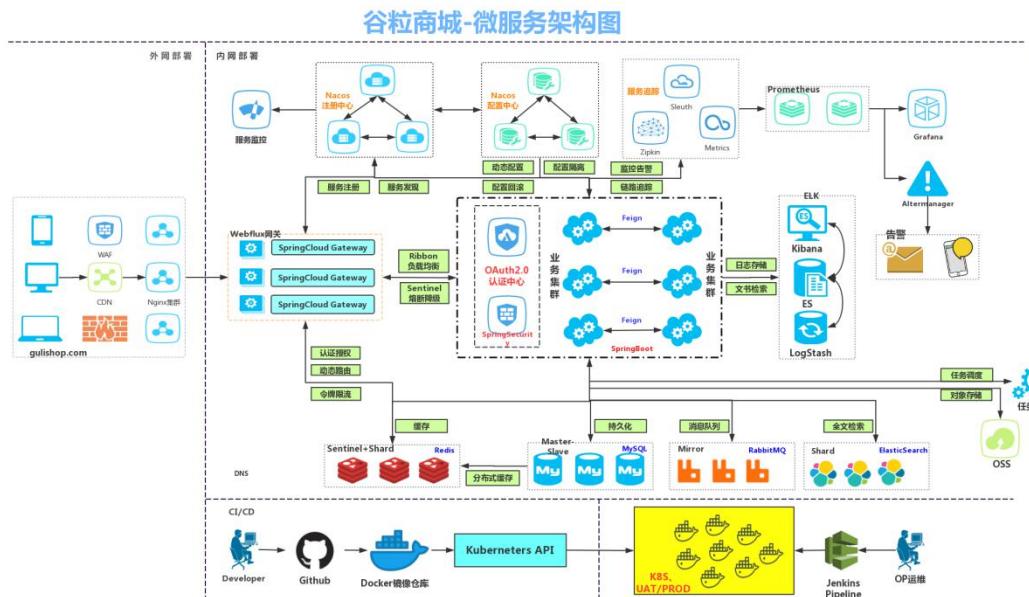
O2O 即 Online To Offline，也即将线下商务的机会与互联网结合在了一起，让互联网成为线下交易的前台。线上快速支付，线下优质服务。如：饿了么，美团，淘票票，京东到家

2) 、谷粒商城

谷粒商城是一个 B2C 模式的电商平台，销售自营商品给客户。

2、项目架构图

1、项目微服务架构图



2、微服务划分图



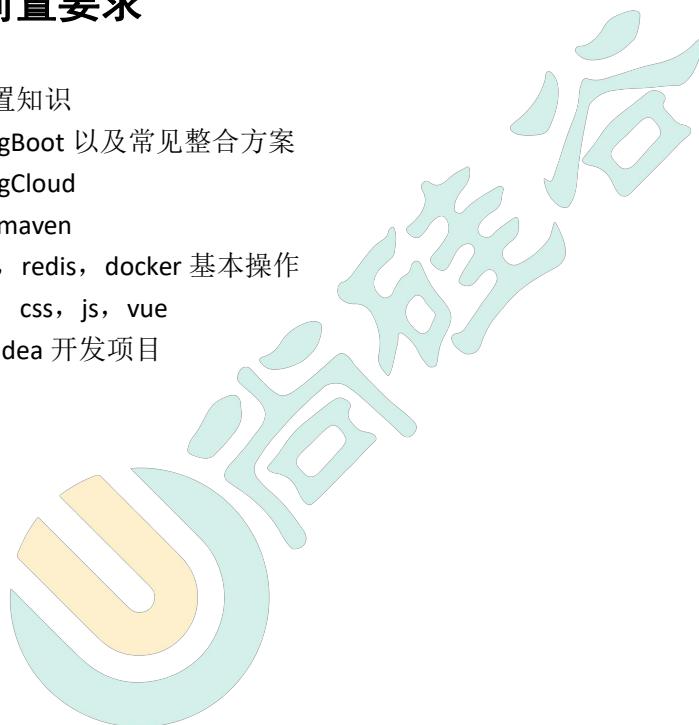
3、项目技术&特色

- 前后分离开发，并开发基于 vue 的后台管理系统
- SpringCloud 全新的解决方案
- 应用监控、限流、网关、熔断降级等分布式方案 全方位涉及
- 透彻讲解分布式事务、分布式锁等分布式系统的难点
- 分析高并发场景的编码方式，线程池，异步编排等使用
- 压力测试与性能优化
- 各种集群技术的区别以及使用
- CI/CD 使用
- ...

4、项目前置要求

学习项目的前置知识

- 熟悉 SpringBoot 以及常见整合方案
- 了解 SpringCloud
- 熟悉 git, maven
- 熟悉 linux, redis, docker 基本操作
- 了解 html, css, js, vue
- 熟练使用 idea 开发项目

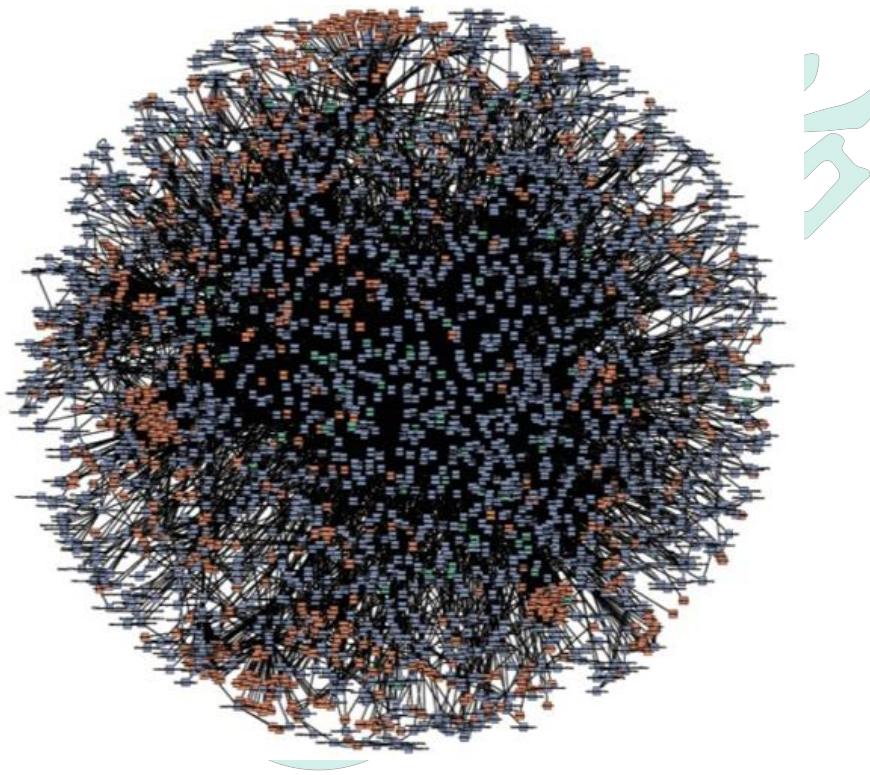


二、分布式基础概念

1、微服务

微服务架构风格，就像是把一个单独的应用程序开发为一套小服务，每个小服务运行在自己的进程中，并使用轻量级机制通信，通常是 HTTP API。这些服务围绕业务能力来构建，并通过完全自动化部署机制来独立部署。这些服务使用不同的编程语言书写，以及不同数据存储技术，并保持最低限度的集中式管理。

简而言之：拒绝大型单体应用，基于业务边界进行服务微化拆分，各个服务独立部署运行。



2、集群&分布式&节点

集群是个物理形态，分布式是个工作方式。

只要是一堆机器，就可以叫集群，他们是不是一起协作着干活，这个谁也不知道；

《分布式系统原理与范型》定义：

“分布式系统是若干独立计算机的集合，这些计算机对于用户来说就像一个相关系统”
分布式系统（distributed system）是建立在网络之上的软件系统。

分布式是指将不同的业务分布在不同的地方。

集群指的是将几台服务器集中在一起，实现同一业务。

例如：京东是一个分布式系统，众多业务运行在不同的机器，所有业务构成一个大型的业

务集群。每一个小的业务，比如用户系统，访问压力大的时候一台服务器是不够的。我们就应该将用户系统部署到多个服务器，也就是每一个业务系统也可以做集群化：

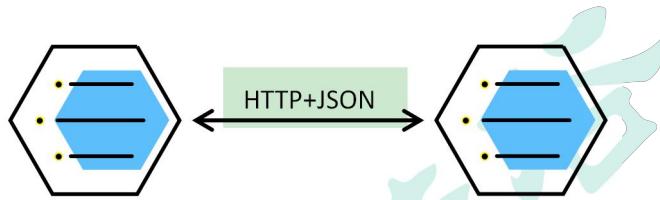
分布式中的每一个节点，都可以做集群。而集群并不一定就是分布式的。

节点：集群中的一个服务器

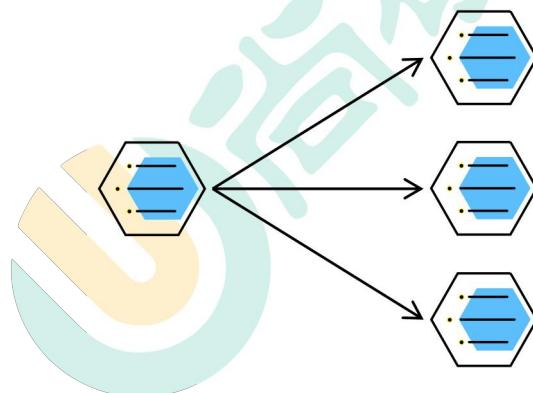
3、远程调用

在分布式系统中，各个服务可能处于不同主机，但是服务之间不可避免的需要互相调用，我们称为远程调用。

SpringCloud 中使用 HTTP+JSON 的方式完成远程调用



4、负载均衡



分布式系统中，A 服务需要调用 B 服务，B 服务在多台机器中都存在，A 调用任意一个服务器均可完成功能。

为了使每一个服务器都不要太忙或者太闲，我们可以负载均衡的调用每一个服务器，提升网站的健壮性。

常见的负载均衡算法：

轮询：为第一个请求选择健康池中的第一个后端服务器，然后按顺序往后依次选择，直到最后一个，然后循环。

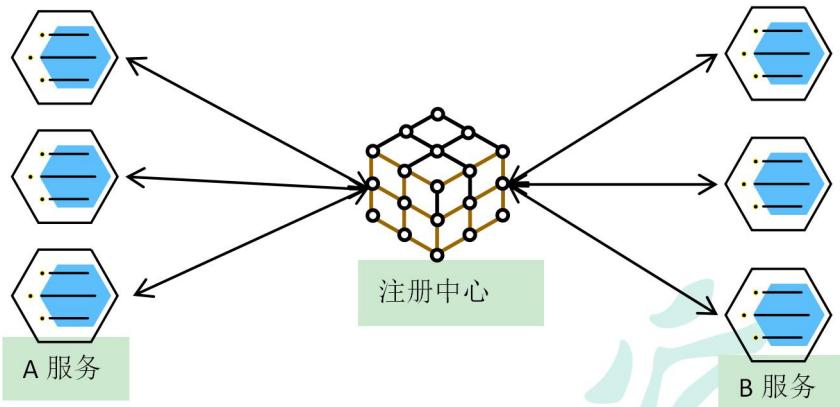
最小连接：优先选择连接数最少，也就是压力最小的后端服务器，在会话较长的情况下可以考虑采取这种方式。

散列：根据请求源的 IP 的散列（hash）来选择要转发的服务器。这种方式可以一定程度上保证特定用户能连接到相同的服务器。如果你的应用需要处理状态而要求用户能连接到

和之前相同的服务器，可以考虑采取这种方式。

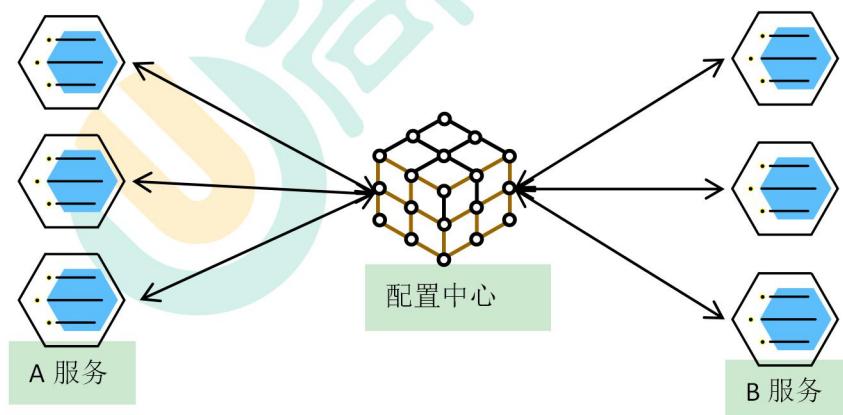
5、服务注册/发现&注册中心

A 服务调用 B 服务，A 服务并不知道 B 服务当前在哪几台服务器有，哪些正常的，哪些服务已经下线。解决这个问题可以引入注册中心；



如果某些服务下线，我们其他人可以实时的感知到其他服务的状态，从而避免调用不可用的服务

6、配置中心



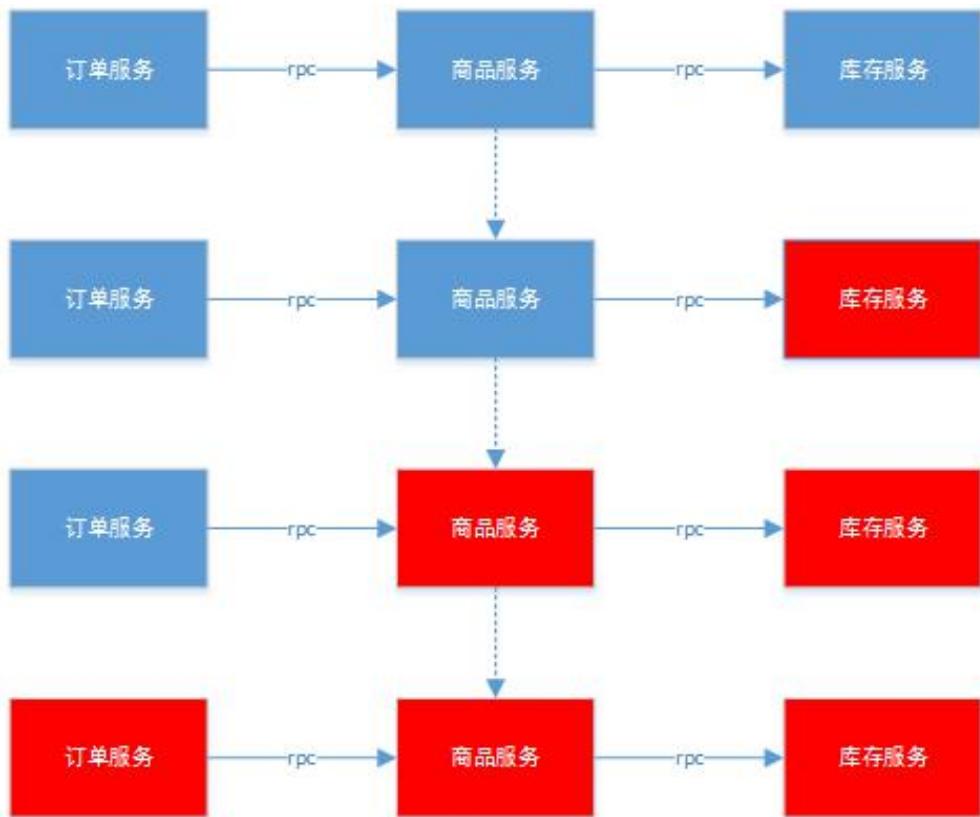
每一个服务最终都有大量的配置，并且每个服务都可能部署在多台机器上。我们经常需要变更配置，我们可以让每个服务在配置中心获取自己的配置。

配置中心用来集中管理微服务的配置信息

7、服务熔断&服务降级

在微服务架构中，微服务之间通过网络进行通信，存在相互依赖，当其中一个服务不可用时，

有可能会造成雪崩效应。要防止这样的情况，必须要有容错机制来保护服务。



1) 服务熔断

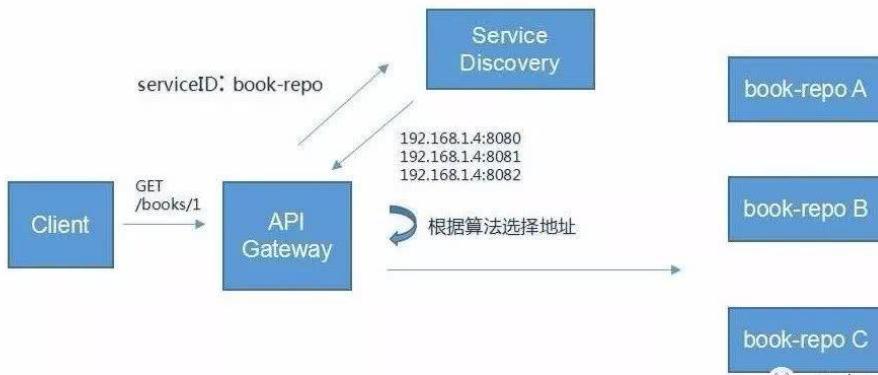
- a. 设置服务的超时，当被调用的服务经常失败到达某个阈值，我们可以开启断路保护机制，后来的请求不再去调用这个服务。本地直接返回默认的数据

2) 服务降级

- a. 在运维期间，当系统处于高峰期，系统资源紧张，我们可以让非核心业务降级运行。降级：某些服务不处理，或者简单处理【抛异常、返回 NULL、调用 Mock 数据、调用 Fallback 处理逻辑】。

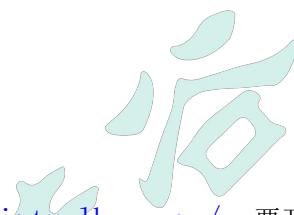
8、API 网关

在微服务架构中，API Gateway 作为整体架构的重要组件，它抽象了微服务中都需要的公共功能，同时提供了客户端负载均衡，服务自动熔断，灰度发布，统一认证，限流流控，日志统计等丰富的功能，帮助我们解决很多 API 管理难题。

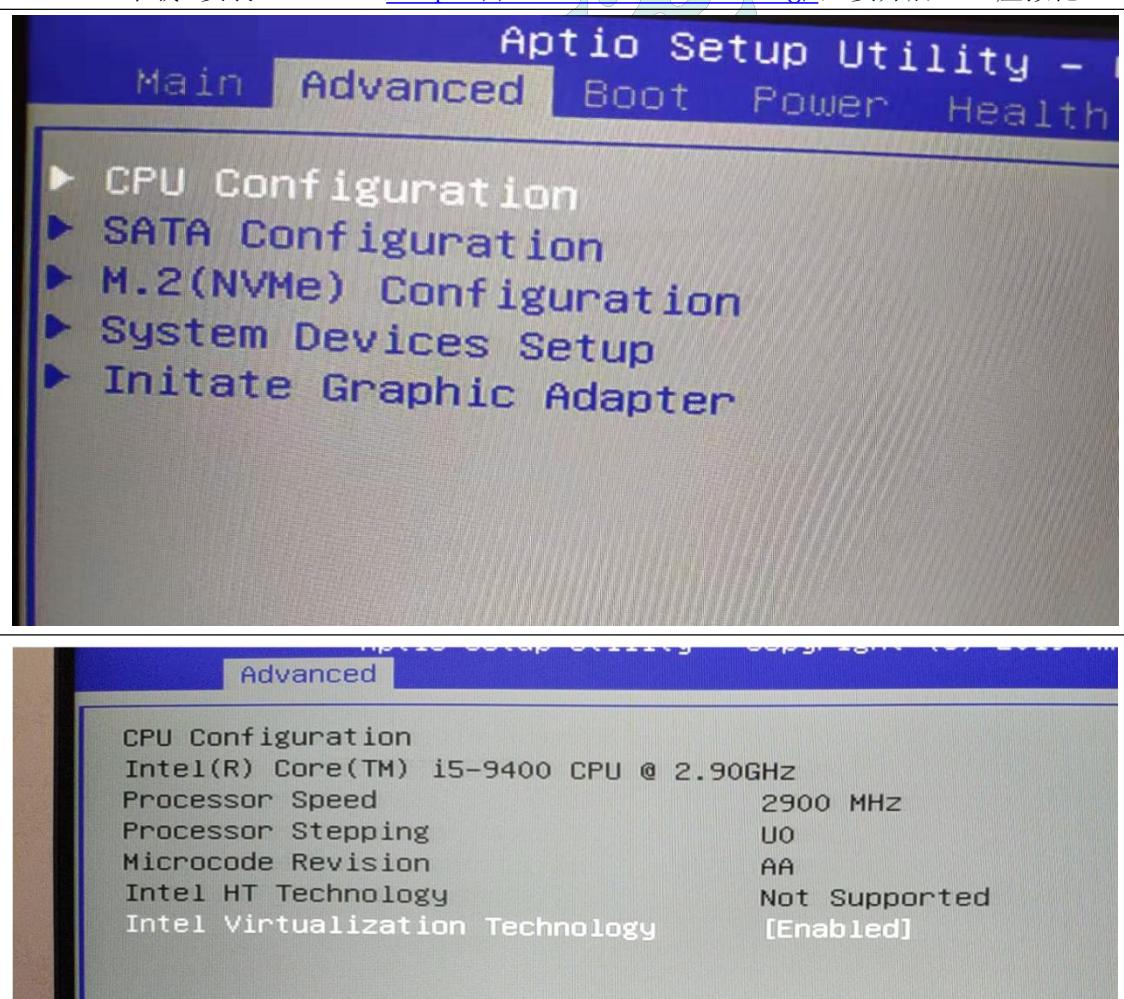


三、环境搭建

1、安装 linux 虚拟机



- 下载&安装 VirtualBox <https://www.virtualbox.org/>, 要开启 CPU 虚拟化



- 下载&安装 Vagrant
 - <https://app.vagrantup.com/boxes/search> Vagrant 官方镜像仓库
 - <https://www.vagrantup.com/downloads.html> Vagrant 下载
- 打开 window cmd 窗口，运行 `Vagrant init centos/7`，即可初始化一个 centos7 系统
- 运行 `vagrant up` 即可启动虚拟机。系统 root 用户的密码是 `vagrant`
- `vagrant` 其他常用命令
 - `vagrant ssh`: 自动使用 `vagrant` 用户连接虚拟机。
 - ◆ `vagrant upload source [destination] [name|id]`: 上传文件
 - <https://www.vagrantup.com/docs/cli/init.html> Vagrant 命令行
- 默认虚拟机的 ip 地址不是固定 ip，开发不方便
 - 修改 `Vagrantfile`

```
config.vm.network "private_network", ip: "192.168.56.10"
```

这里的 ip 需要在物理机下使用 `ipconfig` 命令找到

以太网适配器 VirtualBox Host-Only Network:

```
连接特定的 DNS 后缀 . . . . .
本地链接 IPv6 地址 . . . . . : fe80::8d75:2940:2e9f:b32b%6
IPv4 地址 . . . . . : 192.168.56.1
子网掩码 . . . . . : 255.255.255.0
默认网关 . . . . . :
```

改为这个指定的子网地址

- 重新使用 `vagrant up` 启动机器即可。然后再 `vagrant ssh` 连接机器
- 默认只允许 ssh 登录方式，为了后来操作方便，文件上传等，我们可以配置允许账号密码登录

Vagrant ssh 进去系统之后

```
vi /etc/ssh/sshd_config
```

修改 `PasswordAuthentication yes/no`

重启服务 `service sshd restart`

- 以后可以使用提供的 ssh 连接工具直接连接

注意：VirtualBox 会与包括但不限于如下软件冲突，需要卸载这些软件，然后重启电脑；

冲突的软件：红蜘蛛，360，净网大师（有可能）等

修改 linux 的 yum 源

1)、备份原 yum 源

```
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
```

2)、使用新 yum 源

```
curl -o /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.163.com/.help/CentOS7-Base-163.repo
```

3)、生成缓存

```
yum makecache
```

2、安装 docker

Docker 安装文档: <https://docs.docker.com/install/linux/docker-ce/centos/>

1、卸载系统之前的 docker

```
sudo yum remove docker \
    docker-client \
    docker-client-latest \
    docker-common \
    docker-latest \
    docker-latest-logrotate \
    docker-logrotate \
    docker-engine
```

2、安装 Docker-CE

安装必须的依赖

```
sudo yum install -y yum-utils \
    device-mapper-persistent-data \
    lvm2
```

设置 docker repo 的 yum 位置

```
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

安装 docker，以及 docker-cli

```
sudo yum install docker-ce docker-ce-cli containerd.io
```

3、启动 docker

```
sudo systemctl start docker
```

4、设置 docker 开机自启

```
sudo systemctl enable docker
```

5、测试 docker 常用命令，注意切换到 root 用户下

<https://docs.docker.com/engine/reference/commandline/docker/>

6、配置 docker 镜像加速

阿里云，容器镜像服务

针对 Docker 客户端版本大于 1.10.0 的用户

您可以通过修改 daemon 配置文件/etc/docker/daemon.json 来使用加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://82m9ar63.mirror.aliyuncs.com"]}
```

```
}
```

```
EOF
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl restart docker
```

3、docker 安装 mysql

1、下载镜像文件

```
docker pull mysql:5.7
```

2、创建实例并启动

```
docker run -p 3306:3306 --name mysql \
-v /mydata/mysql/log:/var/log/mysql \
-v /mydata/mysql/data:/var/lib/mysql \
-v /mydata/mysql/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=root \
-d mysql:5.7
```

参数说明

- p 3306:3306: 将容器的 3306 端口映射到主机的 3306 端口
- v /mydata/mysql/conf:/etc/mysql: 将配置文件夹挂载到主机
- v /mydata/mysql/log:/var/log/mysql: 将日志文件夹挂载到主机
- v /mydata/mysql/data:/var/lib/mysql/: 将配置文件夹挂载到主机
- e MYSQL_ROOT_PASSWORD=root: 初始化 root 用户的密码

MySQL 配置

```
vi /mydata/mysql/conf/my.cnf
```

```
[client]
```

```
default-character-set=utf8
```

```
[mysql]
```

```
default-character-set=utf8
```

```
[mysqld]
```

```
init_connect='SET collation_connection = utf8_unicode_ci'
```

```
init_connect='SET NAMES utf8'
```

```
character-set-server=utf8
```

```
collation-server=utf8_unicode_ci
```

```
skip-character-set-client-handshake
```

```
skip-name-resolve
```

注意：解决 MySQL 连接慢的问题

在配置文件中加入如下，并重启 mysql

[mysqld]

skip-name-resolve

解释：

skip-name-resolve：跳过域名解析

3、通过容器的 mysql 命令行工具连接

```
docker exec -it mysql mysql -uroot -proot
```

4、设置 root 远程访问

```
grant all privileges on *.* to 'root'@'%' identified by 'root' with grant option;  
flush privileges;
```

5、进入容器文件系统

```
docker exec -it mysql /bin/bash
```

4、docker 安装 redis

1、下载镜像文件

```
docker pull redis
```

2、创建实例并启动

```
mkdir -p /mydata/redis/conf  
touch /mydata/redis/conf/redis.conf
```

```
docker run -p 6379:6379 --name redis -v /mydata/redis/data:/data \  
-v /mydata/redis/conf/redis.conf:/etc/redis/redis.conf \  
-d redis redis-server /etc/redis/redis.conf
```

redis 自描述文件：

<https://raw.githubusercontent.com/antirez/redis/4.0/redis.conf>

3、使用 redis 镜像执行 redis-cli 命令连接

```
docker exec -it redis redis-cli
```

5、开发环境统一

1、Maven

配置阿里云镜像

```
<mirrors>
  <mirror>
    <id>nexus-aliyun</id>
    <mirrorOf>central</mirrorOf>
    <name>Nexus aliyun</name>
    <url>http://maven.aliyun.com/nexus/content/groups/public</url>
  </mirror>
</mirrors>
```

配置 jdk1.8 编译项目

```
<profiles>
  <profile>
    <id>jdk-1.8</id>
    <activation>
      <activeByDefault>true</activeByDefault>
      <jdk>1.8</jdk>
    </activation>
    <properties>
      <maven.compiler.source>1.8</maven.compiler.source>
      <maven.compiler.target>1.8</maven.compiler.target>
      <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
    </properties>
  </profile>
</profiles>
```

2、Idea&VsCode

Idea 安装 lombok、mybatisx 插件

Vscode 安装开发必备插件

Vetur —— 语法高亮、智能感知、Emmet 等

包含格式化功能，Alt+Shift+F（格式化全文），Ctrl+K Ctrl+F（格式化选中代码，两个 Ctrl

需要同时按着)

EsLint —— 语法纠错

Auto Close Tag —— 自动闭合 HTML/XML 标签

Auto Rename Tag —— 自动完成另一侧标签的同步修改

JavaScript(ES6) code snippets —— ES6 语法智能提示以及快速输入，除 js 外还支持.ts, .jsx, .tsx, .html, .vue, 省去了配置其支持各种包含 js 代码文件的时间

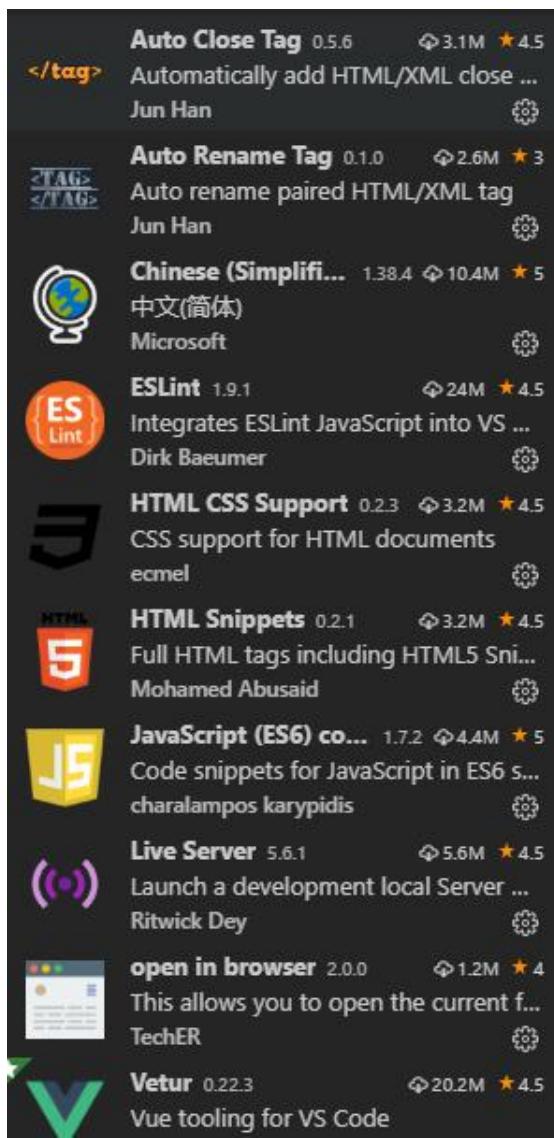
HTML CSS Support —— 让 html 标签上写 class 智能提示当前项目所支持的样式

HTML Snippets —— html 快速自动补全

Open in browser —— 浏览器快速打开

Live Server —— 以内嵌服务器方式打开

Chinese (Simplified) Language Pack for Visual Studio Code —— 中文语言包



3、安装配置 git

1、下载 git; <https://git-scm.com>

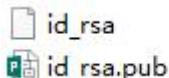
2、配置 git，进入 git bash

```
# 配置用户名  
git config --global user.name "username" // (名字)  
# 配置邮箱  
git config --global user.email "username@email.com" // (注册账号时用的邮箱)
```

3、配置 ssh 免密登录

<https://gitee.com/help/articles/4181#article-header0>

进入 git bash；使用：ssh-keygen -t rsa -C "xxxxx@xxxxx.com" 命令。连续三次回车。
一般用户目录下会有



id_rsa
id_rsa.pub

或者 cat ~/.ssh/id_rsa.pub

登录进入 gitee，在设置里面找到 SSH KEY 将.pub 文件的内容粘贴进去
使用 ssh -T <git@gitee.com> 测试是否成功即可

Git+码云教程 <https://gitee.com/help/articles/4104>

4、逆向工程使用

- 1、导入项目逆向工程
- 2、下载人人开源后台管理系统脚手架工程
 - (1) 导入工程，创建数据库
 - (2) 修改工程 shiro 依赖为 SpringSecurity
 - (3) 删部分暂时不需要的业务
- 3、下载人人开源后台管理系统 vue 端脚手架工程
 - (1) vscode 导入前端项目
 - (2) 前后端联调测试基本功能

6、创建项目微服务

商品服务、仓储服务、订单服务、优惠券服务、用户服务

共同：

- 1)、web、openfeign
- 2)、每一个服务，包名 com.atguigu.gulimall.xxx(product/order/ware/coupon/member)
- 3)、模块名：gulimall-coupon

1) 、从 gitee 初始化一个项目

新建仓库

仓库名称 ✓
gulimall

归属 路径
leifengyang / gulimall

仓库地址: <https://gitee.com/leifengyang/gulimall>

仓库介绍 非必填
谷粒商城

是否开源
 私有 公开
任何人都可以访问该仓库的代码和其他任何形式的资源

选择语言 Java 添加 .gitignore Maven 添加开源许可证 Apache-2.0

使用 README 文件初始化这个仓库
 使用 Issue 模板文件初始化这个仓库 ⓘ
 使用 Pull Request 模板文件初始化这个仓库 ⓘ

选择分支模型 (仓库初始化后将根据所选分支模型创建分支)
生产/开发模型 (支持 master/develop 类型分支)

导入已有仓库

创建

2) 、创建各个微服务项目

- 1) 、了解人人开源项目，快速搭建后台脚手架
- 2) 、修改代码调整为我们的业务逻辑
- 3) 、创建各个微服务以及数据库

谷粒商城

SpringCloud 组件



一、SpringCloud Alibaba

1、SpringCloud Alibaba 简介

1)、简介

Spring Cloud Alibaba 致力于提供微服务开发的一站式解决方案。此项目包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务。

依托 Spring Cloud Alibaba，您只需要添加一些注解和少量配置，就可以将 Spring Cloud 应用接入阿里微服务解决方案，通过阿里中间件来迅速搭建分布式应用系统。

<https://github.com/alibaba/spring-cloud-alibaba>

2)、为什么使用



百度为您找到相关结果约184,000个

搜索工具

[Spring Cloud微服务注册中心Eureka 2.x停止维护了咋办?... CSDN博客](#)

2019年2月26日 - 用中文给大家翻译一下,这里的意思就是说 Eureka 2.0 的开源工作已经停止了,如果你要用 Eureka 2.x 版本的代码来部署到生产环境的话,一切后果请自负。 ...

C CSDN技术社区 - 百度快照

[凉凉了,Eureka 2.x 停止维护, Spring Cloud 何去何从? -... CSDN博客](#)

2018年7月11日 - 同时很不幸, Spring Cloud 下的 Netflix Eureka 组件

[Hystrix停止开发,我们该何去何从? | 周立的博客 - 关注Spring ...](#)

2018年11月29日 - 是的,Hystrix停止开发了。官方的新闻如下: 考虑到之前Netflix宣布Eureka 2.0孵化失败时,被业界过度消费(关于Eureka 2.x,别再人云亦云了!),为了防止再...
[www.itmuch.com/spring-... - 百度快照](http://www.itmuch.com/spring-...)

[Hystrix停止更新了!告诉你如何应对! - Java后端技术 - CSDN博客](#)

2018年12月1日 - 该博客停止维护,新博客地址<https://blog.kuanggenping.com/> 博文 Hystrix 停止开发。。。 Spring Cloud 何去何从? 12-03 阅读数 41 栈长得到消息, Hys...
C CSDN技术社区 - 百度快照

SpringCloud 的几大痛点

SpringCloud 部分组件停止维护和更新，给开发带来不便；

SpringCloud 部分环境搭建复杂，没有完善的可视化界面，我们需要大量的二次开发和定制

SpringCloud 配置复杂，难以上手，部分配置差别难以区分和合理应用

SpringCloud Alibaba 的优势：

阿里使用过的组件经历了考验，性能强悍，设计合理，现在开源出来大家用

成套的产品搭配完善的可视化界面给开发运维带来极大的便利

搭建简单，学习曲线低。

结合 SpringCloud Alibaba 我们最终的技术搭配方案：

SpringCloud Alibaba - Nacos: 注册中心（服务发现/注册）

SpringCloud Alibaba - Nacos: 配置中心（动态配置管理）

SpringCloud - Ribbon: 负载均衡

SpringCloud - Feign: 声明式 HTTP 客户端（调用远程服务）

SpringCloud Alibaba - Sentinel: 服务容错（限流、降级、熔断）

SpringCloud - Gateway: API 网关（webflux 编程模式）

SpringCloud - Sleuth: 调用链监控

SpringCloud Alibaba - Seata: 原 Fescar，即分布式事务解决方案

3)、版本选择

由于 Spring Boot 1 和 Spring Boot 2 在 Actuator 模块的接口和注解有很大的变更，且 spring-cloud-commons 从 1.x.x 版本升级到 2.0.0 版本也有较大的变更，因此我们采取跟 SpringBoot 版本号一致的版本：

- 1.5.x 版本适用于 Spring Boot 1.5.x
- 2.0.x 版本适用于 Spring Boot 2.0.x
- 2.1.x 版本适用于 Spring Boot 2.1.x

4)、项目中的依赖

在 common 项目中引入如下。进行统一管理

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>2.1.0.RELEASE</version>
            <type>pom</type>
```

```
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

2、SpringCloud Alibaba-Nacos[作为注册中心]

Nacos 是阿里巴巴开源的一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。他是使用 java 编写。需要依赖 java 环境

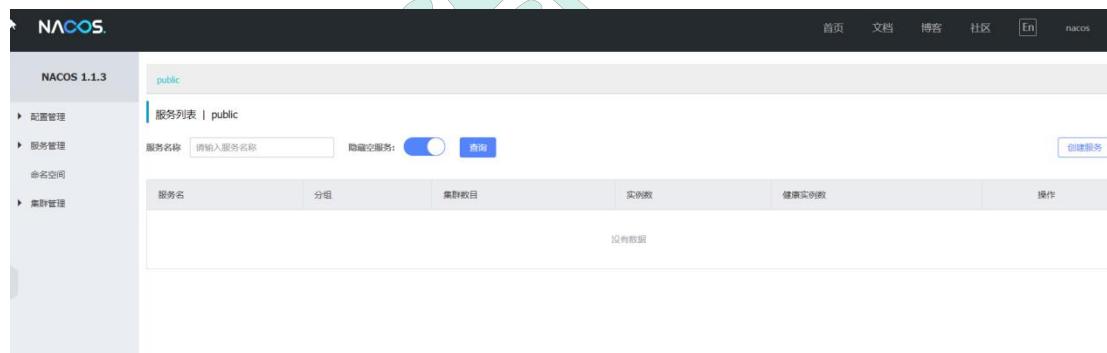
Nacos 文档地址：<https://nacos.io/zh-cn/docs/quick-start.html>

1)、下载 nacos-server

<https://github.com/alibaba/nacos/releases>

2)、启动 nacos-server

- 双击 bin 中的 startup.cmd 文件
- 访问 <http://localhost:8848/nacos/>
- 使用默认的 nacos/nacos 进行登录



3)、将微服务注册到 nacos 中

1、首先，修改 pom.xml 文件，引入 Nacos Discovery Starter。

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

2、在应用的 `/src/main/resources/application.properties` 配置文件中配置 Nacos Server 地址

```
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

3、使用`@EnableDiscoveryClient` 开启服务注册发现功能

```
@SpringBootApplication  
@EnableDiscoveryClient  
public class ProviderApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```

4、启动应用，观察 nacos 服务列表是否已经注册上服务

注意：每一个应用都应该有名字，这样才能注册上去。修改 `application.properties` 文件

```
spring.application.name=service-provider  
server.port=8000
```

5、注册更多的服务上去，测试使用 feign 远程调用

Nacos 使用三步

- 1、导包 `nacos-discovery`
- 2、写配置，指定 nacos 地址，指定应用的名字
- 3、开启服务注册发现功能`@EnableDiscoveryClient`

Feign 使用三步

- 1、导包 `openfeign`
- 2、开启`@EnableFeignClients` 功能
- 3、编写接口，进行远程调用

```
@FeignClient("stores")  
public interface StoreClient {  
    @RequestMapping(method = RequestMethod.GET, value = "/stores")  
    List<Store> getStores();
```

```
@RequestMapping(method = RequestMethod.POST, value = "/stores/{storeId}", consumes = "application/json")
Store update(@PathVariable("storeId") Long storeId, Store store);
}
```

6、更多配置

<https://github.com/alibaba/spring-cloud-alibaba/blob/master/spring-cloud-alibaba-examples/nacos-example/nacos-discovery-example/readme-zh.md#more>

3、SpringCloud Alibaba-Nacos[作为配置中心]

1、pom.xml 引入 Nacos Config Starter。

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

2、在应用的 /src/main/resources/bootstrap.properties 配置文件中配置 Nacos Config 元数据

```
spring.application.name=nacos-config-example
spring.cloud.nacos.config.server-addr=127.0.0.1:8848
```

主要配置应用名和配置中心地址

3、在 nacos 中添加配置

在 nacos 中创建一个 **应用名.properties** 配置文件并编写配置

Nacos Config 数据结构

Nacos Config 主要通过 dataId 和 group 来唯一确定一条配置。

Nacos Client 从 Nacos Server 端获取数据时，调用的是此接口 ConfigService.getConfig(String)

dataId, String group, long timeoutMs)。

Spring Cloud 应用获取数据

dataID:

在 Nacos Config Starter 中，dataId 的拼接格式如下

- \${prefix} - \${spring.profiles.active} . \${file-extension} prefix 默认为 spring.application.name 的值，也可以通过配置项 spring.cloud.nacos.config.prefix 来配置。
- spring.profiles.active 即为当前环境对应的 profile

注意，当 activeprofile 为空时，对应的连接符 - 也将不存在，dataId 的拼接格式变成 \${prefix}.\${file-extension}

file-extension 为配置内容的数据格式，可以通过配置项

spring.cloud.nacos.config.file-extension 来配置。目前只支持 properties 类型。

Group:

Group 默认为 DEFAULT_GROUP，可以通过 spring.cloud.nacos.config.group 配置。

4、在应用中使用@Value 和@RefreshScope

完成上述两步后，应用会从 Nacos Config 中获取相应的配置，并添加在 Spring Environment 的 PropertySources 中。这里我们使用 @Value 注解来将对应的配置注入到 SampleController 的 userName 和 age 字段，并添加 @RefreshScope 打开动态刷新功能

@RefreshScope

```
class SampleController {  
  
    @Value("${user.name}")  
    String userName;  
  
    @Value("${user.age}")  
    int age;  
}
```

5、进阶

1、核心概念

命名空间：

用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置。Namespace 的常用场景之一是不同环境的配置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。

配置集：

一组相关或者不相关的配置项的集合称为配置集。在系统中，一个配置文件通常就是一个配置集，包含了系统各个方面的配置。例如，一个配置集可能包含了数据源、线程池、日志级别等配置项。

配置集 ID:

Nacos 中的某个配置集的 ID。配置集 ID 是组织划分配置的维度之一。**Data ID** 通常用于组织划分系统的配置集。一个系统或者应用可以包含多个配置集，每个配置集都可以被一个有意义的名称标识。**Data ID** 通常采用类 Java 包（如 com.taobao.tc.refund.log.level）的命名规则保证全局唯一性。此命名规则非强制。

配置分组:

Nacos 中的一组配置集，是组织配置的维度之一。通过一个有意义的字符串（如 Buy 或 Trade）对配置集进行分组，从而区分 Data ID 相同的配置集。当您在 Nacos 上创建一个配置时，如果未填写配置分组的名称，则配置分组的名称默认采用 DEFAULT_GROUP。配置分组的常见场景：不同的应用或组件使用了相同的配置类型，如 database_url 配置和 MQ_topic 配置。

2、原理

自动注入:

NacosConfigStarter 实现了 org.springframework.cloud.bootstrap.config.PropertySourceLocator 接口，并将优先级设置成了最高。

在 Spring Cloud 应用启动阶段，会主动从 Nacos Server 端获取对应的数据，并将获取到的数据转换成 PropertySource 且注入到 Environment 的 PropertySources 属性中，所以使用 @Value 注解也能直接获取 Nacos Server 端配置的内容。

动态刷新:

Nacos Config Starter 默认为所有获取数据成功的 Nacos 的配置项添加了监听功能，在监听到服务端配置发生变化时会实时触发 org.springframework.cloud.context.refresh.ContextRefresher 的 refresh 方法。

如果需要对 Bean 进行动态刷新，请参照 Spring 和 Spring Cloud 规范。推荐给类添加 @RefreshScope 或 @ConfigurationProperties 注解，

3、加载多配置文件

```
spring.cloud.nacos.config.server-addr=127.0.0.1:8848
spring.cloud.nacos.config.namespace=31098de9-fa28-41c9-b0bd-c754ce319ed4
spring.cloud.nacos.config.ext-config[0].data-id=gulimall-datasource.yml
spring.cloud.nacos.config.ext-config[0].refresh=false
spring.cloud.nacos.config.ext-config[0].group=dev
```

4、namespace 与 group 最佳实践

每个微服务创建自己的 namespace 进行隔离，group 来区分 dev, beta, prod 等环境

4、SpringCloud Alibaba-Sentinel

1、简介

官方文档：<https://github.com/alibaba/Sentinel/wiki/%E4%BB%8B%E7%BB%8D>

项目地址：<https://github.com/alibaba/Sentinel>

随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

Sentinel 具有以下特征：

丰富的应用场景：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。

完备的实时监控：Sentinel 同时提供实时的监控功能。您可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。

广泛的开源生态：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。您只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。

完善的 SPI 扩展点：Sentinel 提供简单易用、完善的 SPI 扩展接口。您可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。



Sentinel 分为两个部分：

- 核心库（Java 客户端）不依赖任何框架/库，能够运行于所有 Java 运行时环境，同时

对 Dubbo / Spring Cloud 等框架也有较好的支持。

- 控制台（Dashboard）基于 Spring Boot 开发，打包后可以直接运行，不需要额外的 Tomcat 等应用容器。

Sentinel 基本概念

- 资源

资源是 Sentinel 的关键概念。它可以是 Java 应用程序中的任何内容，例如，由应用程序提供的服务，或由应用程序调用的其它应用提供的服务，甚至可以是一段代码。在接下来的文档中，我们都会用资源来描述代码块。

只要通过 **Sentinel API** 定义的代码，就是资源，能够被 **Sentinel** 保护起来。大部分情况下，可以使用方法签名，URL，甚至服务名称作为资源名来标示资源。

- 规则

围绕资源的实时状态设定的规则，可以包括流量控制规则、熔断降级规则以及系统保护规则。所有规则可以动态实时调整。



2、Hystrix 与 Sentinel 比较

Hystrix与Sentinel

功能	Sentinel	Hystrix
隔离策略	信号量隔离(并发线程数限流)	线程池隔离/信号量隔离
熔断降级策略	基于响应时间、异常比率、异常数	基于异常比率
实时统计实现	滑动窗口(LeapArray)	滑动窗口(基于RxJava)
动态规则配置	支持多种数据源	支持多种数据源
扩展性	多个扩展点	插件形式
基于注解的支持	支持	支持
限流	基于QPS,支持基于调用关系的限流	有限的支持
流量整形	支持预热模式、匀速器模式、预热排队模式	不支持
系统自适应保护	支持	不支持
控制台	可配置规则、查看秒级监控、机器发现等	简单的监控查看

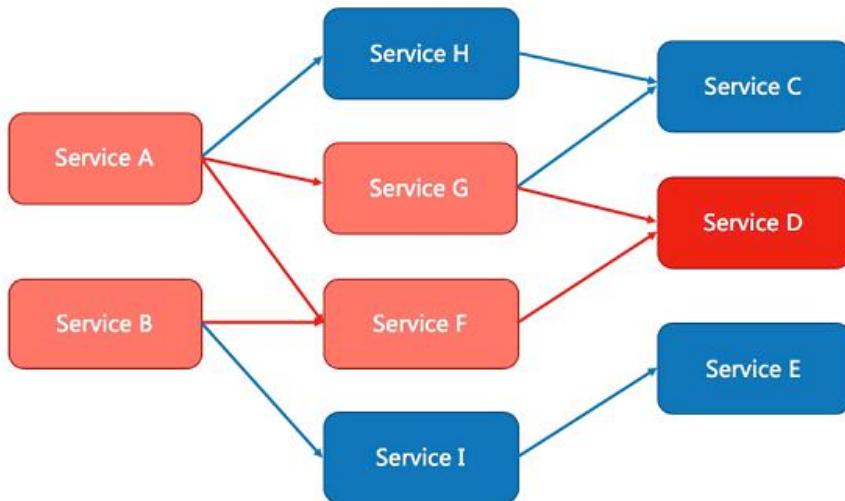
3、整合 Feign+Sentinel 测试熔断降级

<https://github.com/alibaba/Sentinel/wiki/%E4%B8%BB%E9%A1%B5>

什么是熔断降级

除了流量控制以外，降低调用链路中的不稳定资源也是 Sentinel 的使命之一。由于调用关

系的复杂性，如果调用链路中的某个资源出现了不稳定，最终会导致请求发生堆积。



Sentinel 和 Hystrix 的原则是一致的：当检测到调用链路中某个资源出现不稳定的表现，例如响应时间长或异常比例升高的时候，则对这个资源的调用进行限制，让请求快速失败，避免影响到其它的资源而导致级联故障。

熔断降级设计理念

在限制的手段上，Sentinel 和 Hystrix 采取了完全不一样的方法。

Hystrix 通过 线程池隔离 的方式，来对依赖（在 Sentinel 的概念中对应 资源）进行了隔离。这样做的好处是资源和资源之间做到了最彻底的隔离。缺点是除了增加了线程切换的成本（过多的线程池导致线程数目过多），还需要预先给各个资源做线程池大小的分配。

Sentinel 对这个问题采取了两种手段：

- 通过并发线程数进行限制

和资源池隔离的方法不同，Sentinel 通过限制资源并发线程的数量，来减少不稳定资源对其它资源的影响。这样不但没有线程切换的损耗，也不需要您预先分配线程池的大小。当某个资源出现不稳定的情况下，例如响应时间变长，对资源的直接影响就是会造成线程数的逐步堆积。当线程数在特定资源上堆积到一定的数量之后，对该资源的新请求就会被拒绝。堆积的线程完成任务后才开始继续接收请求。

- 通过响应时间对资源进行降级

除了对并发线程数进行控制以外，Sentinel 还可以通过响应时间来快速降级不稳定的资源。当依赖的资源出现响应时间过长后，所有对该资源的访问都会被直接拒绝，直到过了指定的时间窗口之后才重新恢复。

整合测试：

<https://github.com/alibaba/spring-cloud-alibaba/blob/master/spring-cloud-alibaba-examples/sentinel-example/sentinel-feign-example/readme-zh.md>

1、引入依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>

<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
```

2、使用 Nacos 注册中心

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

3、定义 fallback 实现

在服务消费者中，实现 feign 远程接口，接口的实现方法即为调用错误的容错方法

```
public class OrderFeignServiceFallBack implements OrderFeignService {
    @Override
    public Resp<OrderVo> getOrderInfo(String orderSn) {
        return null;
    }
}
```

4、定义 fallbackfactory 并放在容器中

```
@Component
public class OrderFeignFallbackFactory implements
FallbackFactory<OrderFeignServiceFallBack> {
    @Override
    public OrderFeignServiceFallBack create(Throwable throwable) {
        return new OrderFeignServiceFallBack(throwable);
    }
}
```

5、改造 fallback 类接受异常并实现容错方法

```
public class OrderFeignServiceFallBack implements OrderFeignService {

    private Throwable throwable;
    public OrderFeignServiceFallBack(Throwable throwable){
        this.throwable = throwable;
    }
}
```

```
@Override  
public Resp<OrderVo> getOrderInfo(String orderSn) {  
    return Resp.fail(new OrderVo());  
}  
}
```

6、远程接口配置 feign 客户端容错

```
@FeignClient(value = "gulimall-oms", fallbackFactory =  
OrderFeignFallbackFactory.class)  
public interface OrderFeignService {  
  
    @GetMapping("/oms/order/bySn/{orderSn}")  
    public Resp<OrderVo> getOrderInfo(@PathVariable("orderSn") String  
orderSn);  
}
```

7、开启 sentinel 代理 feign 功能；在 application.properties 中配置

feign.sentinel.enabled=true

测试熔断效果。当远程服务出现问题，会自动调用回调方法返回默认数据，并且

更快的容错方式

1、使用@SentinelResource， 并定义 fallback

```
@SentinelResource(value = "order", fallback = "e")
```

Fallback 和原方法签名一致，但是最多个一个 Throwable 类型的变量接受异常。

<https://github.com/alibaba/Sentinel/wiki/%E6%B3%A8%E8%A7%A3%E6%94%AF%E6%8C%81>

需要给容器中配置注解切面

```
@Bean  
public SentinelResourceAspect sentinelResourceAspect() {  
    return new SentinelResourceAspect();  
}
```

在控制台添加降级策略

编辑降级规则

资源名	order
流控应用	default
阈值类型	<input type="radio"/> RT <input checked="" type="radio"/> 异常比例
异常比例	0.1
时间窗口	10

保存 取消

2、测试降级效果

当远程服务停止，前几个服务会尝试调用远程服务，满足降级策略条件以后则不会再尝试调用远程服务

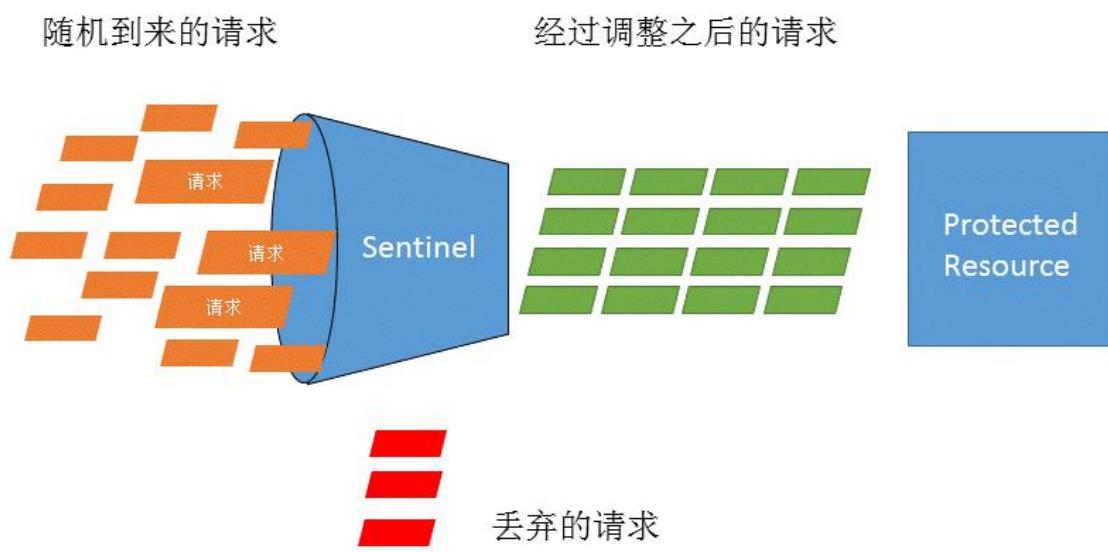
4、整合 Sentinel 测试限流（流量控制）

<https://github.com/alibaba/spring-cloud-alibaba/blob/master/spring-cloud-alibaba-examples/sentinel-example/sentinel-core-example/readme-zh.md>

什么是流量控制

流量控制在网络传输中是一个常用的概念，它用于调整网络包的发送数据。然而，从系统稳定性角度考虑，在处理请求的速度上，也有非常多的讲究。任意时间到来的请求往往是随机不可控的，而系统的处理能力是有限的。我们需要根据系统的处理能力对流量进行控制。

Sentinel 作为一个调度器，可以根据需要把随机的请求调整成合适的形状，如下图所示：



流量控制设计理念

流量控制有以下几个角度：

- 资源的调用关系，例如资源的调用链路，资源和资源之间的关系；
- 运行指标，例如 QPS、线程池、系统负载等；
- 控制的效果，例如直接限流、冷启动、排队等。

Sentinel 的设计理念是让您自由选择控制的角度，并进行灵活组合，从而达到想要的效果。

1、引入 Sentinel starter

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
```

2、接入限流埋点

- HTTP 埋点

Sentinel starter 默认为所有的 HTTP 服务提供了限流埋点，如果只想对 HTTP 服务进行限流，那么只需要引入依赖，无需修改代码。

- 自定义埋点

如果需要对某个特定的方法进行限流或降级，可以通过 `@SentinelResource` 注解来完成限流的埋点，示例代码如下：

```
@SentinelResource("resource")
public String hello() {
    return "Hello";
}
```

当然也可以通过原始的 `SphU.entry(xxx)` 方法进行埋点，可以参见 Sentinel 文档（<https://github.com/alibaba/Sentinel/wiki/%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8#%E5%AE%9A%E4%B9%89%E8%B5%84%E6%BA%90>）。

3、配置限流规则

Sentinel 提供了两种配置限流规则的方式：代码配置 和 控制台配置。

- 通过代码来实现限流规则的配置。一个简单的限流规则配置示例代码如下，更多限流规则配置详情请参考 Sentinel 文档。
(<https://github.com/alibaba/Sentinel/wiki/%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8#%E5%AE%9A%E4%B9%89%E8%A7%84%E5%88%99>)

```
List<FlowRule> rules = new ArrayList<FlowRule>();
FlowRule rule = new FlowRule();
rule.setResource(str);
// set limit qps to 10
rule.setCount(10);
rule.setGrade(RuleConstant.FLOW_GRADE_QPS);
rule.setLimitApp("default");
rules.add(rule);
```

```
FlowRuleManager.loadRules(rules);
```

- 通过控制台进行限流规则配置

1、下载控制台：

http://edas-public.oss-cn-hangzhou.aliyuncs.com/install_package/demo/sentinel-dashboard.jar

2、启动控制台，执行 Java 命令 `java -jar sentinel-dashboard.jar` 完成 Sentinel 控制台的启动。控制台默认的监听端口为 8080。

4、启动应用并配置

增加配置，在应用的 `/src/main/resources/application.properties` 中添加基本配置信息

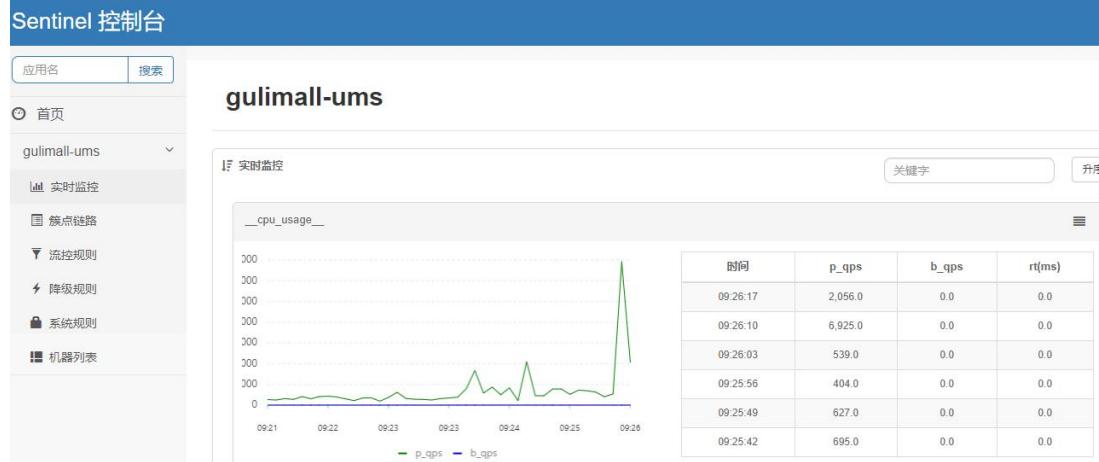
`spring.application.name=sentinel-example`

`server.port=18083`

`spring.cloud.sentinel.transport.dashboard=localhost:8080`

5、控制台配置限流规则并验证

访问 `http://localhost:8080` 页面。



时间	p_qps	b_qps	rt(ms)
09:26:17	2,056.0	0.0	0.0
09:26:10	6,925.0	0.0	0.0
09:26:03	539.0	0.0	0.0
09:25:56	404.0	0.0	0.0
09:25:49	627.0	0.0	0.0
09:25:42	695.0	0.0	0.0

如果您在控制台没有找到应用，请调用一下进行了 Sentinel 埋点的 URL 或方法，因为 Sentinel 使用了 `lazy load` 策略。

任意发送请求，可以在簇点链路里面看到刚才的请求，可以对请求进行流控；

新增流控规则

资源名	/ums/member/list		
流控应用	default		
阈值类型	<input checked="" type="radio"/> QPS <input type="radio"/> 线程数	单机阈值	1
流控模式	<input checked="" type="radio"/> 直接 <input type="radio"/> 关联 <input type="radio"/> 链路		
流控方式	<input checked="" type="radio"/> 快速失败 <input type="radio"/> Warm Up <input type="radio"/> 排队等待		

关闭高级选项

[新增并继续添加](#) [新增](#) [取消](#)

测试流控效果

localhost:14000/ums/member/list

Blocked by Sentinel (flow limiting)

6、自定义流控响应

```
@Configuration
public class SentinelConfig {

    public SentinelConfig(){
        WebCallbackManager.setUrlBlockHandler(new UrlBlockHandler(){
            @Override
            public void blocked(HttpServletRequest request, HttpServletResponse response,
                                BlockException ex) throws IOException {
                response.getWriter().write(s: "limited");
            }
        });
    }
}
```

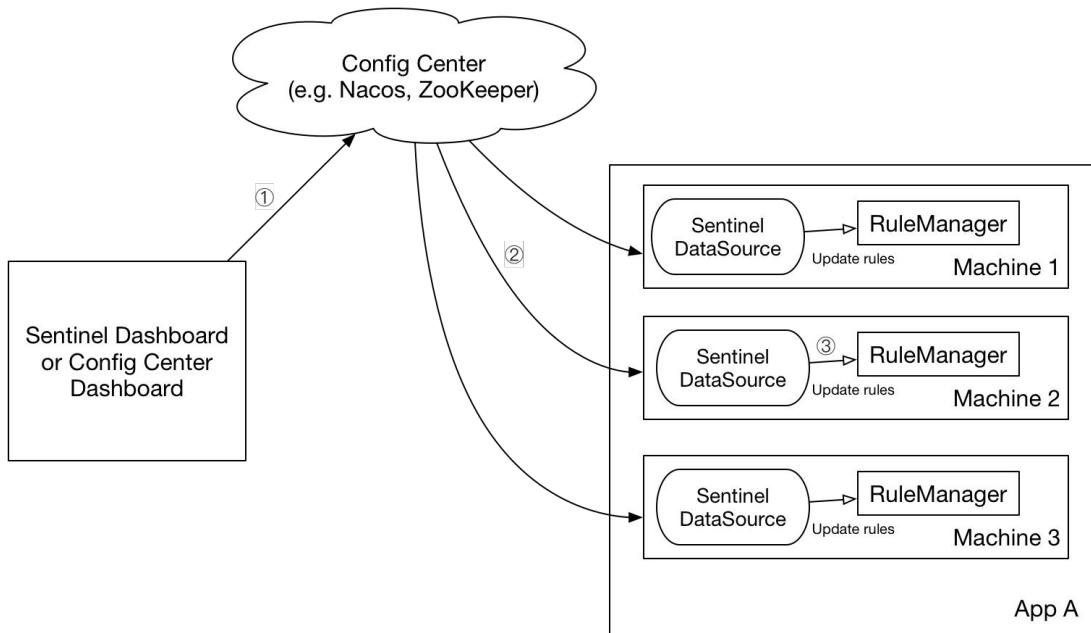
7、持久化流控规则

默认的流控规则是保存在项目的内存中，项目停止再启动，流控规则就是失效。我们可以持久化保存规则；

<https://github.com/alibaba/Sentinel/wiki/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%99%E6%89%A9%E5%B1%95#datasource-%E6%89%A9%E5%B1%95>

生产环境使用模式：

我们推荐通过控制台设置规则后将规则推送到统一的规则中心，客户端实现 `ReadableDataSource` 接口端监听规则中心实时获取变更，



解决方案：

DataSource 扩展常见的实现方式有：

- **拉模式：**客户端主动向某个规则管理中心定期轮询拉取规则，这个规则中心可以是 RDBMS、文件，甚至是 VCS 等。这样做的方式是简单，缺点是无法及时获取变更；
- **推模式：**规则中心统一推送，客户端通过注册监听器的方式时刻监听变化，比如使用 Nacos、Zookeeper 等配置中心。这种方式有更好的实时性和一致性保证。

推模式：使用 Nacos 配置规则

1、引入依赖

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
    <version>1.6.3</version>
</dependency>
```

2、编写配置类，

<https://github.com/alibaba/Sentinel/wiki/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%99%E6%89%A9%E5%B1%95#%E6%8E%A8%E6%A8%A1%E5%BC%8F%E4%BD%BF%E7%94%A8-nacos-%E9%85%8D%E7%BD%AE%E8%A7%84%E5%88%99>

```
@Configuration
public class SentinelConfig {

    public SentinelConfig(){
        //1、加载流控策略
        ReadableDataSource<String, List<FlowRule>> flowRuleDataSource = new
        NacosDataSource<>("127.0.0.1:8848", "demo", "sentinel",
            source -> JSON.parseObject(source, new
            TypeReference<List<FlowRule>>() {}));
        FlowRuleManager.register2Property(flowRuleDataSource.getProperty());
    }
}
```

```
//2、加载降级策略
ReadableDataSource<String, List<DegradeRule>> degradeRuleDataSource =
new NacosDataSource<>("127.0.0.1:8848", "demo", "sentinel",
    source -> JSON.parseObject(source, new
TypeReference<List<DegradeRule>>() {}));

DegradeRuleManager.register2Property(degradeRuleDataSource.getProperty());

//3、加载系统规则
ReadableDataSource<String, List<SystemRule>> systemRuleDataSource =
new NacosDataSource<>("127.0.0.1:8848", "demo", "sentinel",
    source -> JSON.parseObject(source, new
TypeReference<List<SystemRule>>() {}));

SystemRuleManager.register2Property(systemRuleDataSource.getProperty());

//4、加载权限策略
ReadableDataSource<String, List<AuthorityRule>>
authorityRuleDataSource = new NacosDataSource<>("127.0.0.1:8848", "demo",
"sentinel",
    source -> JSON.parseObject(source, new
TypeReference<List<AuthorityRule>>() {}));

AuthorityRuleManager.register2Property(authorityRuleDataSource.getProperty());
}

}
```

参照 <https://github.com/alibaba/Sentinel/wiki/Dynamic-Rule-Configuration> 查看更多控制规则

3、在 nacos 中创建 dataId，并使用 json 格式

The screenshot shows the Nacos configuration interface for creating a new data ID. The fields are as follows:

- * Data ID: sentinel
- * Group: demo
- 描述: null
- Beta发布: 默认不要勾选。
- 配置格式: TEXT JSON XML YAML HTML Properties

4、添加一条流控规则测试

```
[  
  {  
    "resource": "/ums/member/list",  
    "limitApp": "default",  
    "grade": 1,  
    "count": 5,  
    "strategy": 0,  
    "controlBehavior": 0,  
    "clusterMode": false  
  }  
]
```

配置含义说明：

<https://github.com/alibaba/Sentinel/wiki/%E6%B5%81%E9%87%8F%E6%8E%A7%E5%88%B6>

resource: 资源名，即限流规则的作用对象

count: 限流阈值

grade: 限流阈值类型（QPS 或并发线程数）

limitApp: 流控针对的调用来源，若为 default 则不区分调用来源

strategy: 调用关系限流策略

controlBehavior: 流量控制效果（直接拒绝、Warm Up、匀速排队）

5、系统规则，降级规则等均可添加

```
[  
  {  
    "resource": "/ums/member/list",  
    "limitApp": "default",  
    "grade": 1,  
    "count": 5,  
    "strategy": 0,  
    "controlBehavior": 0,  
    "clusterMode": false  
  },  
  {  
    "highestSystemLoad": -1,  
    "highestCpuUsage": 0.99,  
    "qps": 2,  
    "avgRt": 10,  
    "maxThread": 10  
  }  
]
```

6、最终效果

Sentinel 控制台改变流控规则，不能推送到 nacos 中，

Nacos 中改变流控规则可以实时观察到变化

系统规则	IP 地址	单机阈值	操作
RT	192.168.128.1:8720	10	<button>编辑</button> <button>删除</button>
线程数		10	<button>编辑</button> <button>删除</button>
QPS		2	<button>编辑</button> <button>删除</button>

第 2 步 API 的方式，可以直接变为配置方式；在 application.properties 中配置

```
spring.cloud.sentinel.datasource.ds.nacos.server-addr=127.0.0.1:8848
spring.cloud.sentinel.datasource.ds.nacos.data-id=sentinel
spring.cloud.sentinel.datasource.ds.nacos.group-id=demo
spring.cloud.sentinel.datasource.ds.nacos.rule-type=flow
```



```
spring.cloud.sentinel.datasource.ds1.nacos.server-addr=127.0.0.1:8848
spring.cloud.sentinel.datasource.ds1.nacos.data-id=sentinel
spring.cloud.sentinel.datasource.ds1.nacos.group-id=demo
spring.cloud.sentinel.datasource.ds1.nacos.rule-type=system
```

ds,ds1 是随便写的。

5、SpringCloud Alibaba-Seata

1、简介

6、SpringCloud Alibaba-OSS

1、简介

对象存储服务（Object Storage Service, OSS）是一种海量、安全、低成本、高可靠的云存储服务，适合存放任意类型的文件。容量和处理能力弹性扩展，多种存储类型供选择，全面优化存储成本。

2、使用步骤

1、开通阿里云对象存储服务



<https://www.aliyun.com/product/oss>

2、引入 SpringCloud Alibaba-OSS

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alicloud-oss</artifactId>
</dependency>
```

3、配置阿里云 oss 相关的账号信息

```
spring:
  cloud:
    alicloud:
      oss:
        endpoint: oss-cn-shanghai.aliyuncs.com
        access-key: xxxxxxxx
        secret-key: xxxxxxxx
```

注意：必须申请 RAM 账号信息，并且分配 OSS 操作权限

4、测试使用 OssClient 上传

```
@Autowired  
OSSClient ossClient;  
  
@Test  
public void contextLoads2() throws FileNotFoundException {  
  
    InputStream inputStream = new  
FileInputStream("C:\\\\Users\\\\lfy\\\\Pictures\\\\bug.jpg");  
    ossClient.putObject("gulimall", "aaa/bug222.jpg", inputStream);  
    System.out.println("ok");  
}
```

二、SpringCloud

1、Feign 声明式远程调用

1、简介

Feign 是一个声明式的 HTTP 客户端，它的目的就是让远程调用更加简单。Feign 提供了 HTTP 请求的模板，通过编写简单的接口和插入注解，就可以定义好 HTTP 请求的参数、格式、地址等信息。

Feign 整合了 **Ribbon**（负载均衡）和 **Hystrix**（服务熔断），可以让我们不再需要显式地使用这两个组件。

SpringCloudFeign 在 NetflixFeign 的基础上扩展了对 SpringMVC 注解的支持，在其实现下，我们只需创建一个接口并用注解的方式来配置它，即可完成对服务提供方的接口绑定。简化了 SpringCloudRibbon 自行封装服务调用客户端的开发量。

2、使用

1、引入依赖

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-openfeign</artifactId>
```



```
</dependency>
```

2、开启 feign 功能

```
@EnableFeignClients(basePackages = "com.atguigu.gulimall.pms.feign")
```

3、声明远程接口

```
@FeignClient("gulimall-ware")
```

```
public interface WareFeignService {
```

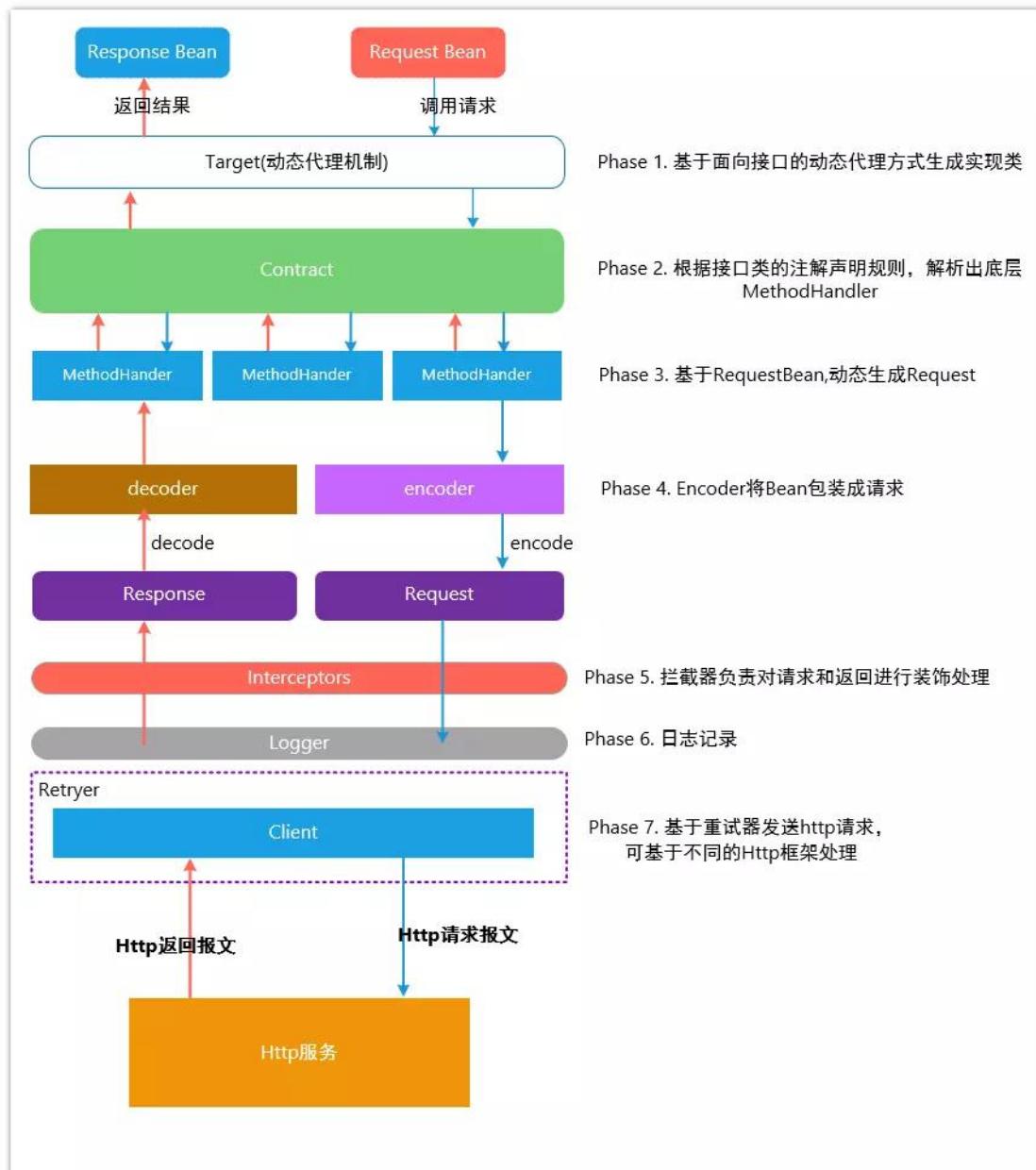
```
    @PostMapping("/ware/waresku/skus")
```

```
    public Resp<List<SkuStockVo>> skuWareInfos(@RequestBody List<Long> skuids);
```

```
}
```



3、原理



2、Gateway

1、简介

网关作为流量的入口，常用功能包括路由转发、权限校验、限流控制等。而 springcloud gateway 作为 SpringCloud 官方推出的第二代网关框架，取代了 Zuul 网关。

组件	RPS(request per second)
Spring Cloud Gateway	Requests/sec: 32213.38
Zuul	Requests/sec: 20800.13
Linkerd	Requests/sec: 28050.76

网关提供 API 全托管服务，丰富的 API 管理功能，辅助企业管理大规模的 API，以降低管理成本和安全风险，包括协议适配、协议转发、安全策略、防刷、流量、监控日志等功能。

Spring Cloud Gateway 旨在提供一种简单而有效的方式来对 API 进行路由，并为他们提供切面，例如：安全性，监控/指标 和 弹性等。

官方文档地址：

<https://cloud.spring.io/spring-cloud-static/spring-cloud-gateway/2.1.3.RELEASE/single/spring-cloud-gateway.html>

Spring Cloud Gateway 特点：

- 基于 Spring5，支持响应式编程和 SpringBoot2.0
- 支持使用任何请求属性进行路由匹配
- 特定于路由的断言和过滤器
- 集成 Hystrix 进行断路保护
- 集成服务发现功能
- 易于编写 Predicates 和 Filters
- 支持请求速率限制
- 支持路径重写

思考：

为什么使用 API 网关？

API 网关出现的原因是微服务架构的出现，不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求，如果让客户端直接与各个微服务通信，会有以下的问题：

- 客户端会多次请求不同的微服务，增加了客户端的复杂性。
- 存在跨域请求，在一定场景下处理相对复杂。
- 认证复杂，每个服务都需要独立认证。
- 难以重构，随着项目的迭代，可能需要重新划分微服务。例如，可能将多个服务合并在一个或者将一个服务拆分成多个。如果客户端直接与微服务通信，那么重构将会很难实施。
- 某些微服务可能使用了防火墙 / 浏览器不友好的协议，直接访问会有一定的困难。

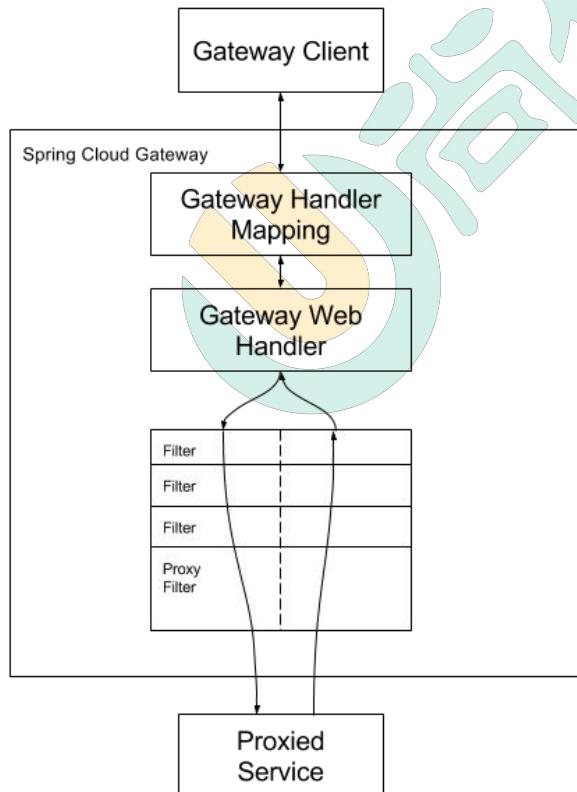
以上这些问题可以借助 API 网关解决。API 网关是介于客户端和服务器端之间的中间层，所有的外部请求都会先经过 API 网关这一层。也就是说，API 的实现方面更多的考虑业务逻辑，而安全、性能、监控可以交由 API 网关来做，这样既提高业务灵活性又不失安全性：使用 API 网关后的优点如下：

- 易于监控。可以在网关收集监控数据并将其推送到外部系统进行分析。
- 易于认证。可以在网关上进行认证，然后再将请求转发到后端的微服务，而无须在每个微服务中进行认证。
- 减少了客户端与各个微服务之间的交互次数。

2、核心概念

- 路由。路由是网关最基础的部分，路由信息有一个 ID、一个目的 URL、一组断言和一组 Filter 组成。如果断言路由为真，则说明请求的 URL 和配置匹配
- 断言。Java8 中的断言函数。Spring Cloud Gateway 中的断言函数输入类型是 Spring5.0 框架中的 ServerWebExchange。Spring Cloud Gateway 中的断言函数允许开发者去定义匹配来自于 http request 中的任何信息，比如请求头和参数等。
- 过滤器。一个标准的 Spring webFilter。Spring cloud gateway 中的 filter 分为两种类型的 Filter，分别是 Gateway Filter 和 Global Filter。过滤器 Filter 将会对请求和响应进行修改处理

工作原理：



客户端发送请求给网关，弯管 HandlerMapping 判断是否请求满足某个路由，满足就发给网关的 WebHandler。这个 WebHandler 将请求交给一个过滤器链，请求到达目标服务之前，会执行所有过滤器的 pre 方法。请求到达目标服务处理之后再依次执行所有过滤器的 post 方法。

一句话：满足某些断言（predicates）就路由到指定的地址（uri），使用指定的过滤器（filter）

3、使用

1、HelloWorld

1、创建网关项目，引入网关

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

2、编写网关配置文件

```
spring:
  cloud:
    gateway:
      routes:
        - id: add_request_parameter_route
          uri: https://example.org
          predicates:
            - Query=baz
          filters:
            - AddRequestParameter=foo, bar
```

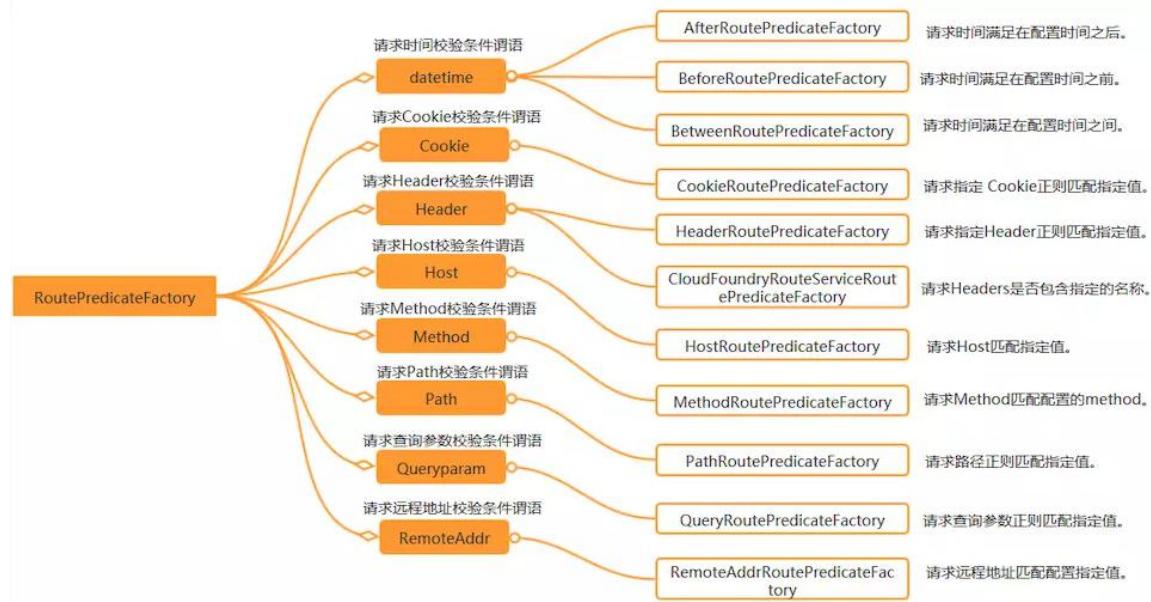
3、注意

- 各种 Predicates 同时存在于同一个路由时，请求必须同时满足所有的条件才被这个路由匹配。
- 一个请求满足多个路由的谓词条件时，请求只会被首个成功匹配的路由转发

4、测试

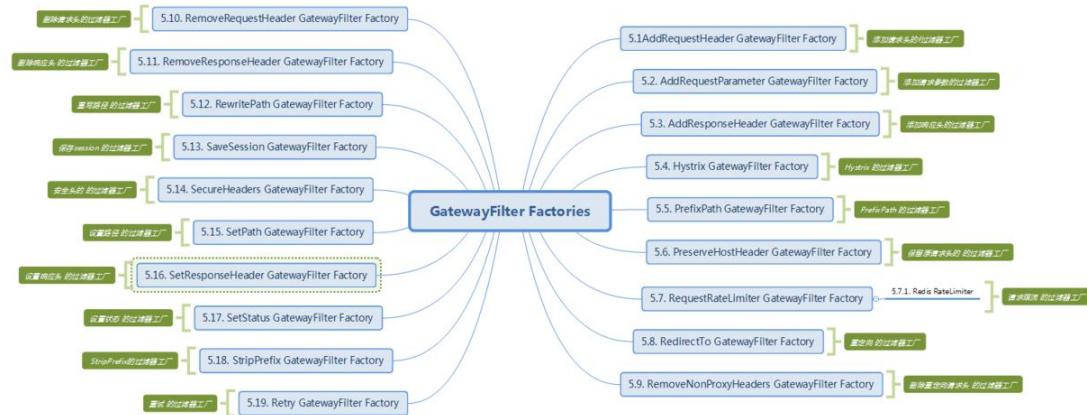
可以使用 postman 进行测试网关的路由功能

2、断言 (Predicates)



3、过滤器 (filters)

1、GatewayFilter



2、GlobalFilter



3、Sleuth+Zipkin 服务链路追踪

1、为什么用

微服务架构是一个分布式架构，它按业务划分服务单元，一个分布式系统往往有很多个服务单元。由于服务单元数量众多，业务的复杂性，如果出现了错误和异常，很难去定位。主要体现在，一个请求可能需要调用很多个服务，而内部服务的调用复杂性，决定了问题难以定位。所以微服务架构中，必须实现分布式链路追踪，去跟进一个请求到底有哪些服务参与，参与的顺序又是怎样的，从而达到每个请求的步骤清晰可见，出了问题，很快定位。

链路追踪组件有 Google 的 Dapper，Twitter 的 Zipkin，以及阿里的 Eagleeye（鹰眼）等，它们都是非常优秀的链路追踪开源组件。

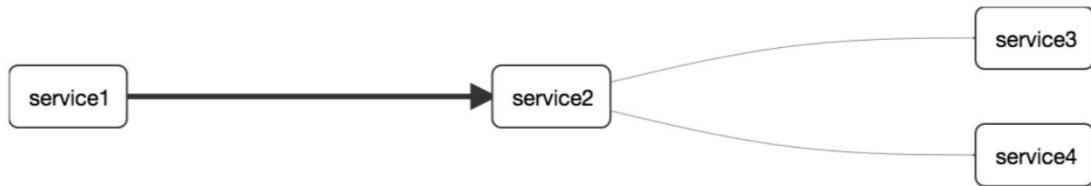
2、基本术语

- **Span (跨度)**：基本工作单元，发送一个远程调度任务 就会产生一个 Span，Span 是一个 64 位 ID 唯一标识的，Trace 是用另一个 64 位 ID 唯一标识的，Span 还有其他数据信息，比如摘要、时间戳事件、Span 的 ID、以及进度 ID。
- **Trace (跟踪)**：一系列 Span 组成的一个树状结构。请求一个微服务系统的 API 接口，这个 API 接口，需要调用多个微服务，调用每个微服务都会产生一个新的 Span，所有由这个请求产生的 Span 组成了这个 Trace。
- **Annotation (标注)**：用来及时记录一个事件的，一些核心注解用来定义一个请求的开始和结束。这些注解包括以下：
 - **cs - Client Sent** -客户端发送一个请求，这个注解描述了这个 Span 的开始
 - **sr - Server Received** -服务端获得请求并准备开始处理它，如果将其 sr 减去 cs 时间戳便可得到网络传输的时间。
 - **ss - Server Sent** （服务端发送响应）-该注解表明请求处理的完成(当请求返回客户端)，如果 ss 的时间戳减去 sr 时间戳，就可以得到服务器请求的时间。
 - **cr - Client Received** （客户端接收响应）-此时 Span 的结束，如果 cr 的时间戳减去 cs 时间戳便可以得到整个请求所消耗的时间。

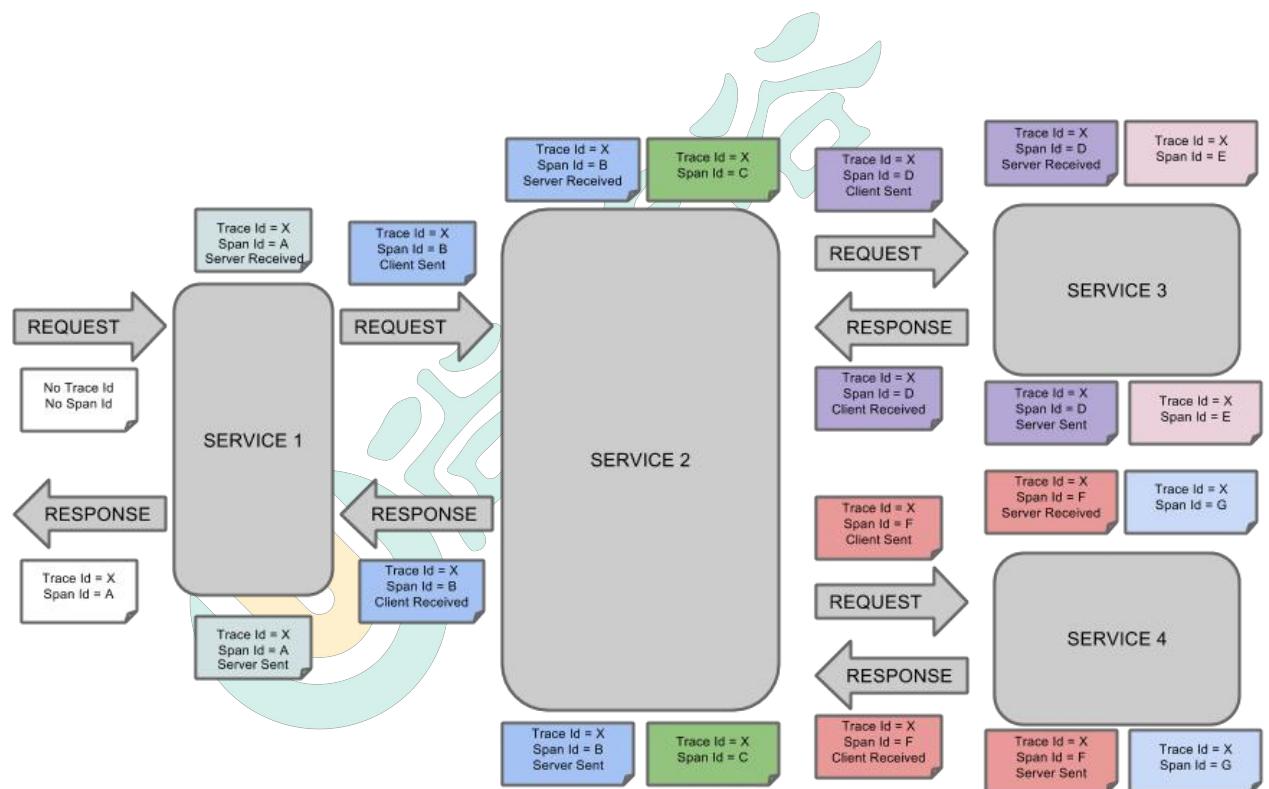
官方文档：

<https://cloud.spring.io/spring-cloud-static/spring-cloud-sleuth/2.1.3.RELEASE/single/spring-cloud-sleuth.html>

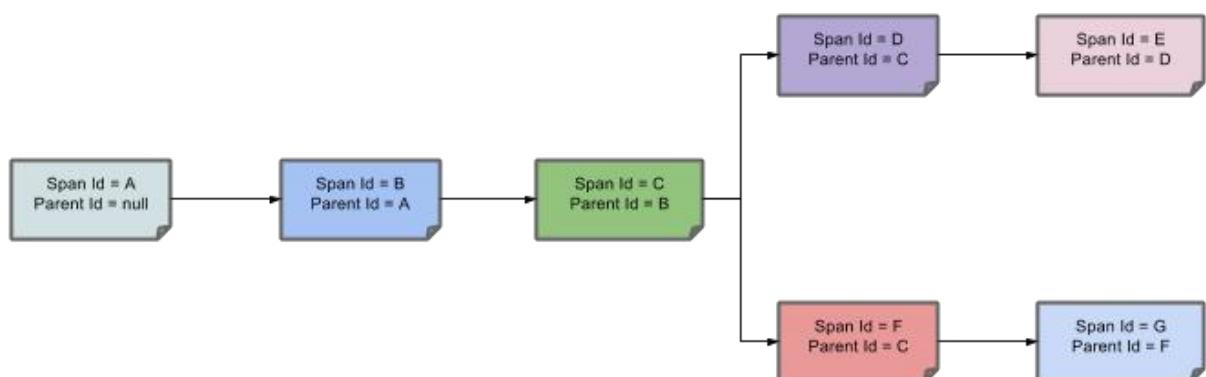
如果服务调用顺序如下



那么用以上概念完整的表示出来如下：



Span 之间的父子关系如下：



3、整合 Sleuth

1、服务提供者与消费者导入依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

2、打开 debug 日志

logging:

level:

```
    org.springframework.cloud.openfeign: debug
    org.springframework.cloud.sleuth: debug
```

3、发起一次远程调用，观察控制台

```
DEBUG [user-service,541450f08573fff5,541450f08573fff5,false]
```

user-service: 服务名

541450f08573fff5: 是 Tranceld，一条链路中，只有一个 Tranceld

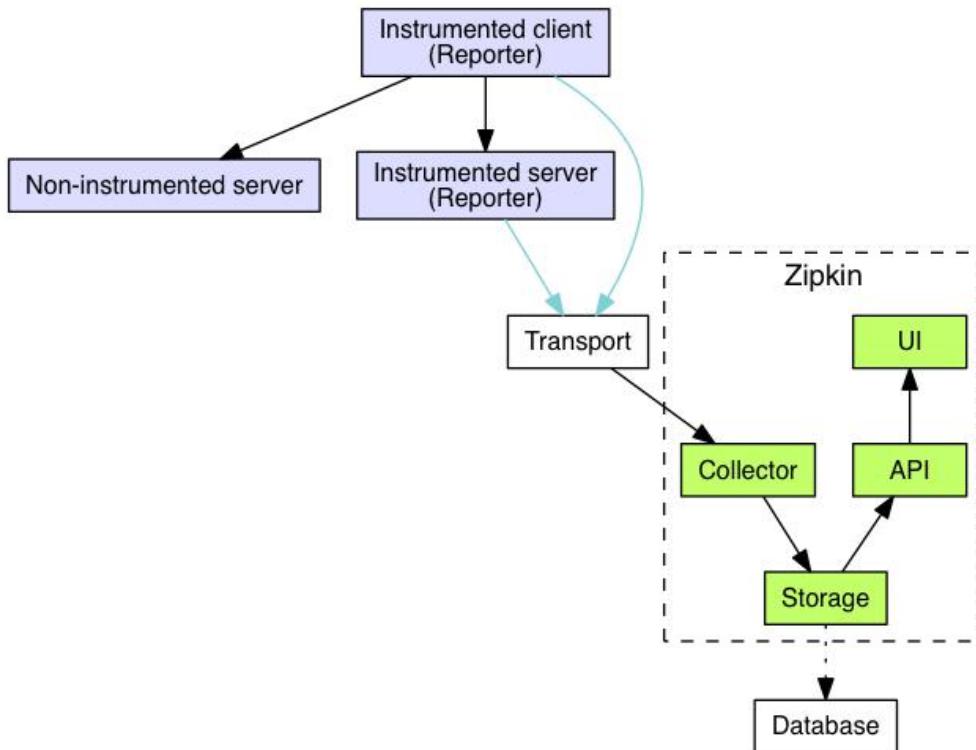
541450f08573fff5: 是 spanId，链路中的基本工作单元 id

false: 表示是否将数据输出到其他服务，true 则会把信息输出到其他可视化的服务上观察

5、整合 zipkin 可视化观察

通过 Sleuth 产生的调用链监控信息，可以得知微服务之间的调用链路，但监控信息只输出到控制台不方便查看。我们需要一个图形化的工具-zipkin。Zipkin 是 Twitter 开源的分布式跟踪系统，主要用来收集系统的时序数据，从而追踪系统的调用问题。zipkin 官网地址如下：

<https://zipkin.io/>



1、docker 安装 zipkin 服务器

```
docker run -d -p 9411:9411 openzipkin/zipkin
```

2、导入

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

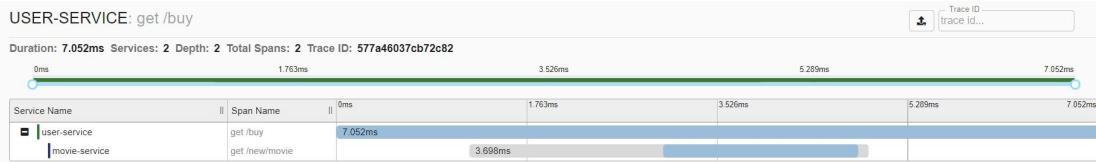
zipkin 依赖也同时包含了 sleuth，可以省略 sleuth 的引用

3、添加 zipkin 相关配置

```
spring:
  application:
    name: user-service
  zipkin:
    base-url: http://192.168.56.10:9411/ # zipkin 服务器的地址
    # 关闭服务发现，否则 Spring Cloud 会把 zipkin 的 url 当做服务名称
    discoveryClientEnabled: false
  sender:
    type: web # 设置使用 http 的方式传输数据
  sleuth:
    sampler:
      probability: 1 # 设置抽样采集率为 100%，默认为 0.1，即 10%
```

发送远程请求，测试 zipkin。

服务调用链追踪信息统计



服务依赖信息统计



5、Zipkin 数据持久化

Zipkin 默认是将监控数据存储在内存的，如果 Zipkin 挂掉或重启的话，那么监控数据就会丢失。所以如果想要搭建生产可用的 Zipkin，就需要实现监控数据的持久化。而想要实现数据持久化，自然就是得将数据存储至数据库。好在 Zipkin 支持将数据存储至：

- 内存（默认）
- MySQL
- Elasticsearch
- Cassandra

Zipkin 数据持久化相关的官方文档地址如下：

<https://github.com/openzipkin/zipkin#storage-component>

Zipkin 支持的这几种存储方式中，内存显然是不适用于生产的，这一点开始也说了。而使用 MySQL 的话，当数据量大时，查询较为缓慢，也不建议使用。Twitter 官方使用的是 Cassandra 作为 Zipkin 的存储数据库，但国内大规模用 Cassandra 的公司较少，而且 Cassandra 相关文档也不多。

综上，故采用 Elasticsearch 是个比较好的选择，关于使用 Elasticsearch 作为 Zipkin 的存储数据库的官方文档如下：

elasticsearch-storage:

<https://github.com/openzipkin/zipkin/tree/master/zipkin-server#elasticsearch-storage>

zipkin-storage/elasticsearch

<https://github.com/openzipkin/zipkin/tree/master/zipkin-storage/elasticsearch>

通过 docker 的方式

```
docker run --env STORAGE_TYPE=elasticsearch --env ES_HOSTS=192.168.190.129:9200
openzipkin/zipkin-dependencies
```

环境变量	
STORAGE_TYPE	指定存储类型，可选项为elasticsearch、mysql、cassandra等，详见： https://github.com/openzipkin/zipkin/tree/master/zipkin-server#environment-variables
ES_HOSTS	Elasticsearch地址，多个使用 , 分隔。默认 <code>http://localhost:9200</code>
ES_PIPELINE	指定span被索引之前的pipeline (pipeline是Elasticsearch的概念)
ES_TIMEOUT	连接Elasticsearch的超时时间，单位是毫秒；默认10000 (10秒)
ES_INDEX	Zipkin所使用的索引 (Zipkin会每天建索引) 前缀，默认是 <code>zipkin</code>
ES_DATE_SEPARATOR	Zipkin建立索引的日期分隔符，默认是 -
ES_INDEX_SHARDS	shard (shard是Elasticsearch的概念) 个数，默认5
ES_INDEX_REPLICAS	副本 (replica是Elasticsearch的概念) 个数，默认1
ES_USERNAME/ES_PASSWORD	Elasticsearch账号密码
ES_HTTP_LOGGING	控制Elasticsearch Api的日志级别，可选项为BASIC、HEADERS、BODY

@51CTO博客

使用 es 时 Zipkin Dependencies 支持的环境变量

环境变量	含义
STORAGE_TYPE	指定存储类型，可选项为elasticsearch、mysql、cassandra等，详见： https://github.com/openzipkin/zipkin/tree/master/zipkin-server#environment-variables
ES_INDEX	生成每日索引名称时使用的索引前缀。默认为"zipkin"。
ES_DATE_SEPARATOR	在索引中生成日期时使用的分隔符。默认为'-'，所以查询的索引看起来像 <code>zipkin-yyyy-DD-mm</code> ，可以改为"."，这样查询索引就变成 <code>zipkin-yyyy.MM.dd</code> 。示例： <code>ES_DATE_SEPARATOR=.</code>
ES_HOSTS	ElasticSearch主机列表，多个主机使用逗号分隔。默认为 <code>localhost:9200</code>
ES_NODES_WAN_ONLY	如设为true，则表示仅使用ES_HOSTS所设置的值，默认为false。当ElasticSearch集群运行在Docker中时，可将该环境变量设为true

@51CTO博客

谷粒商城

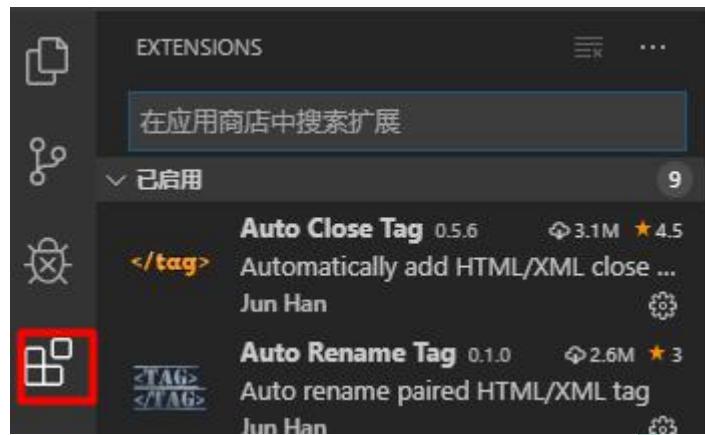
前端开发基础知识&快速入门



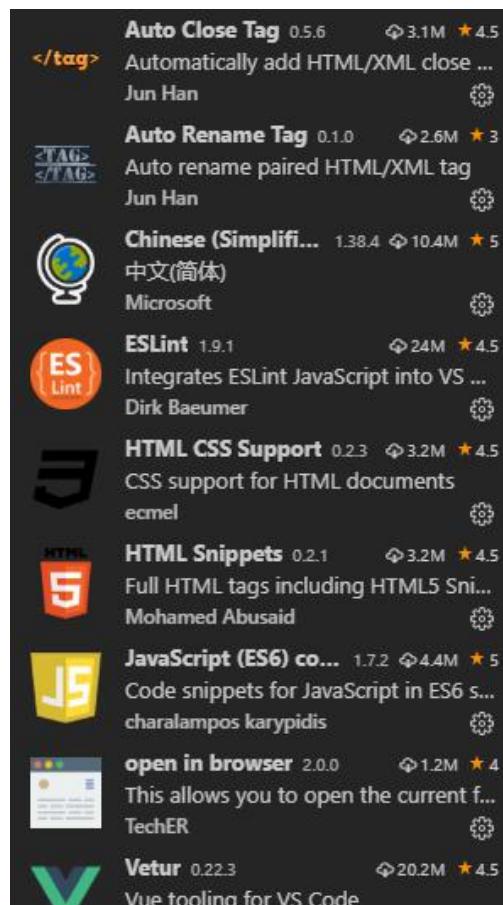
一、VSCode 使用

1、安装常用插件

切换到插件标签页



安装以下基本插件



2、创建项目

vscode 很轻量级，本身没有新建项目的选项，创建一个空文件夹就可以当做一个项目

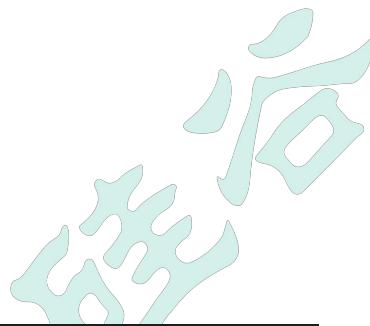


3、创建网页

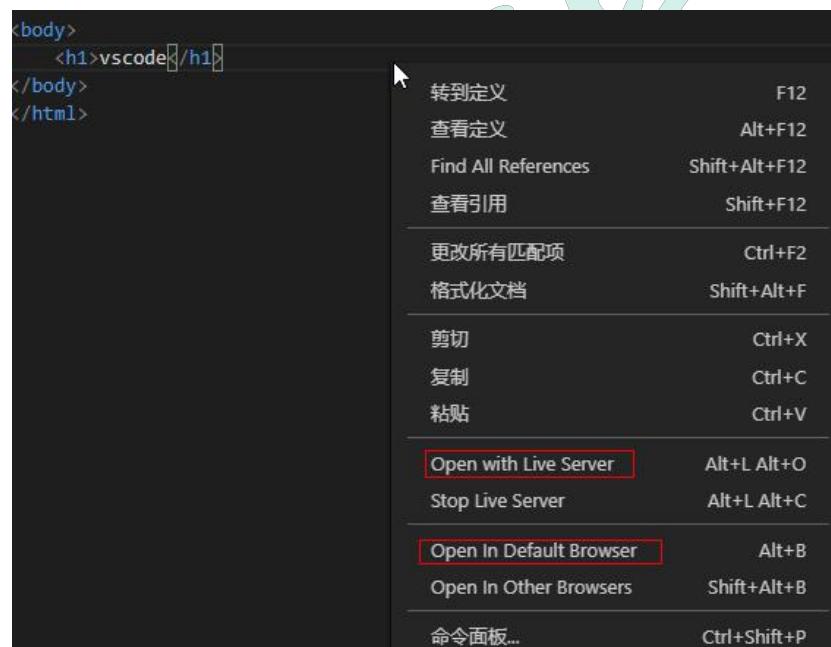
创建文件，命名为 index.html

快捷键 !，快速创建网页模板

h1 + 回车，自动补全标签



4、运行效果

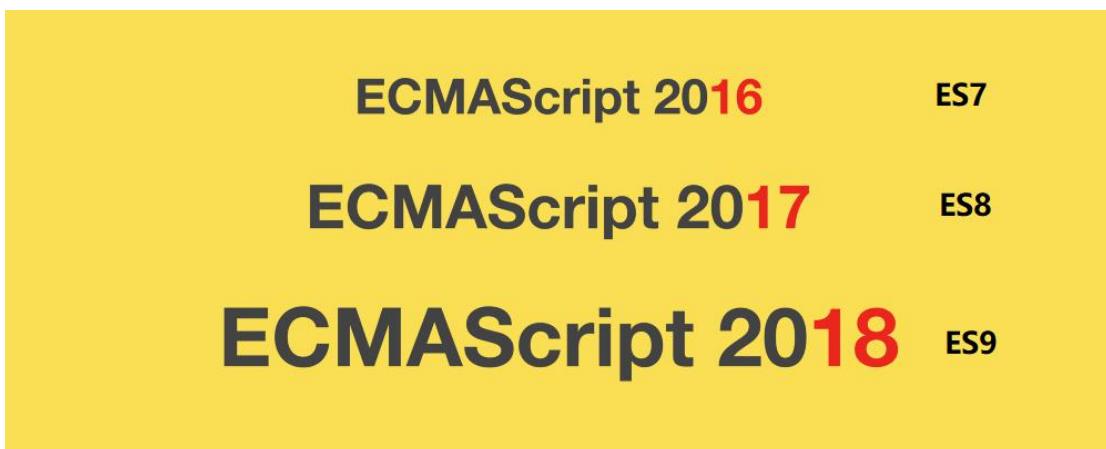


如果使用 live server，页面内容变化，保存以后，浏览器会自动变化；

二、ES6

1、简介

ECMAScript 6.0 (以下简称 ES6, ECMAScript 是一种由 Ecma 国际(前身为欧洲计算机制造商协会,英文名称是 European Computer Manufacturers Association)通过 ECMA-262 标准化的脚本程序设计语言) **是 JavaScript 语言的下一代标准**, 已经在 2015 年 6 月正式发布了, 并且从 ECMAScript 6 开始, 开始采用年号来做版本。即 ECMAScript 2015, 就是 ECMAScript6。它的目标, 是使得 JavaScript 语言可以用来编写复杂的大型应用程序, 成为企业级开发语言。**每年一个新版本。**



2、什么是 ECMAScript

来看下前端的发展历程:

- **web1.0 时代:**

最初的网页以 HTML 为主, 是纯静态的网页。网页是只读的, 信息流只能从服务的到客户端单向流通。开发人员也只关心页面的样式和内容即可。

- **web2.0 时代:**

- 1995 年, 网景工程师 Brendan Eich 花了 10 天时间设计了 JavaScript 语言。
- 1996 年, 微软发布了 JScript, 其实是 JavaScript 的逆向工程实现。
- 1996 年 11 月, JavaScript 的创造者 Netscape 公司, 决定将 JavaScript 提交给标准化组织 ECMA, 希望这种语言能够成为国际标准。
- 1997 年, ECMA 发布 262 号标准文件 (ECMA-262) 的第一版, 规定了浏览器脚本语言的标准, 并将这种语言称为 ECMAScript, 这个版本就是 1.0 版。JavaScript 和 JScript 都是 'ECMAScript' 的标准实现者, 随后各大浏览器厂商纷纷实现了 'ECMAScript' 标准。

所以，**ECMAScript** 是浏览器脚本语言的规范，而各种我们熟知的 **js** 语言，如 **JavaScript** 则是规范的具体实现。

3、ES6 新特性

1、let 声明变量

```
// var 声明的变量往往会越域
// let 声明的变量有严格局部作用域
{
    var a = 1;
    let b = 2;
}
console.log(a); // 1
console.log(b); // ReferenceError: b is not defined

// var 可以声明多次
// let 只能声明一次
var m = 1
var m = 2
let n = 3
// let n = 4
console.log(m) // 2
console.log(n) // Identifier 'n' has already been declared

// var 会变量提升
// let 不存在变量提升
console.log(x); // undefined
var x = 10;
console.log(y); //ReferenceError: y is not defined
let y = 20;
```

2、const 声明常量（只读变量）

```
// 1. 声明之后不允许改变
// 2. 一旦声明必须初始化，否则会报错
const a = 1;
a = 3; //Uncaught TypeError: Assignment to constant variable.
```

3、解构表达式

1) 、数组解构

```
let arr = [1,2,3];
//以前我们想获取其中的值，只能通过角标。ES6 可以这样：
const [x,y,z] = arr;// x, y, z 将与 arr 中的每个位置对应来取值
// 然后打印
console.log(x,y,z);
```

2) 、对象解构

```
const person = {
    name: "jack",
    age: 21,
    language: ['java', 'js', 'css']
}
// 解构表达式获取值，将 person 里面每一个属性和左边对应赋值
const { name, age, language } = person;
// 等价于下面
// const name = person.name;
// const age = person.age;
// const language = person.language;
// 可以分别打印
console.log(name);
console.log(age);
console.log(language);
```

```
//扩展：如果想要将 name 的值赋值给其他变量，可以如下，nn 是新的变量名
const { name: nn, age, language } = person;
console.log(nn);
console.log(age);
console.log(language);
```

4、字符串扩展

1) 、几个新的 API

ES6 为字符串扩展了几个新的 API:

- `includes()`：返回布尔值，表示是否找到了参数字符串。
- `startsWith()`：返回布尔值，表示参数字符串是否在原字符串的头部。
- `endsWith()`：返回布尔值，表示参数字符串是否在原字符串的尾部。

```
let str = "hello.vue";
console.log(str.startsWith("hello")); //true
console.log(str.endsWith(".vue")); //true
console.log(str.includes("e")); //true
console.log(str.includes("hello")); //true
```

2) 、字符串模板

模板字符串相当于加强版的字符串，用反引号`除了作为普通字符串，还可以用来定义多行字符串，还可以在字符串中加入变量和表达式。

```
// 1、多行字符串
let ss = `
<div>
    <span>hello world</span>
</div>
`
```



// 2、字符串插入变量和表达式。变量名写在 \${} 中，\${} 中可以放入 JavaScript 表达式。

```
let name = "张三";
let age = 18;
let info = `我是${name}，今年${age}了`;
console.log(info)
```

```
// 3、字符串中调用函数
function fun() {
    return "这是一个函数"
}
let sss = `0(n_n)0 哈哈~, ${fun()}`;
console.log(sss); // 0(n_n)0 哈哈~, 这是一个函数
```

5、函数优化

1)、函数参数默认值

```
//在 ES6 以前，我们无法给一个函数参数设置默认值，只能采用变通写法：  
function add(a, b) {  
    // 判断 b 是否为空，为空就给默认值 1  
    b = b || 1;  
    return a + b;  
}  
// 传一个参数  
console.log(add(10));  
  
//现在可以这么写：直接给参数写上默认值，没传就会自动使用默认值  
function add2(a, b = 1) {  
    return a + b;  
}  
// 传一个参数  
console.log(add2(10));
```

2)、不定参数

不定参数用来表示不确定参数个数，形如，...变量名，由...加上一个具名参数标识符组成。
具名参数只能放在参数列表的最后，并且有且只有一个不定参数

```
function fun(...values) {  
    console.log(values.length)  
}  
fun(1, 2)      //2  
fun(1, 2, 3, 4) //4
```

3)、箭头函数

ES6 中定义函数的简写方式

- 一个参数时：

```
//以前声明一个方法
// var print = function (obj) {
//   console.log(obj);
// }
// 可以简写为:
var print = obj => console.log(obj);
// 测试调用
print(100);
```

- 多个参数:

```
// 两个参数的情况:
var sum = function (a, b) {
  return a + b;
}
// 简写为:
//当只有一行语句，并且需要返回结果时，可以省略 {}，结果会自动返回。
var sum2 = (a, b) => a + b;
//测试调用
console.log(sum2(10, 10));//20

// 代码不止一行，可以用`{}`括起来
var sum3 = (a, b) => {
  c = a + b;
  return c;
};
//测试调用
console.log(sum3(10, 20));//30
```

4)、实战：箭头函数结合解构表达式

```
//需求，声明一个对象，hello 方法需要对象的个别属性
//以前的方式:
const person = {
  name: "jack",
  age: 21,
  language: ['java', 'js', 'css']
}

function hello(person) {
  console.log("hello," + person.name)
```

```
    }
    //现在的方式
    var hello2 = ({ name }) => { console.log("hello," + name) };
    //测试
    hello2(person);
```

6、对象优化

1)、新增的 API

ES6 给 Object 拓展了许多新的方法，如：

- `Object.keys(obj)`: 获取对象的所有 key 形成的数组
- `Object.values(obj)`: 获取对象的所有 value 形成的数组
- `Object.entries(obj)`: 获取对象的所有 key 和 value 形成的二维数组。格式: `[[k1,v1],[k2,v2],...]`
- `Object.assign(dest, ...src)` : 将多个 src 对象的值 拷贝到 dest 中。（第一层为深拷贝，第二层为浅拷贝）

```
const person = {
  name: "jack",
  age: 21,
  language: ['java', 'js', 'css']
}

console.log(Object.keys(person)); //["name", "age", "language"]
console.log(Object.values(person)); //["jack", 21, Array(3)]
console.log(Object.entries(person)); //[[name, "jack"], [age, 21], [language, ["java", "js", "css"]]]
```

```
const target = { a: 1 };
const source1 = { b: 2 };
const source2 = { c: 3 };

//Object.assign 方法的第一个参数是目标对象，后面的参数都是源对象。
Object.assign(target, source1, source2);
console.log(target) // {a: 1, b: 2, c: 3}
```

2)、声明对象简写

```
const age = 23
const name = "张三"
```

```
// 传统
const person1 = { age: age, name: name }
console.log(person1)

// ES6: 属性名和属性值变量名一样，可以省略
const person2 = { age, name }
console.log(person2) // {age: 23, name: "张三"}
```

3)、对象的函数属性简写

```
let person = {
  name: "jack",
  // 以前:
  eat: function (food) {
    console.log(this.name + "在吃" + food);
  },
  // 箭头函数版: 这里拿不到 this
  eat2: food => console.log(person.name + "在吃" + food),
  // 简写版:
  eat3(food) {
    console.log(this.name + "在吃" + food);
  }
}
person.eat("apple");
```



4)、对象拓展运算符

拓展运算符 (...) 用于取出参数对象所有可遍历属性然后拷贝到当前对象。

```
// 1、拷贝对象（深拷贝）
let person1 = { name: "Amy", age: 15 }
let someone = { ...person1 }
console.log(someone) // {name: "Amy", age: 15}

// 2、合并对象
let age = { age: 15 }
let name = { name: "Amy" }
let person2 = { ...age, ...name } // 如果两个对象的字段名重复, 后面对象字段值会覆盖前面对象的字段值
console.log(person2) // {age: 15, name: "Amy"}
```

7、map 和 reduce

数组中新增了 map 和 reduce 方法。

1) 、map

map(): 接收一个函数，将原数组中的所有元素用这个函数处理后放入新数组返回。

```
let arr = ['1', '20', '-5', '3'];
console.log(arr)

arr = arr.map(s => parseInt(s));
console.log(arr)
```

2) 、reduce

语法：

```
arr.reduce(callback,[initialValue])
```

reduce 为数组中的每一个元素依次执行回调函数，不包括数组中被删除或从未被赋值的元素，接受四个参数：初始值（或者上一次回调函数的返回值），当前元素值，当前索引，调用 reduce 的数组。

callback （执行数组中每个值的函数，包含四个参数）

- 1、previousValue （上一次调用回调返回的值，或者是提供的初始值（initialValue））
- 2、currentValue （数组中当前被处理的元素）
- 3、index （当前元素在数组中的索引）
- 4、array （调用 reduce 的数组）

initialValue （作为第一次调用 callback 的第一个参数。）

示例：

```
const arr = [1,20,-5,3];
//没有初始值:
console.log(arr.reduce((a,b)=>a+b)); //19
console.log(arr.reduce((a,b)=>a*b)); // -300
```

```
//指定初始值:  
console.log(arr.reduce((a,b)=>a+b,1)); //20  
console.log(arr.reduce((a,b)=>a*b,0)); // -0
```

8、Promise

在 JavaScript 的世界中，所有代码都是单线程执行的。由于这个“缺陷”，导致 JavaScript 的所有网络操作，浏览器事件，都必须是异步执行。异步执行可以用回调函数实现。一旦有一连串的 ajax 请求 a,b,c,d... 后面的请求依赖前面的请求结果，就需要层层嵌套。这种缩进和层层嵌套的方式，非常容易造成上下文代码混乱，我们不得不非常小心翼翼处理内层函数与外层函数的数据，一旦内层函数使用了上层函数的变量，这种混乱程度就会加剧.....总之，这种‘层叠上下文’的层层嵌套方式，着实增加了神经的紧张程度。

案例：用户登录，并展示该用户的各科成绩。在页面发送两次请求：

1. 查询用户，查询成功说明可以登录
2. 查询用户成功，查询科目
3. 根据科目的查询结果，获取去成绩

分析：此时后台应该提供三个接口，一个提供用户查询接口，一个提供科目的接口，一个提供各科成绩的接口，为了渲染方便，最好响应 json 数据。在这里就不编写后台接口了，而是提供三个 json 文件，直接提供 json 数据，模拟后台接口：

```
user.json:  
{  
  "id": 1,  
  "name": "zhangsan",  
  "password": "123456"  
}
```

```
user_corse_1.json:  
{  
  "id": 10,  
  "name": "chinese"  
}
```

```
corse_score_10.json:  
{  
  "id": 100,  
  "score": 90  
}
```

```
//回调函数嵌套的噩梦：层层嵌套。
```

```
$.ajax({
    url: "mock/user.json",
    success(data) {
        console.log("查询用户: ", data);
        $.ajax({
            url: `mock/user_corse_${data.id}.json`,
            success(data) {
                console.log("查询到课程: ", data);
                $.ajax({
                    url: `mock/corse_score_${data.id}.json`,
                    success(data) {
                        console.log("查询到分数: ", data);
                    },
                    error(error) {
                        console.log("出现异常了: " + error);
                    }
                });
            },
            error(error) {
                console.log("出现异常了: " + error);
            }
        });
    },
    error(error) {
        console.log("出现异常了: " + error);
    }
});
```

我们可以通过 Promise 解决以上问题。

1) 、Promise 语法

```
const promise = new Promise(function (resolve, reject) {
    // 执行异步操作
    if /* 异步操作成功 */ {
        resolve(value); // 调用 resolve, 代表 Promise 将返回成功的结果
    } else {
        reject(error); // 调用 reject, 代表 Promise 会返回失败结果
    }
});
```

使用箭头函数可以简写为：

```
const promise = new Promise((resolve, reject) =>{
    // 执行异步操作
    if (*异步操作成功*) {
        resolve(value); // 调用 resolve, 代表 Promise 将返回成功的结果
    } else {
        reject(error); // 调用 reject, 代表 Promise 会返回失败结果
    }
});
```

这样，在 promise 中就封装了一段异步执行的结果。

2) 、处理异步结果

如果我们想要等待异步执行完成，做一些事情，我们可以通过 promise 的 then 方法来实现。如果想要处理 promise 异步执行失败的事件，还可以跟上 catch:

```
promise.then(function (value) {
    // 异步执行成功后的回调
}).catch(function (error) {
    // 异步执行失败后的回调
})
```

3) 、Promise 改造以前嵌套方式

```
new Promise((resolve, reject) => {
    $.ajax({
        url: "mock/user.json",
        success(data) {
            console.log("查询用户: ", data);
            resolve(data.id);
        },
        error(error) {
            console.log("出现异常了: " + error);
        }
    });
}).then((userId) => {
    return new Promise((resolve, reject) => {
        $.ajax({
            url: `mock/user_course_${userId}.json`,
            ...
```

```
        success(data) {
            console.log("查询到课程: ", data);
            resolve(data.id);
        },
        error(error) {
            console.log("出现异常了: " + error);
        }
    });
});
}).then((corseId) => {
    console.log(corseId);

    $.ajax({
        url: `mock/corse_score_${corseId}.json`,
        success(data) {
            console.log("查询到分数: ", data);
        },
        error(error) {
            console.log("出现异常了: " + error);
        }
    });
});
```

4) 、优化处理

优化：通常在企业开发中，会把 promise 封装成通用方法，如下：封装了一个通用的 get 请求方法；

```
let get = function (url, data) { // 实际开发中会单独放到 common.js 中
    return new Promise((resolve, reject) => {
        $.ajax({
            url: url,
            type: "GET",
            data: data,
            success(result) {
                resolve(result);
            },
            error(error) {
                reject(error);
            }
        });
    })
}
```

```
// 使用封装的 get 方法，实现查询分数
get("mock/user.json").then((result) => {
    console.log("查询用户：", result);
    return get(`mock/user_course_${result.id}.json`);
}).then((result) => {
    console.log("查询到课程：", result);
    return get(`mock/course_score_${result.id}.json`)
}).then((result) => {
    console.log("查询到分数：", result);
}).catch(() => {
    console.log("出现异常了：" + error);
});
```

通过比较，我们知道了 Promise 的扁平化设计理念，也领略了这种‘上层设计’带来的好处。我们的项目中会使用到这种异步处理的方式；

9、模块化

1) 、什么是模块化

模块化就是把代码进行拆分，方便重复利用。类似 java 中的导包：要使用一个包，必须先导包。而 JS 中没有包的概念，换来的是 模块。

模块功能主要由两个命令构成：`export`和`import`。

- `export`命令用于规定模块的对外接口。
- `import`命令用于导入其他模块提供的功能。

2) 、export

比如我定义一个 js 文件:hello.js，里面有一个对象

```
const util = {
    sum(a,b){
        return a + b;
    }
}
```

我可以使用 export 将这个对象导出：

```
const util = {
    sum(a,b){
```

```
        return a + b;
    }
}

export {util};
```

当然，也可以简写为：

```
export const util = {
    sum(a,b){
        return a + b;
    }
}
```

`export`不仅可以导出对象，一切 JS 变量都可以导出。比如：基本类型变量、函数、数组、对象。

当要导出多个值时，还可以简写。比如我有一个文件：user.js：

```
var name = "jack"
var age = 21
export {name,age}
```

省略名称

上面的导出代码中，都明确指定了导出的变量名，这样其它人在导入使用时就必须准确写出变量名，否则就会出错。

因此 js 提供了`default`关键字，可以对导出的变量名进行省略

例如：

```
// 无需声明对象的名字
export default {
    sum(a,b){
        return a + b;
    }
}
```

这样，当使用者导入时，可以任意起名字

3) 、import

使用`export`命令定义了模块的对外接口以后，其他 JS 文件就可以通过`import`命令加载这个模块。

例如我要使用上面导出的 util：

```
// 导入 util
```

```
import util from 'hello.js'  
// 调用 util 中的属性  
util.sum(1,2)
```

要批量导入前面导出的 name 和 age:

```
import {name, age} from 'user.js'  
console.log(name + "，今年" + age + "岁了")
```

但是上面的代码暂时无法测试，因为浏览器目前还不支持 ES6 的导入和导出功能。除非借助于工具，把 ES6 的语法进行编译降级到 ES5，比如`Babel-cli`工具
我们暂时不做测试，大家了解即可。

三、Node.js

前端开发，少不了 node.js; Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

<http://nodejs.cn/api/>

我们关注与 node.js 的 npm 功能就行；

NPM 是随同 NodeJS 一起安装的包管理工具，JavaScript-NPM，Java-Maven；

1)、官网下载安装 node.js，并使用 node -v 检查版本

2)、配置 npm 使用淘宝镜像

npm config set registry <http://registry.npm.taobao.org/>

3)、大家如果 npm install 安装依赖出现 chromedriver 之类问题，先在项目里运行下面命令
npm install chromedriver --chromedriver_cdnurl=http://cdn.npm.taobao.org/dist/chromedriver
然后再运行 npm install

四、Vue

1、MVVM 思想

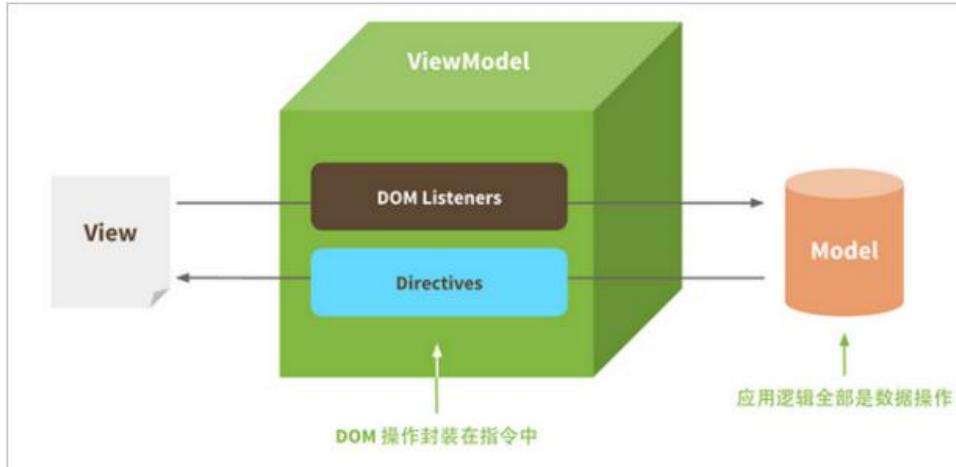
- M: 即 Model，模型，包括数据和一些基本操作
- V: 即 View，视图，页面渲染结果
- VM: 即 View-Model，模型与视图间的双向操作（无需开发人员干涉）

在 MVVM 之前，开发人员从后端获取需要的数据模型，然后要通过 DOM 操作 Model 渲染到 View 中。而后当用户操作视图，我们还需要通过 DOM 获取 View 中的数据，然后同步到 Model 中。

而 MVVM 中的 VM 要做的事情就是把 DOM 操作完全封装起来，开发人员不用再关心 Model 和 View 之间是如何互相影响的：

- 只要我们 Model 发生了改变，View 上自然就会表现出来。
- 当用户修改了 View，Model 中的数据也会跟着改变。

把开发人员从繁琐的 DOM 操作中解放出来，把关注点放在如何操作 Model 上。



2、Vue 简介

Vue (读音 /vju:/, 类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

官网：<https://cn.vuejs.org/>

参考：<https://cn.vuejs.org/v2/guide/>

Git 地址：<https://github.com/vuejs>

尤雨溪，Vue.js 创作者，Vue Technology 创始人，致力于 Vue 的研究开发。

3、入门案例

1) 、安装

官网文档提供了 3 中安装方式：

1. 直接 script 引入本地 vue 文件。需要通过官网下载 vue 文件。
2. 通过 script 引入 CDN 代理。需要联网，生产环境可以使用这种方式

3. 通过 npm 安装。这种方式也是官网推荐的方式，需要 nodejs 环境。

本课程就采用第三种方式

2) 、创建示例项目

1、新建文件夹 hello-vue，并使用 vscode 打开

2、使用 vscode 控制台，npm install -y;

项目会生成 package-lock.json 文件，类似于 maven 项目的 pom.xml 文件。

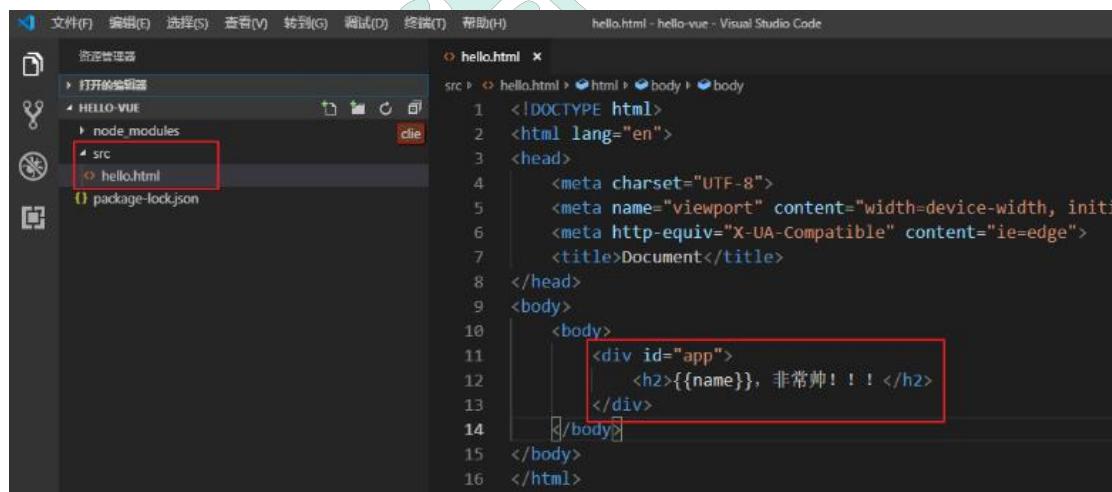
3、使用 npm install vue，给项目安装 vue；项目下会多 node_modules 目录，并且在下面有一个 vue 目录。



3) 、HelloWorld

在 hello.html 中，我们编写一段简单的代码。

h2 中要输出一句话：`xx 非常帅`。前面的`xx`是要渲染的数据。



4) 、vue 声明式渲染

页面代码

```
<body>
<div id="app">
```

```
<h1>{{name}}, 非常帅！！！</h1>
</div>

<script src="./node_modules/vue/dist/vue.min.js"></script>
<script>
    let vm = new Vue({
        el:"#app",
        data:{
            name: "张三"
        }
    });
</script>
</body>
```

- 首先通过 `new Vue()` 来创建 `Vue` 实例
- 然后构造函数接收一个对象，对象中有一些属性：
 - `el:` 是 `element` 的缩写，通过 `id` 选中要渲染的页面元素，本例中是一个 `div`
 - `data:` 数据，数据是一个对象，里面有很多属性，都可以渲染到视图中
 - ◆ `name:` 这里我们指定了一个 `name` 属性
- 页面中的``h1``元素中，我们通过`{{name}}`的方式，来渲染刚刚定义的 `name` 属性。

打开页面查看效果：



张三, 非常帅！！！

更神奇的在于，当你修改 `name` 属性时，页面会跟着变化：

A screenshot of a browser window. The title bar says '李四, 非常帅！！！'. Below the title bar, there is a red arrow pointing from the text to a red box around the console tab in the developer tools. The console tab is labeled 'Console'. In the console, the command 'vm.name="李四"' has been entered, and the output '李四' is shown below it. A red box highlights the command 'vm.name="李四"'.

5)、双向绑定

我们对刚才的案例进行简单修改：

```
<body>
  <div id="app">
    <input type="text" v-model="num">
    <h2>
      {{name}}, 非常帅！！！有{{num}}个人为他点赞。
    </h2>
  </div>
  <script src="./node_modules/vue/dist/vue.js"></script>
  <script>
    // 创建 vue 实例
    let app = new Vue({
      el: "#app", // el 即 element, 该 vue 实例要渲染的页面元素
      data: { // 渲染页面需要的数据
        name: "张三",
        num: 5
      }
    });

  </script>
</body>
```

双向绑定：

效果：我们修改表单项，`num` 会发生变化。我们修改 `num`，表单项也会发生变化。为了实时观察到这个变化，我们将 `num` 输出到页面。

我们不需要关注他们为什么会建立起来关联，以及页面如何变化，我们只需要做好数据和视图的关联即可（MVVM）



张三 ,非常帅，有1个人为他点赞

6)、事件处理

给页面添加一个按钮：

```
<body>
  <div id="app">
    <input type="text" v-model="num">
```

```
<button v-on:click="num++">关注</button>
<h2>
    {{name}}, 非常帅！！！有{{num}}个人为他点赞。
</h2>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script>
    // 创建 vue 实例
    let app = new Vue({
        el: "#app", // el 即 element, 该 vue 实例要渲染的页面元素
        data: { // 渲染页面需要的数据
            name: "张三",
            num: 5
        }
    });

</script>
</body>
```

10 点赞

张三 ,非常帅，有10个人为他点赞

- 这里用`v-on`指令绑定点击事件，而不是普通的`onclick`，然后直接操作 num
- 普通 click 是无法直接操作 num 的。
- 未来我们会见到更多 v-xxx，这些都是 vue 定义的不同功能的指令。

简单使用总结：

- 1)、使用 Vue 实例管理 DOM
- 2)、DOM 与数据/事件等进行相关绑定
- 3)、我们只需要关注数据，事件等处理，无需关心视图如何进行修改

4、概念

1、创建 Vue 实例

每个 Vue 应用都是通过用 **Vue** 函数创建一个新的 **Vue 实例**开始的：

```
let app = new Vue({
    })
```

在构造函数中传入一个对象，并且在对象中声明各种 Vue 需要的数据和方法，包括：

- el
- data
- methods

等等

接下来我们一一介绍。

2、模板或元素

每个 Vue 实例都需要关联一段 Html 模板，Vue 会基于此模板进行视图渲染。

我们可以通过 el 属性来指定。

例如一段 html 模板：

```
<div id="app">  
    ...  
</div>
```

然后创建 Vue 实例，关联这个 div

```
let vm = new Vue({  
    el: "#app"  
})
```

这样，Vue 就可以基于 id 为`app`的 div 元素作为模板进行渲染了。在这个 div 范围以外的部分是无法使用 vue 特性的。

3、数据

当 Vue 实例被创建时，它会尝试获取在 **data** 中定义的所有属性，用于视图的渲染，并且监视 **data** 中的属性变化，当 **data** 发生改变，所有相关的视图都将重新渲染，这就是“响应式”系统。

html:

```
<div id="app">  
    <input type="text" v-model="name" />  
</div>
```

JS:

```
let vm = new Vue({  
    el: "#app",  
    data: {  
        name: "刘德华"  
    }  
})
```

```
}
```

- name 的变化会影响到`input`的值
- input 中输入的值，也会导致 vm 中的 name 发生改变

4、方法

Vue 实例中除了可以定义 data 属性，也可以定义方法，并且在 Vue 实例的作用范围内使用。

Html:

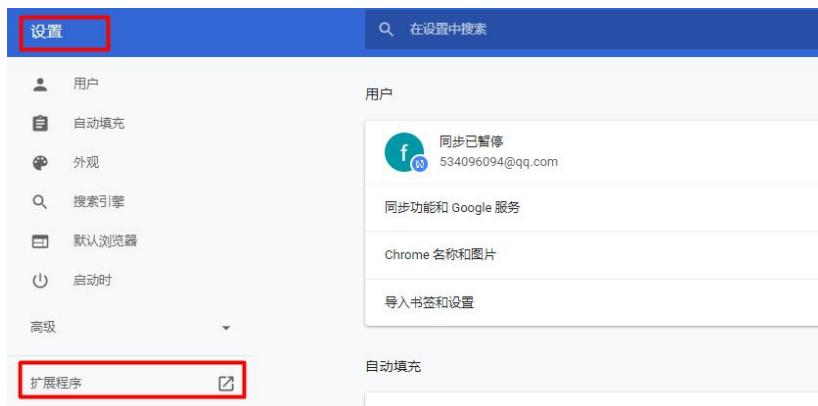
```
<div id="app">
    {{num}}
    <button v-on:click="add">加</button>
</div>
```

JS:

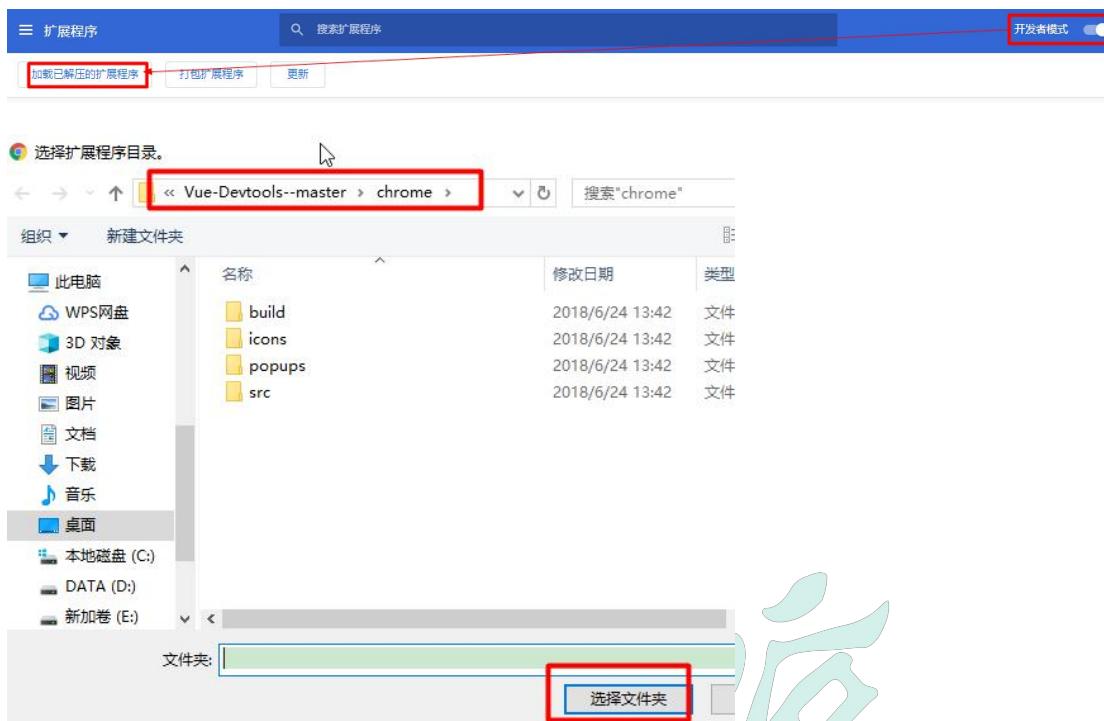
```
let vm = new Vue({
    el: "#app",
    data: {
        num: 0
    },
    methods: {
        add: function () {
            // this 代表的当前 vue 实例
            this.num++;
        }
    }
})
```

5、安装 vue-devtools 方便调试

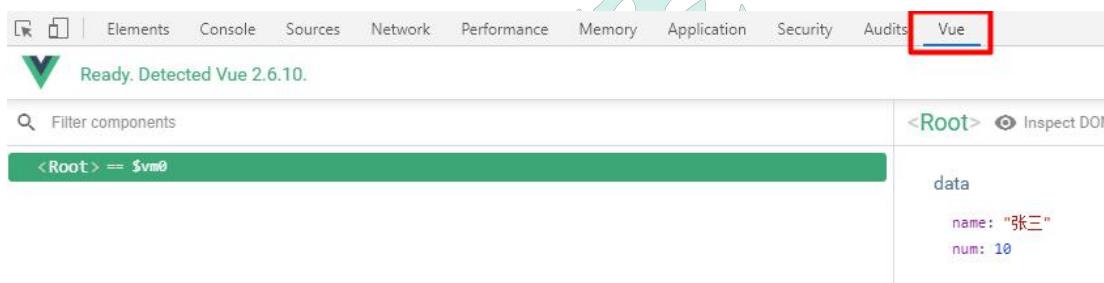
- 将软件包中的 vue-devtools 解压。
- 打开 chrome 设置->扩展程序



- 开启开发者模式，并加载插件



● 打开浏览器控制台，选择 vue



6、安装 vscode 的 vue 插件



安装这个插件就可以有语法提示

5、指令

什么是指令？

- 指令 (Directives) 是带有 `v-` 前缀的特殊特性。
- 指令特性的预期值是：单个 JavaScript 表达式。
- 指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

例如我们在入门案例中的 `v-on`, 代表绑定事件。

1、插值表达式

1) 、花括号

格式: `{表达式}`

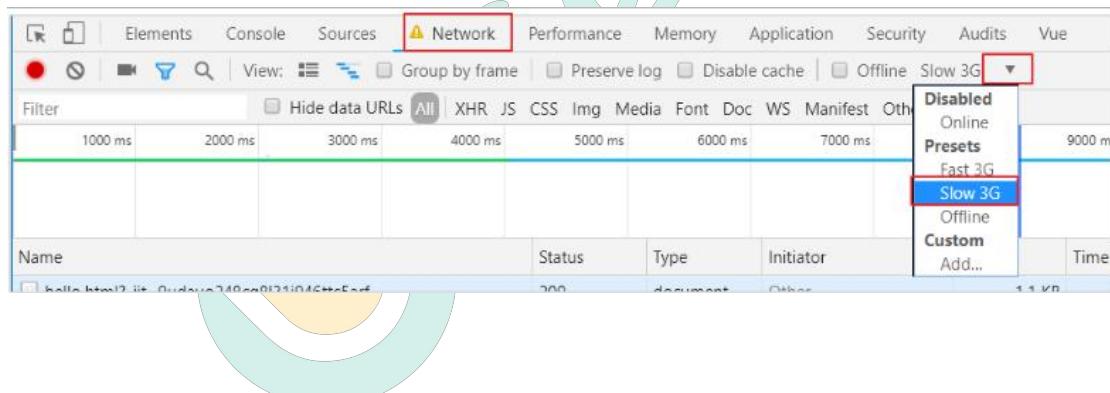
说明:

- 该表达式支持 JS 语法, 可以调用 js 内置函数 (必须有返回值)
- 表达式必须有返回结果。例如 `1 + 1`, 没有结果的表达式不允许使用, 如: `let a = 1 + 1;`
- 可以直接获取 Vue 实例中定义的数据或函数

2) 、插值闪烁

使用`{}`方式在网速较慢时会出现问题。在数据未加载完成时, 页面会显示出原始的``{}``, 加载完毕后才显示正确数据, 我们称为插值闪烁。

我们将网速调慢一些, 然后刷新页面, 试试看刚才的案例:



3) 、`v-text` 和 `v-html`

可以使用 `v-text` 和 `v-html` 指令来替代`{}`

说明:

- `v-text`: 将数据输出到元素内部, 如果输出的数据有 HTML 代码, 会作为普通文本输出
- `v-html`: 将数据输出到元素内部, 如果输出的数据有 HTML 代码, 会被渲染

示例:

```
<div id="app">
    v-text:<span v-text="hello"></span> <br />
    v-html:<span v-html="hello"></span>
</div>
```

```
<script src="./node_modules/vue/dist/vue.js"></script>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            hello: "<h1>大家好</h1>"
        }
    })
</script>
```

效果：

v-text:<h1>大家好</h1>
v-html:

大家好

并且不会出现插值闪烁，当没有数据时，会显示空白或者默认数据。

2、v-bind

html 属性不能使用双大括号形式绑定，我们使用 v-bind 指令给 HTML 标签属性绑定值；而且在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。

1)、绑定 class

```
<div class="static" v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            isActive: true,
            hasError: false
        }
    })
</script>
```

2)、绑定 style

`v-bind:style` 的对象语法十分直观，看着非常像 CSS，但其实是一个 JavaScript 对象。style 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，这种方式记得用单引号括起来) 来命名。

例如：font-size-->fontSize

```
<div id="app" v-bind:style="{ color: activeColor, fontSize: fontSz  
e + 'px' }"></div>  
<script>  
    let vm = new Vue({  
        el: "#app",  
        data: {  
            activeColor: 'red',  
            fontSize: 30  
        }  
    })  
</script>
```

结果：<div style="color: red; font-size: 30px;"></div>

3)、绑定其他任意属性

```
<div id="app" v-bind:style="{ color: activeColor, fontSize: fontSz  
e + 'px' }"  
      v-bind:user="userName">  
</div>  
<script>  
    let vm = new Vue({  
        el: "#app",  
        data: {  
            activeColor: 'red',  
            fontSize: 30,  
            userName: 'zhangsan'  
        }  
    })  
</script>
```

效果：

<div id="app" user="zhangsan" style="color: red; font-size: 30px;"></div>

4)、v-bind 缩写

```
<div id="app" :style="{ color: activeColor, fontSize: fontSize + 'px' }"
      :user="userName">
</div>
```

3、v-model

刚才的 v-text、v-html、v-bind 可以看做是单向绑定，数据影响了视图渲染，但是反过来就不行。接下来学习的 v-model 是双向绑定，视图（View）和模型（Model）之间会互相影响。

既然是双向绑定，一定是在视图中可以修改数据，这样就限定了视图的元素类型。目前 v-model 的可使用元素有：

- input
- select
- textarea
- checkbox
- radio
- components（Vue 中的自定义组件）

基本上除了最后一项，其它都是表单的输入项。

示例：

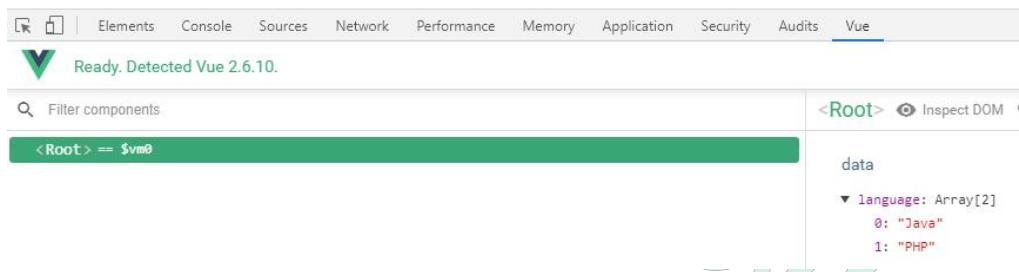
```
<div id="app">
  <input type="checkbox" v-model="language" value="Java" />Java<br />
  <input type="checkbox" v-model="language" value="PHP" />PHP<br />
  <input type="checkbox" v-model="language" value="Swift" />Swift<br />
  <h1>
    你选择了: {{language.join(',')}}
  </h1>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  let vm = new Vue({
    el: "#app",
    data: {
      language: []
    }
  })
</script>
```

- 多个`CheckBox`对应一个 model 时，model 的类型是一个数组，单个 checkbox 值默认是 boolean 类型
- radio 对应的值是 input 的 value 值
- `text` 和 `textarea` 默认对应的 model 是字符串
- `select` 单选对应字符串，多选对应也是数组

效果：

Java
 PHP
 Swift

你选择了：Java, PHP



```
<Root> == $vm0
data
language: Array[2]
0: "Java"
1: "PHP"
```

4、v-on

1、基本用法

v-on 指令用于给页面元素绑定事件。

语法： `v-on:事件名="js 片段或函数名"`

示例：

```
<div id="app">
    <!--事件中直接写 js 片段-->
    <button v-on:click="num++">点赞</button>
    <!--事件指定一个回调函数，必须是 Vue 实例中定义的函数-->
    <button v-on:click="decrement">取消</button>
    <h1>有{{num}}个赞</h1>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    let vm = new Vue({
        el: "#app",
        data: {
            num: 100
        },
        methods: {
```

```
        decrement() {
            this.num--; //要使用 data 中的属性，必须 this.属性名
        }
    })
</script>
```

另外，事件绑定可以简写，例如`v-on:click='add'`可以简写为`@click='add'`

2、事件修饰符

在事件处理程序中调用 `event.preventDefault()` 或 `event.stopPropagation()` 是非常常见的需求。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法只有纯粹的数据逻辑，而不是去处理 DOM 事件细节。

为了解决这个问题，Vue.js 为 `v-on` 提供了事件修饰符。修饰符是由点开头的指令后缀来表示的。

- `.`stop`：阻止事件冒泡到父元素
- `.`prevent`：阻止默认事件发生
- `.`capture`：使用事件捕获模式
- `.`self`：只有元素自身触发事件才执行。（冒泡或捕获的都不执行）
- `.`once`：只执行一次

```
<div id="app">
    <!--右击事件，并阻止默认事件发生-->
    <button v-on:contextmenu.prevent="num++">点赞</button>
    <br />
    <!--右击事件，不阻止默认事件发生-->
    <button v-on:contextmenu="decrement($event)">取消</button>
    <br />
    <h1>有{{num}}个赞</h1>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    let app = new Vue({
        el: "#app",
        data: {
            num: 100
        },
        methods: {
            decrement(ev) {
                // ev.preventDefault();
                this.num--;
            }
        }
    });
</script>
```

```
        }
    })
</script>
```

效果：右键“点赞”，不会触发默认的浏览器右击事件；右键“取消”，会触发默认的浏览器右击事件）

3、按键修饰符

在监听键盘事件时，我们经常需要检查常见的键值。Vue 允许为 `v-on` 在监听键盘事件时添加按键修饰符：

```
<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->
<input v-on:keyup.13="submit">
```

记住所有的 `keyCode` 比较困难，所以 Vue 为最常用的按键提供了别名：

```
<!-- 同上 -->
<input v-on:keyup.enter="submit">
<!-- 缩写语法 -->
<input @keyup.enter="submit">
```

全部的按键别名：

- `enter`
- `tab`
- `delete` (捕获“删除”和“退格”键)
- `esc`
- `space`
- `up`
- `down`
- `left`
- `right`

4、组合按钮

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

- `ctrl`
- `alt`
- `shift`

```
<!-- Alt + C -->
<input @keyup.alt.67="clear">
<!-- Ctrl + Click -->
```

```
<div @click.ctrl="doSomething">Do something</div>
```

5、v-for

遍历数据渲染页面是非常常用的需求，Vue 中通过 v-for 指令来实现。

1、遍历数组

语法: **v-for="item in items"**

- items: 要遍历的数组，需要在 vue 的 data 中定义好。
- item: 迭代得到的当前正在遍历的元素

示例：

```
<div id="app">
  <ul>
    <li v-for="user in users">
      {{user.name}} - {{user.gender}} - {{user.age}}
    </li>
  </ul>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  let app = new Vue({
    el: "#app",
    data: {
      users: [
        { name: '柳岩', gender: '女', age: 21 },
        { name: '张三', gender: '男', age: 18 },
        { name: '范冰冰', gender: '女', age: 24 },
        { name: '刘亦菲', gender: '女', age: 18 },
        { name: '古力娜扎', gender: '女', age: 25 }
      ]
    }
  })
</script>
```

效果：

- 柳岩 - 女 - 21
- 张三 - 男 - 18
- 范冰冰 - 女 - 24
- 刘亦菲 - 女 - 18
- 古力娜扎 - 女 - 25

2、数组角标

在遍历的过程中，如果我们需要知道数组角标，可以指定第二个参数：

语法: `v-for="(item,index) in items"`

- `items`: 要迭代的数组
- `item`: 迭代得到的数组元素别名
- `index`: 迭代到的当前元素索引，从 0 开始。

示例：

```
<div id="app">
  <ul>
    <li v-for="(user, index) in users">
      {{index + 1}}. {{user.name}} - {{user.gender}} - {{user.age}}
    </li>
  </ul>
</div>
```

效果：

- 1. 柳岩 - 女 - 21
- 2. 张三 - 男 - 18
- 3. 范冰冰 - 女 - 24
- 4. 刘亦菲 - 女 - 18
- 5. 古力娜扎 - 女 - 25

3、遍历对象

`v-for` 除了可以迭代数组，也可以迭代对象。语法基本类似

语法：

`v-for="value in object"`

`v-for="(value,key) in object"`

`v-for="(value,key,index) in object"`

- 1 个参数时，得到的是对象的属性值
- 2 个参数时，第一个是属性值，第二个是属性名
- 3 个参数时，第三个是索引，从 0 开始

示例：

```
<div id="app">
  <ul>
    <li v-for="(value, key, index) in user">
      {{index + 1}}. {{key}} - {{value}}
    </li>
  </ul>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  let vm = new Vue({
    el: "#app",
    data: {
      user: { name: '张三', gender: '男', age: 18 }
    }
  })
</script>
```

效果：

- 1. name - 张三
- 2. gender - 男
- 3. age - 18

4、Key

用来标识每一个元素的唯一特征，这样 Vue 可以使用“就地复用”策略有效的提高渲染的效率。

示例：

```
<ul>
  <li v-for="(item,index) in items" :key="index"></li>
</ul>

<ul>
  <li v-for="item in items" :key="item.id"></li>
</ul>
```

最佳实践：

如果 items 是数组，可以使用 index 作为每个元素的唯一标识

如果 items 是对象数组，可以使用 item.id 作为每个元素的唯一标识

6、v-if 和 v-show

1、基本用法

v-if，顾名思义，条件判断。当得到结果为 true 时，所在的元素才会被渲染。

v-show，当得到结果为 true 时，所在的元素才会被显示。

语法： **v-if="布尔表达式", v-show="布尔表达式",**

示例：

```
<div id="app">
    <button v-on:click="show = !show">点我呀</button>
    <br>
    <h1 v-if="show">
        看到我啦？！
    </h1>
    <h1 v-show="show">
        看到我啦？！ show
    </h1>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    let app = new Vue({
        el: "#app",
        data: {
            show: true
        }
    })
</script>
```

2、与 v-for 结合

当 v-if 和 v-for 出现在一起时，v-for 优先级更高。也就是说，会先遍历，再判断条件。

修改 v-for 中的案例，添加 v-if:

```
<ul>
    <li v-for="(user, index) in users" v-if="user.gender == '女'">
```

```
    {{index + 1}}. {{user.name}} - {{user.gender}} - {{user.age}}  
  </li>  
</ul>
```

效果：只显示女性

- 1. 柳岩 - 女 - 21
- 3. 范冰冰 - 女 - 24
- 4. 刘亦菲 - 女 - 18
- 5. 古力娜扎 - 女 - 25

7、v-else 和 v-else-if

v-else 元素必须紧跟在带 `v-if` 或者 `v-else-if` 的元素的后面，否则它将不会被识别。

示例：

```
<div id="app">  
  <button v-on:click="random=Math.random()">点我呀  
</button><span>{{random}}</span>  
  <h1 v-if="random >= 0.75">  
    看到我啦？！v-if >= 0.75  
  </h1>  
  <h1 v-else-if="random > 0.5">  
    看到我啦？！v-else-if > 0.5  
  </h1>  
  <h1 v-else-if="random > 0.25">  
    看到我啦？！v-else-if > 0.25  
  </h1>  
  <h1 v-else>  
    看到我啦？！v-else  
  </h1>  
</div>  
<script src="../node_modules/vue/dist/vue.js"></script>  
<script type="text/javascript">  
  let app = new Vue({  
    el: "#app",  
    data: {  
      random: 1  
    }  
  })  
</script>
```

6、计算属性和侦听器

1、计算属性（computed）

某些结果是基于之前数据实时计算出来的，我们可以利用计算属性。来完成示例：

```
<div id="app">
  <ul>
    <li>西游记： 价格{{xyjPrice}}， 数量：
<input type="number" v-model="xyjNum"></li>
    <li>水浒传： 价格{{shzPrice}}， 数量：
<input type="number" v-model="shzNum"></li>
    <li>总价： {{totalPrice}}</li>
  </ul>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
  let app = new Vue({
    el: "#app",
    data: {
      xyjPrice: 56.73,
      shzPrice: 47.98,
      xyjNum: 1,
      shzNum: 1
    },
    computed: {
      totalPrice(){
        return this.xyjPrice*this.xyjNum + this.shzPrice*this.shzNum;
      }
    }
  })
</script>
```

效果：只要依赖的属性发生变化，就会重新计算这个属性

- 西游记： 价格56.73， 数量：
- 水浒传： 价格47.98， 数量：
- 总价： 370.86 动态计算

2、侦听（watch）

watch 可以让我们监控一个值的变化。从而做出相应的反应。

示例：

```
<div id="app">
    <ul>
        <li>西游记： 价格{{xyjPrice}}, 数量:
<input type="number" v-model="xyjNum"></li>
        <li>水浒传： 价格{{shzPrice}}, 数量:
<input type="number" v-model="shzNum"></li>
        <li>总价： {{totalPrice}}</li>
        {{msg}}
    </ul>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    let app = new Vue({
        el: "#app",
        data: {
            xyjPrice: 56.73,
            shzPrice: 47.98,
            xyjNum: 1,
            shzNum: 1,
            msg: ""
        },
        computed: {
            totalPrice(){
                return this.xyjPrice*this.xyjNum + this.shzPrice*this.shzNum;
            }
        },
        watch: {
            xyjNum(newVal, oldVal){
                if(newVal >= 3){
                    this.msg = "西游记没有更多库存了";
                    this.xyjNum = 3;
                }else{
                    this.msg = "";
                }
            }
        }
    })
</script>
```

效果：

- 西游记：价格56.73，数量： 1
 - 水浒传：价格47.98，数量：
 - 总价：218.17
- 西游记没有更多库存了 数量不能超过3

3、过滤器 (filters)

过滤器不改变真正的`data`，而只是改变渲染的结果，并返回过滤后的版本。在很多不同的情况下，过滤器都是有用的，比如尽可能保持 API 响应的干净，并在前端处理数据的格式。

示例：展示用户列表性别显示男女

```
<body>
  <div id="app">
    <table>
      <tr v-for="user in userList">
        <td>{{user.id}}</td>
        <td>{{user.name}}</td>
        <!-- 使用代码块实现，有代码侵入 -->
        <td>{{user.gender==1? "男": "女"}}</td>
      </tr>
    </table>
  </div>
</body>
<script src="../node_modules/vue/dist/vue.js"></script>
<script>

  let app = new Vue({
    el: "#app",
    data: {
      userList: [
        { id: 1, name: 'jacky', gender: 1 },
        { id: 2, name: 'peter', gender: 0 }
      ]
    }
  });

</script>
```

1、局部过滤器

注册在当前 vue 实例中，只有当前实例能用

```
let app = new Vue({
  el: "#app",
  data: {
    userList: [
      { id: 1, name: 'jacky', gender: 1 },
      { id: 2, name: 'peter', gender: 0 }
    ],
    // filters 定义局部过滤器，只可以在当前 vue 实例中使用
    filters: {
      genderFilter(gender) {
        return gender === 1 ? '男~' : '女~'
      }
    }
});
```

<!-- | 管道符号：表示使用后面的过滤器处理前面的数据 -->

```
<td>{{user.gender | genderFilter}}</td>
```

2、全局过滤器

```
// 在创建 Vue 实例之前全局定义过滤器:
Vue.filter('capitalize', function (value) {
  return value.charAt(0).toUpperCase() + value.slice(1)
})
```

任何 vue 实例都可以使用: <td>{{user.name | capitalize}}</td>

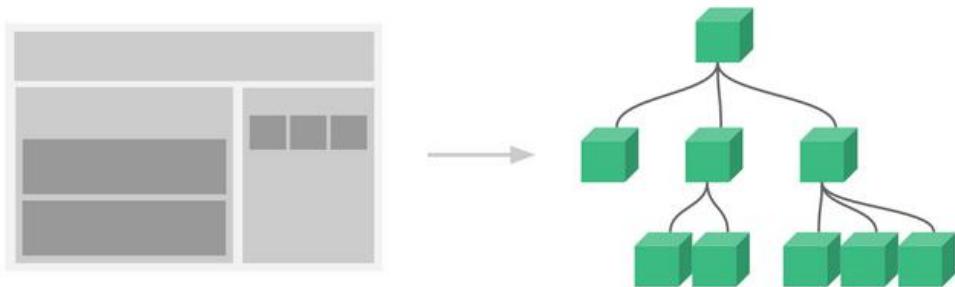
过滤器常用来处理文本格式化的操作。过滤器可以用在两个地方：[双花括号插值](#)和 [v-bind 表达式](#)

7、组件化

在大型应用开发的时候，页面可以划分成很多部分。往往不同的页面，也会有相同的部分。例如可能会有相同的头部导航。

但是如果每个页面都独自开发，这无疑增加了我们开发的成本。所以我们会把页面的不同部分拆分成独立的组件，然后在不同页面就可以共享这些组件，避免重复开发。

在 vue 里，所有的 vue 实例都是组件



例如，你可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的数据。

1、全局组件

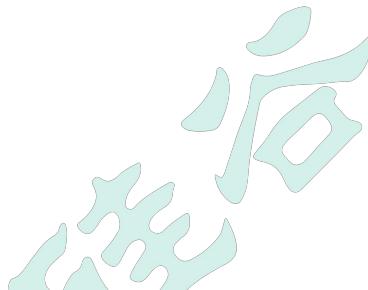
我们通过 Vue 的 `component` 方法来定义一个全局组件。

```
<div id="app">
    <!-- 使用定义好的全局组件 -->
    <counter></counter>
</div>
<script src="../node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    // 定义全局组件，两个参数：1，组件名称。2，组件参数
    Vue.component("counter", {
        template: '<button v-on:click="count++>你点了
我 {{ count }} 次，我记住了.</button>',
        data() {
            return {
                count: 0
            }
        }
    })
</script>
```

```
let app = new Vue({
  el: "#app"
})
</script>
```

- 组件其实也是一个 `Vue` 实例，因此它在定义时也会接收：`data`、`methods`、生命周期函数等
- 不同的是组件不会与页面的元素绑定，否则就无法复用了，因此没有 `el` 属性。
- 但是组件渲染需要 `html` 模板，所以增加了 `template` 属性，值就是 `HTML` 模板
- 全局组件定义完毕，任何 `vue` 实例都可以直接在 `HTML` 中通过组件名称来使用组件了
- `data` 必须是一个函数，不再是一个对象。

你点了我 7 次，我记住了。



2、组件的复用

定义好的组件，可以任意复用多次：

```
<div id="app">
  <!-- 使用定义好的全局组件 -->
  <counter></counter>
  <counter></counter>
  <counter></counter>
</div>
```

组件的 `data` 属性必须是函数！

一个组件的 `data` 选项必须是一个函数，因此每个实例可以维护一份被返回对象的独立的拷贝；

否则：

<https://cn.vuejs.org/v2/guide/components.html#data-%E5%BF%85%E9%A1%BB%E6%98%AF%E4%B8%80%E4%B8%AA%E5%87%BD%E6%95%B0>

3、局部组件

一旦全局注册，就意味着即便以后你不再使用这个组件，它依然会随着 `Vue` 的加载而加载。因此，对于一些并不频繁使用的组件，我们会采用局部注册。

我们先在外部定义一个对象，结构与创建组件时传递的第二个参数一致：

```
const counter = {
    template: '<button v-on:click="count++">你点了
我 {{ count }} 次，我记住了.</button>',
    data() {
        return {
            count: 0
        }
    }
};
```

然后在 Vue 中使用它：

```
let app = new Vue({
    el: "#app",
    components: {
        counter: counter // 将定义的对象注册为组件
    }
})
```

- components 就是当前 vue 对象子组件集合。
 - 其 key 就是子组件名称
 - 其值就是组件对象名
- 效果与刚才的全局注册是类似的，不同的是，这个 counter 组件只能在当前的 Vue 实例中使用

简写：

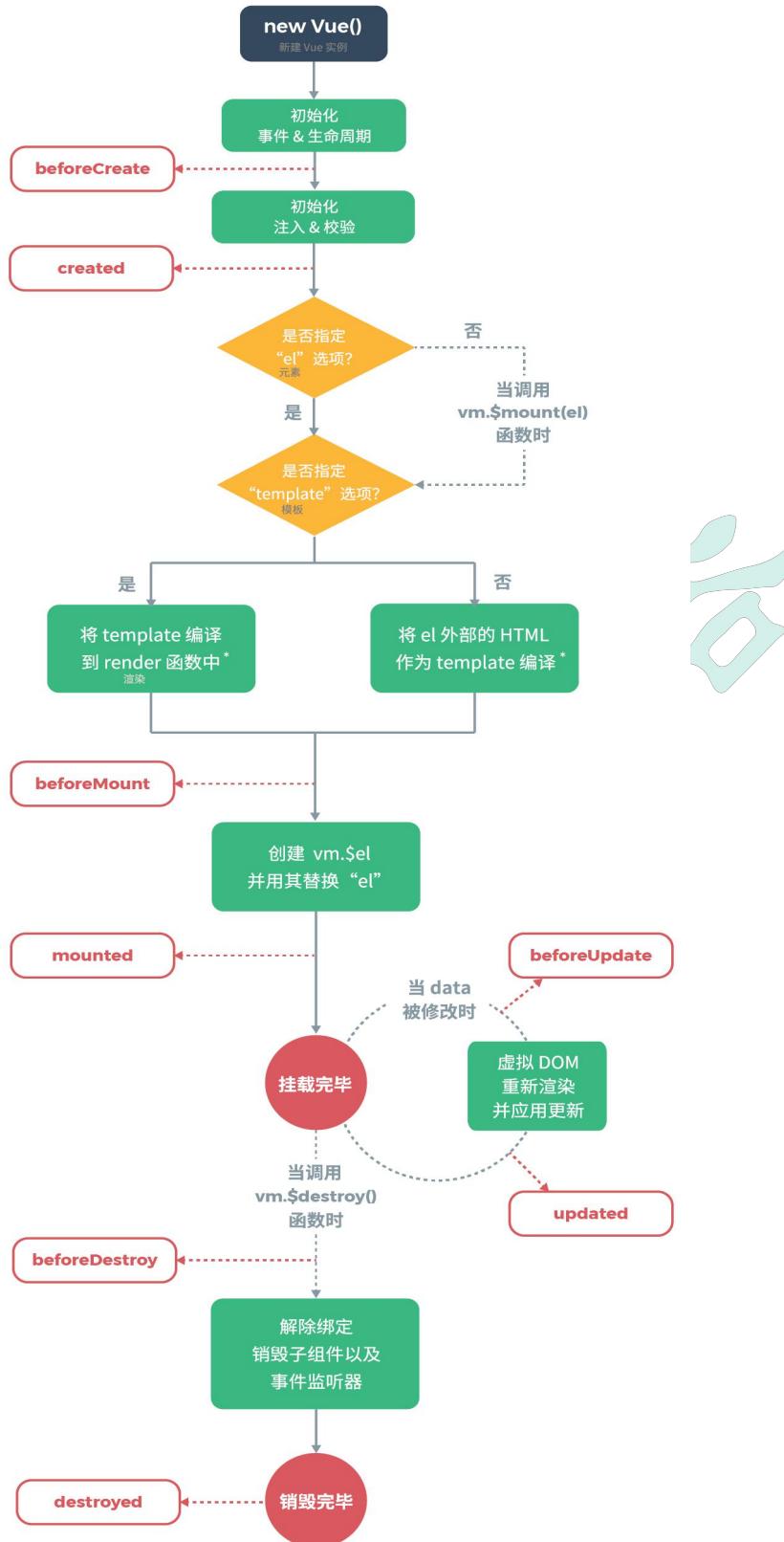
```
let app = new Vue({
    el: "#app",
    components: {
        counter // 将定义的对象注册为组件
    }
})
```

8、生命周期钩子函数

1、生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程：创建实例，装载模板，渲染模板等等。Vue 为生命周期中的每个状态都设置了钩子函数（监听函数）。每当 Vue 实例处于不同的生命周期时，对应的函数就会被触发调用。

生命周期：你不需要立马弄明白所有的东西。



* 如果使用构造生成文件（例如构造单文件组件），
模板编译将提前执行

2、钩子函数

- `beforeCreated`: 我们在用 `Vue` 时都要进行实例化，因此，该函数就是在 `Vue` 实例化时调用，也可以将他理解为初始化函数比较方便一点，在 `Vue1.0` 时，这个函数的名字就是 `init`。
- `created`: 在创建实例之后进行调用。
- `beforeMount`: 页面加载完成，没有渲染。如：此时页面还是`{{name}}`
- `mounted`: 我们可以将他理解为原生 js 中的 `window.onload=function(..){..}`,或许大家也在用 `jquery`，所以也可以理解为 `jquery` 中的`$(document).ready(function(){....})`，他的功能就是：在 `dom` 文档渲染完毕之后将要执行的函数，该函数在 `Vue1.0` 版本中名字为 `compiled`。此时页面中的`{{name}}`已被渲染成张三
- `beforeDestroy`: 该函数将在销毁实例前进行调用。
- `destroyed`: 该函数将在销毁实例时进行调用。
- `beforeUpdate`: 组件更新之前。
- `updated`: 组件更新之后。

示例



```
<body>
  <div id="app">
    <span id="num">{{num}}</span>
    <button v-on:click="num++">赞！</button>
    <h2>
      {{name}}, 非常帅！！！有{{num}}个人点赞。
    </h2>
  </div>
</body>
<script src="../node_modules/vue/dist/vue.js"></script>
<script>
  let app = new Vue({
    el: "#app",
    data: {
      name: "张三",
      num: 100
    },
    methods: {
      show() {
        return this.name;
      },
      add() {
        this.num++;
      }
    },
    beforeCreate() {
```

```
        console.log("=====beforeCreate=====");
        console.log("数据模型未加载: " + this.name, this.num);
        console.log("方法未加载: " + this.show());
        console.log("html 模板未加载:
" + document.getElementById("num"));
    },
    created: function () {
        console.log("=====created=====");
        console.log("数据模型已加载: " + this.name, this.num);
        console.log("方法已加载: " + this.show());
        console.log("html 模板已加载:
" + document.getElementById("num"));
        console.log("html 模板未渲染:
" + document.getElementById("num").innerText);
    },
    beforeMount() {
        console.log("=====beforeMount=====");
        console.log("html 模板未渲染:
" + document.getElementById("num").innerText);
    },
    mounted() {
        console.log("=====mounted=====");
        console.log("html 模板已渲染:
" + document.getElementById("num").innerText);
    },
    beforeUpdate() {
        console.log("=====beforeUpdate=====");
        console.log("数据模型已更新: " + this.num);
        console.log("html 模板未更新:
" + document.getElementById("num").innerText);
    },
    updated() {
        console.log("=====updated=====");
        console.log("数据模型已更新: " + this.num);
        console.log("html 模板已更新:
" + document.getElementById("num").innerText);
    }
});

</script>
```

9、vue 模块化开发

1、npm install webpack -g

全局安装 webpack

2、npm install -g @vue/cli-init

全局安装 vue 脚手架

3、初始化 vue 项目；

vue init webpack appname: vue 脚手架使用 webpack 模板初始化一个 appname 项目

4、启动 vue 项目；

项目的 package.json 中有 scripts，代表我们能运行的命令

npm start = npm run dev: 启动项目

npm run build: 将项目打包

5、模块化开发

1、项目结构

目录/文件	说明
build	项目构建(webpack)相关代码
config	配置目录，包括端口号等。我们初学可以使用默认的。
node_modules	npm 加载的项目依赖模块
src	这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件： <ul style="list-style-type: none">● assets：放置一些图片，如logo等。● components：目录里面放了一个组件文件，可以不用。● App.vue：项目入口文件，我们也可以直接将组件写这里，而不使用 components 目录。● main.js：项目的根本文件。
static	静态资源目录，如图片、字体等。
test	初始测试目录，可删除
.xxxx文件	这些是一些配置文件，包括语法配置，git配置等。
index.html	首页入口文件，你可以添加一些 meta 信息或统计代码啥的。
package.json	项目配置文件。
README.md	项目的说明文档，markdown 格式

● 运行流程

- 进入页面首先加载 index.html 和 main.js 文件。
- main.js 导入了一些模块【vue、app、router】，并且创建 vue 实例，关联 index.html 页面的<div id="app">元素。使用了 router，导入了 App 组件。并且使用<App/>标签引用了这个组件
- 第一次默认显示 App 组件。App 组件有个图片和<router-view>，所以显示了图片。但是由于<router-view>代表路由的视图，默认是访问/#/路径（router 路径默认使用 HASH 模式）。在 router 中配置的/是显示 HelloWorld 组件。
- 所以第一次访问，显示图片和 HelloWorld 组件。
- 我们尝试自己写一个组件，并且加入路由。点击跳转。需要使用<router-link to="/foo">Go to Foo</router-link>标签

2、Vue 单文件组件

Vue 单文件组件模板有三个部分：

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h1, h2 {
  font-weight: normal;
}
</style>
```

Template：编写模板

Script：vue 实例配置

Style：样式

3、vscode 添加用户代码片段（快速生成 vue 模板）

文件-->首选项-->用户代码片段-->点击新建代码片段--取名 vue.json 确定

```
{
  "生成 vue 模板": {
    "prefix": "vue",
    "body": [
      "<template>",
      "<div></div>",
      "</template>",
      "
```

```
        "",
        "<script>",
        "//这里可以导入其他文件（比如：组件，工具 js，第三方插件 js，json
文件，图片文件等等）",
        " //例如：import 《组件名称》 from '《组件路径》';",
        "",
        "export default {",
        " //import 引入的组件需要注入到对象中才能使用",
        "components: {},",
        "props: {},",
        "data() {",
        " //这里存放数据",
        "return {",
        "",
        "",
        "};",
        "}",
        " //计算属性 类似于 data 概念",
        "computed: {},",
        " //监控 data 中的数据变化",
        "watch: {},",
        " //方法集合",
        "methods: {",
        "",
        "",
        "}",
        " //生命周期 - 创建完成（可以访问当前 this 实例）",
        "created() {",
        "",
        "}",
        " //生命周期 - 挂载完成（可以访问 DOM 元素）",
        "mounted() {",
        "",
        "}",
        "beforeCreate() {}, //生命周期 - 创建之前",
        "beforeMount() {}, //生命周期 - 挂载之前",
        "beforeUpdate() {}, //生命周期 - 更新之前",
        "updated() {}, //生命周期 - 更新之后",
        "beforeDestroy() {}, //生命周期 - 销毁之前",
        "destroyed() {}, //生命周期 - 销毁完成",
        "activated() {}, //如果页面有 keep-alive 缓存功能，这个函数会触发
",
        "}",
        "</script>",
        "<style lang='scss' scoped>",
        " //@import url($3); 引入公共 css 类",
```

```
        "$4",
        "</style>"
    ],
    "description": "生成 vue 模板"
}
}
```

4、导入 element-ui 快速开发

1、安装 element-ui: `npm i element-ui`

2、在 main.js 中引入 element-ui 就可以全局使用了。

```
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'
```

```
Vue.use(ElementUI)
```

3、将 App.vue 改为 element-ui 中的后台布局

4、添加测试路由、组件，测试跳转逻辑

(1) 、参照文档 el-menu 添加 router 属性

(2) 、参照文档 el-menu-item 指定 index 需要跳转的地址

五、Babel

Babel 是一个 JavaScript 编译器，我们可以使用 es 的最新语法编程，而不用担心浏览器兼容问题。他会自动转化为浏览器兼容的代码

六、Webpack

自动化项目构建工具。gulp 也是同类产品

谷粒商城

商品服务



一、基础概念

1、三级分类



2、SPU 与 SKU

SPU: Standard Product Unit (标准化产品单元)

是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。





iphoneX 是 SPU、MI 8 是 SPU

iphoneX 64G 黑曜石 是 SKU

MI8 8+64G+黑色 是 SKU

SKU: Stock Keeping Unit (库存量单位)

即库存进出计量的基本单元，可以是以件，盒，托盘等为单位。SKU 这是对于大型连锁超市 DC（配送中心）物流管理的一个必要的方法。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

3、基本属性【规格参数】与销售属性

每个分类下的商品共享规格参数，与销售属性。只是有些商品不一定用这个分类下全部的属性；

- 属性是以三级分类组织起来的
- 规格参数中有些是可以提供检索的
- 规格参数也是基本属性，他们具有自己的分组
- 属性的分组也是以三级分类组织起来的
- 属性名确定的，但是值是每一个商品不同来决定的

二、接口编写

1、HTTP 请求模板

```
"http-get 请求": {
    "prefix": "httpget",
    "body": [
        "this.\$\$http({",
        "url: this.\$\$http.adornUrl(),
        "method: 'get',
        "params: this.\$\$http.adornParams({})",
        "}).then(({data}) => {",
        "})"
    ],
    "description": "httpGET 请求"
}

"http-post 请求": {
    "prefix": "httppost",
    "body": [
        "this.\$\$http({",
        "url: this.\$\$http.adornUrl(),
        "method: 'post',
        "data: this.\$\$http.adornData(data, false)",
        "}).then(({ data }) => { });
    ],
    "description": "httpPOST 请求"
}
```

2、JSR303 数据校验

1) 、使用步骤

- 标注校验注解
javax.validation.constraints 中定义了非常多的校验注解
@Email、@Future、@NotBlank、@Size 等

- 使用校验功能
@Valid 开启校验功能

- 提取校验错误信息

BindingResult 获取校验结果

- 分组校验与自定义校验

Groups 定义校验分组信息；

可以编写自定义校验注解和自定义校验器

默认情况下，异常信息会从应用的 classpath 下的 ValidationMessages.properties 文件中加载；

3、全局异常处理

@ControllerAdvice+@ExceptionHandler

系统错误码

```
/**
 * 错误码和错误信息定义类
 * 1. 错误码定义规则为 5 为数字
 * 2. 前两位表示业务场景，最后三位表示错误码。例如：100001。10:通用 001:系统未知
 * 异常
 * 3. 维护错误码后需要维护错误描述，将他们定义为枚举形式
 * 错误码列表：
 * 10: 通用
 * 001: 参数格式校验
 * 11: 商品
 * 12: 订单
 * 13: 购物车
 * 14: 物流
 *
 */

```

4、接口文档地址

<https://easydoc.xyz/#/s/78237135>

5、Object 划分

1.PO(persistent object) 持久对象

PO 就是对应数据库中某个表中的一条记录，多个记录可以用 PO 的集合。PO 中应该不包含任何对数据库的操作。

2.DO (Domain Object) 领域对象

就是从现实世界中抽象出来的有形或无形的业务实体。

3.TO(Transfer Object) , 数据传输对象

不同的应用程序之间传输的对象

4.DTO (Data Transfer Object) 数据传输对象

这个概念来源于 J2EE 的设计模式，原来的目的是为了 EJB 的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，泛指用于展示层与服务层之间的数据传输对象。

5.VO(value object) 值对象

通常用于业务层之间的数据传递，和 PO 一样也是仅仅包含数据而已。但是是抽象出的业务对象，可以和表对应，也可以不，这根据业务的需要。用 new 关键字创建，由 GC 回收的。

View object: 视图对象；

接受页面传递来的数据，封装对象

将业务处理完成的对象，封装成页面要用的数据

6.BO(business object) 业务对象

从业务模型的角度看，见 UML 元件领域模型中的领域对象。封装业务逻辑的 java 对象，通过调用 DAO 方法，结合 PO,VO 进行业务操作。business object: 业务对象 主要作用是把业务逻辑封装为一个对象。这个对象可以包括一个或多个其它的对象。比如一个简历，有教育经历、工作经历、社会关系等等。我们可以把教育经历对应一个 PO，工作经历对应一个 PO，社会关系对应一个 PO。建立一个对应简历的 BO 对象处理简历，每个 BO 包含这些 PO。这样处理业务逻辑时，我们就可以针对 BO 去处理。



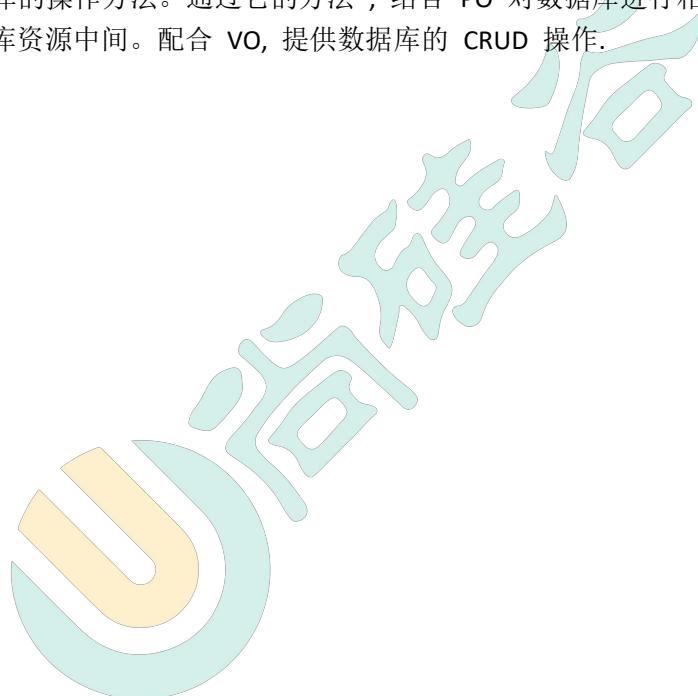
7.POJO(plain ordinary java object) 简单无规则 java 对象

传统意义的 java 对象。就是说在一些 Object/Relation Mapping 工具中，能够做到维护数据库表记录的 persistent object 完全是一个符合 Java Bean 规范的纯 Java 对象，没有增加别的属性和方法。我的理解就是最基本的 Java Bean ，只有属性字段及 setter 和 getter 方法！。

POJO 是 DO/DTO/BO/VO 的统称。

8.DAO(data access object) 数据访问对象

是一个 sun 的一个标准 j2ee 设计模式，这个模式中有个接口就是 DAO ，它负责持久层的操作。为业务层提供接口。此对象用于访问数据库。通常和 PO 结合使用， DAO 中包含了各种数据库的操作方法。通过它的方法，结合 PO 对数据库进行相关操作。夹在业务逻辑与数据库资源中间。配合 VO，提供数据库的 CRUD 操作。





让天下没有难学的技术



谷粒商城

本地事务与分布式事务



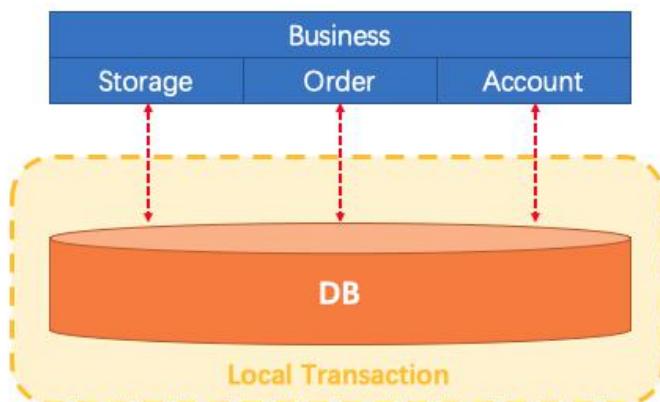
一、本地事务

1、事务的基本性质

数据库事务的几个特性：原子性(Atomicity)、一致性(Consistency)、隔离性或独立性(Isolation)和持久性(Durability)，简称就是ACID；

- 原子性：一系列的操作整体不可拆分，要么同时成功，要么同时失败
- 一致性：数据在事务的前后，业务整体一致。
 - 转账。A:1000; B:1000; 转 200 事务成功; A: 800 B: 1200
- 隔离性：事务之间互相隔离。
- 持久性：一旦事务成功，数据一定会落盘在数据库。

在以往的单体应用中，我们多个业务操作使用同一条连接操作不同的数据表，一旦有异常，我们可以很容易的整体回滚；



Business: 我们具体的业务代码

Storage: 库存业务代码；扣库存

Order: 订单业务代码；保存订单

Account: 账号业务代码；减账户余额

比如买东西业务，扣库存，下订单，账户扣款，是一个整体；必须同时成功或者失败

一个事务开始，代表以下的所有操作都在同一个连接里面；

2、事务的隔离级别

- READ UNCOMMITTED（读未提交）

该隔离级别的事务会读到其它未提交事务的数据，此现象也称之为脏读。

- READ COMMITTED (读提交)

一个事务可以读取另一个已提交的事务，多次读取会造成不一样的结果，此现象称为不可重复读问题，Oracle 和 SQL Server 的默认隔离级别。

- REPEATABLE READ (可重复读)

该隔离级别是 MySQL 默认的隔离级别，在同一个事务里，select 的结果是事务开始时时间点的状态，因此，同样的 select 操作读到的结果会是一致的，但是，会有幻读现象。MySQL 的 InnoDB 引擎可以通过 next-key locks 机制（参考下文“行锁的算法”一节）来避免幻读。

- SERIALIZABLE (序列化)

在该隔离级别下事务都是串行顺序执行的，MySQL 数据库的 InnoDB 引擎会给读操作隐式加一把读共享锁，从而避免了脏读、不可重读复读和幻读问题。

3、事务的传播行为

1、**PROPAGATION_REQUIRED**: 如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

2、**PROPAGATION_SUPPORTS**: 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。

3、**PROPAGATION_MANDATORY**: 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

4、**PROPAGATION_REQUIRE_NEW**: 创建新事务，无论当前存不存在事务，都创建新事务。

5、**PROPAGATION_NOT_SUPPORTED**: 以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

6、**PROPAGATION_NEVER**: 以非事务方式执行，如果当前存在事务，则抛出异常。

7、**PROPAGATION_NESTED**: 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与 PROPAGATION_REQUIRED 类似的操作。

4、SpringBoot 事务关键点

1、事务的自动配置

TransactionAutoConfiguration

2、事务的坑

在同一个类里面，编写两个方法，内部调用的时候，会导致事务设置失效。原因是没有用到

代理对象的缘故。

解决：

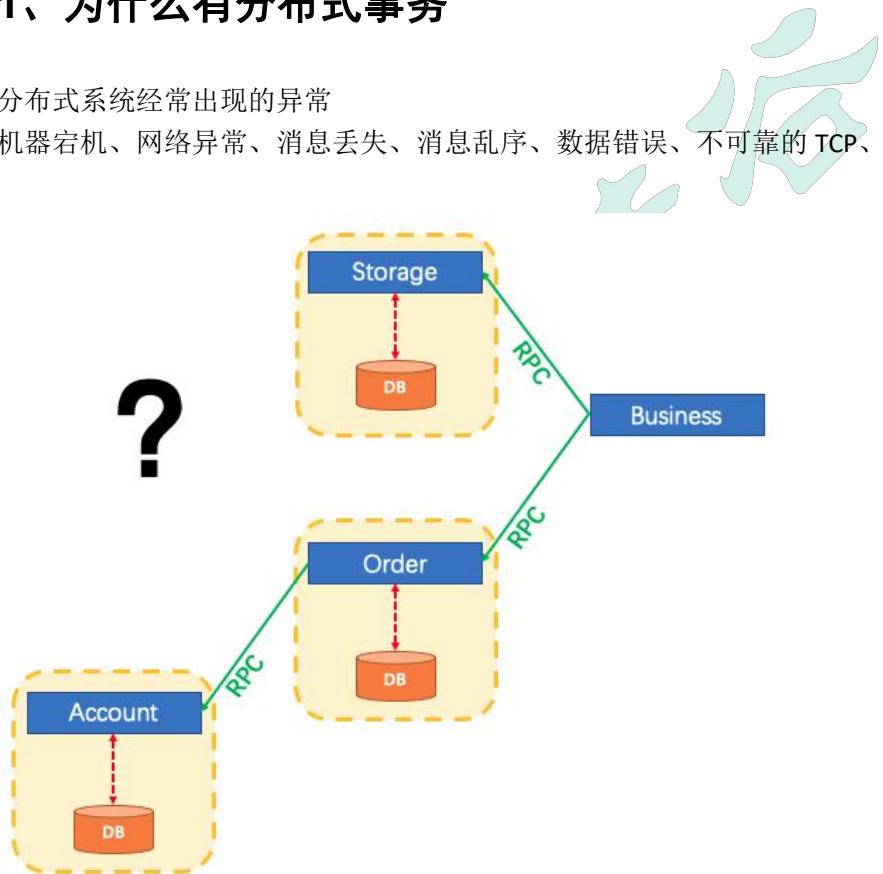
- 0)、导入 spring-boot-starter-aop
- 1)、`@EnableTransactionManagement(proxyTargetClass = true)`
- 2)、`@EnableAspectJAutoProxy(exposeProxy=true)`
- 3)、`AopContext.currentProxy()` 调用方法

二、分布式事务

1、为什么有分布式事务

分布式系统经常出现的异常

机器宕机、网络异常、消息丢失、消息乱序、数据错误、不可靠的 TCP、存储数据丢失...



分布式事务是企业集成中的一个技术难点，也是每一个分布式系统架构中都会涉及到的一个东西，特别是在微服务架构中，几乎可以说是无法避免。

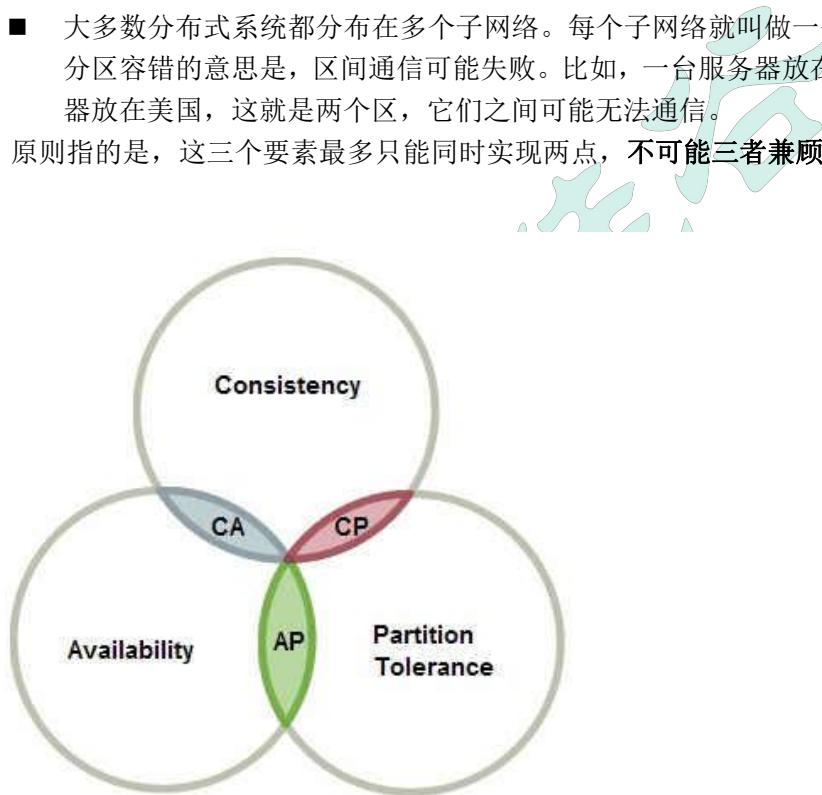
2、CAP 定理与 BASE 理论

1、CAP 定理

CAP 原则又称 CAP 定理，指的是在一个分布式系统中

- 一致性（Consistency）：
 - 在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）
- 可用性（Availability）：
 - 在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）
- 分区容错性（Partition tolerance）：
 - 大多数分布式系统都分布在多个子网络。每个子网络就叫做一个区（partition）。分区容错的意思是，区间通信可能失败。比如，一台服务器放在中国，另一台服务器放在美国，这就是两个区，它们之间可能无法通信。

CAP 原则指的是，这三个要素最多只能同时实现两点，不可能三者兼顾。



一般来说，分区容错无法避免，因此可以认为 CAP 的 P 总是成立。CAP 定理告诉我们，剩下的 C 和 A 无法同时做到。

分布式系统中实现一致性的 raft 算法、paxos

<http://thesecretlivesofdata.com/raft/>

2、面临的问题

对于多数大型互联网应用的场景，主机众多、部署分散，而且现在的集群规模越来越大，所以节点故障、网络故障是常态，而且要保证服务可用性达到 99.99999% (N 个 9)，即保证 P 和 A，舍弃 C。

3、BASE 理论

是对 CAP 理论的延伸，思想是即使无法做到强一致性 (CAP 的一致性就是强一致性)，但可以采用适当的采取弱一致性，即**最终一致性**。

BASE 是指

- 基本可用 (Basically Available)
 - 基本可用是指分布式系统在出现故障的时候，允许损失部分可用性(例如响应时间、功能上的可用性)，允许损失部分可用性。需要注意的是，基本可用绝不等价于系统不可用。
 - ◆ 响应时间上的损失：正常情况下搜索引擎需要在 0.5 秒之内返回给用户相应的查询结果，但由于出现故障（比如系统部分机房发生断电或断网故障），查询结果的响应时间增加到了 1~2 秒。
 - ◆ 功能上的损失：购物网站在购物高峰（如双十一）时，为了保护系统的稳定性，部分消费者可能会被引导到一个降级页面。
- 软状态 (Soft State)
 - 软状态是指允许系统存在中间状态，而该中间状态不会影响系统整体可用性。分布式存储中一般一份数据会有多个副本，允许不同副本同步的延时就是软状态的体现。mysql replication 的异步复制也是一种体现。
- 最终一致性 (Eventual Consistency)
 - 最终一致性是指系统中的所有数据副本经过一定时间后，最终能够达到一致的状态。弱一致性和强一致性相反，最终一致性是弱一致性的一种特殊情况。

4、强一致性、弱一致性、最终一致性

从客户端角度，多进程并发访问时，更新过的数据在不同进程如何获取的不同策略，决定了不同的一致性。对于关系型数据库，要求更新过的数据能被后续的访问都能看到，这是**强一致性**。如果能容忍后续的部分或者全部访问不到，则是**弱一致性**。如果经过一段时间后要求能访问到更新后的数据，则是**最终一致性**

3、分布式事务几种方案

1) 、2PC 模式

数据库支持的 2PC 【2 phase commit 二阶提交】，又叫做 XA Transactions。

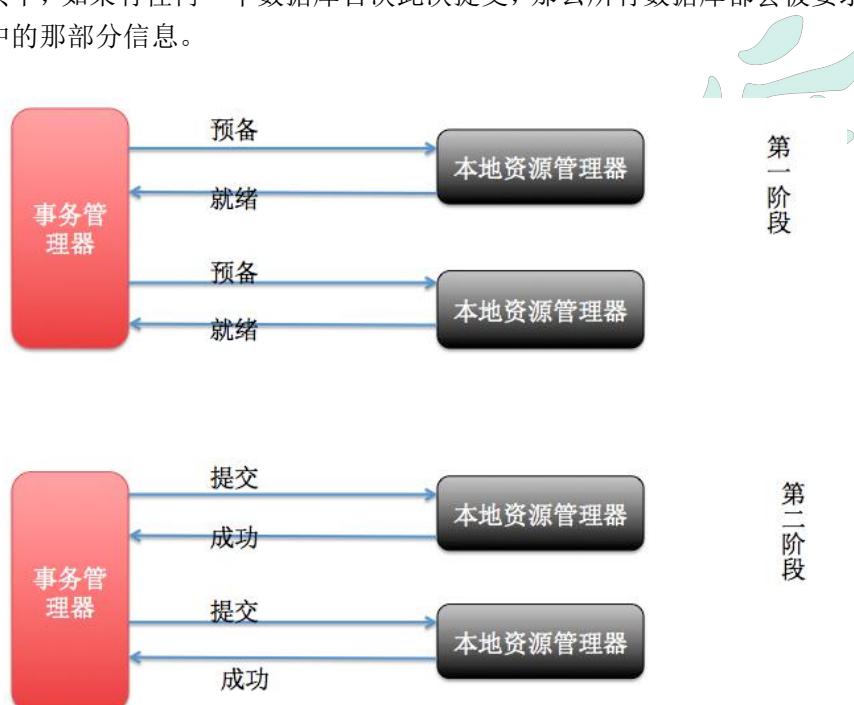
MySQL 从 5.5 版本开始支持，SQL Server 2005 开始支持，Oracle 7 开始支持。

其中，XA 是一个两阶段提交协议，该协议分为以下两个阶段：

第一阶段：事务协调器要求每个涉及到事务的数据库预提交(precommit)此操作，并反映是否可以提交。

第二阶段：事务协调器要求每个数据库提交数据。

其中，如果有任何一个数据库否决此次提交，那么所有数据库都会被要求回滚它们在此事务中的那部分信息。



- XA 协议比较简单，而且一旦商业数据库实现了 XA 协议，使用分布式事务的成本也比较低。
- **XA 性能不理想**，特别是在交易下单链路，往往并发量很高，XA 无法满足高并发场景
- XA 目前在商业数据库支持的比较理想，在 mysql 数据库中支持的不太理想，mysql 的 XA 实现，没有记录 prepare 阶段日志，主备切换回导致主库与备库数据不一致。
- 许多 nosql 也没有支持 XA，这让 XA 的应用场景变得非常狭隘。
- 也有 3PC，引入了超时机制（无论协调者还是参与者，在向对方发送请求后，若长时间未收到回应则做出相应处理）

2) 柔性事务-TCC 事务补偿型方案

刚性事务：遵循 ACID 原则，强一致性。

柔性事务：遵循 BASE 理论，最终一致性；

与刚性事务不同，柔性事务允许一定时间内，不同节点的数据不一致，但要求最终一致。

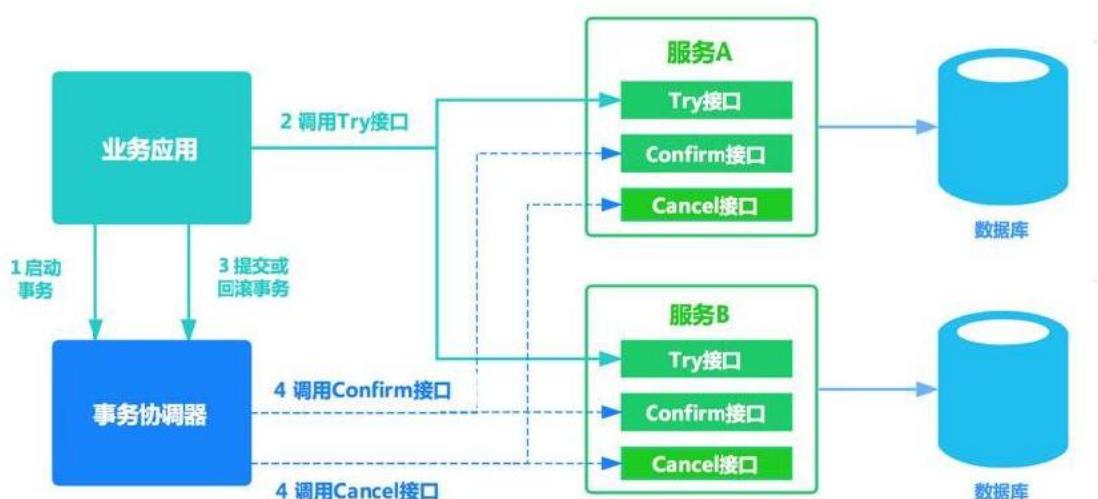


一阶段 prepare 行为：调用 自定义 的 prepare 逻辑。

二阶段 commit 行为：调用 自定义 的 commit 逻辑。

二阶段 rollback 行为：调用 自定义 的 rollback 逻辑。

所谓 TCC 模式，是指支持把 自定义 的分支事务纳入到全局事务的管理中。



3) 、柔性事务-最大努力通知型方案

按规律进行通知，不保证数据一定能通知成功，但会提供可查询操作接口进行核对。这种方案主要用在与第三方系统通讯时，比如：调用微信或支付宝支付后的支付结果通知。这种方案也是结合 MQ 进行实现，例如：通过 MQ 发送 http 请求，设置最大通知次数。达到通知次数后即不再通知。

案例：银行通知、商户通知等（各大交易业务平台间的商户通知：多次通知、查询校对、对账文件），支付宝的支付成功异步回调



4)、柔性事务-**可靠消息+最终一致性方案（异步确保型）**

实现：业务处理服务在业务事务提交之前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不是真正的发送。业务处理服务在业务事务提交之后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才会真正发送。

防止消息丢失：

```
/**  
 * 1、做好消息确认机制 (publisher, consumer 【手动ack】)  
 * 2、每一个发送的消息都在数据库做好记录。定期将失败的消息再次发送一遍  
 */
```





```
CREATE TABLE `mq_message` (
  `message_id` char(32) NOT NULL,
  `content` text,
  `to_exchane` varchar(255) DEFAULT NULL,
  `routing_key` varchar(255) DEFAULT NULL,
  `class_type` varchar(255) DEFAULT NULL,
  `message_status` int(1) DEFAULT '0' COMMENT '0-新建 1-已发送 2-错误抵达 3-已抵达',
  `create_time` datetime DEFAULT NULL,
  `update_time` datetime DEFAULT NULL,
  PRIMARY KEY (`message_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```





让天下没有难学的技术



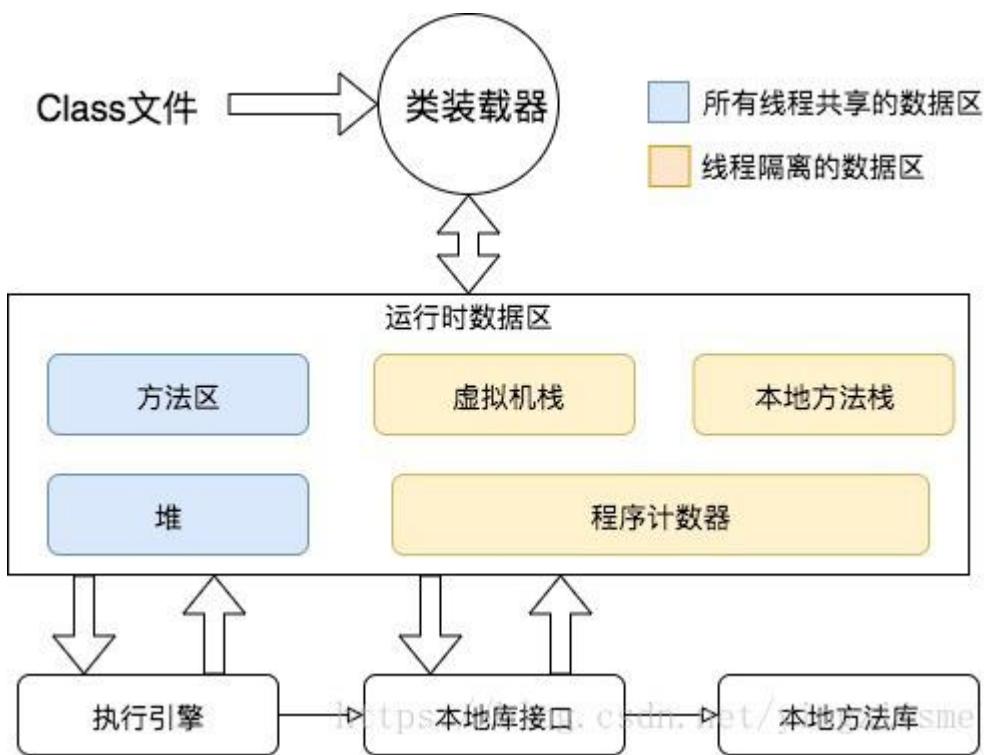
Gulimall

性能与压力测试

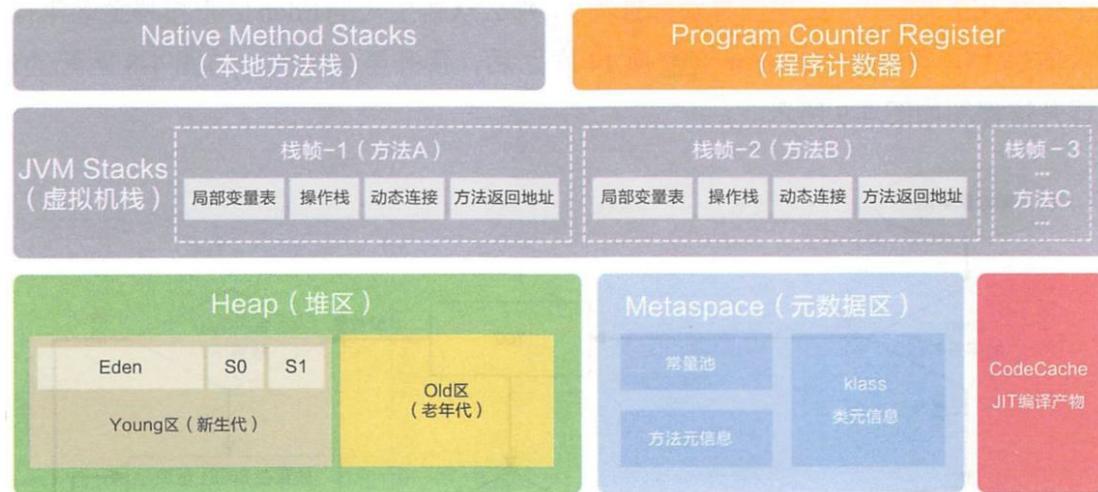


一、性能监控

1、jvm 内存模型



- 程序计数器 Program Counter Register:
 - 记录的是正在执行的虚拟机字节码指令的地址，
 - 此内存区域是唯一一个在 JAVA 虚拟机规范中没有规定任何 OutOfMemoryError 的区域
- 虚拟机: VM Stack
 - 描述的是 JAVA 方法执行的内存模型，每个方法在执行的时候都会创建一个栈帧，用于存储局部变量表，操作数栈，动态链接，方法接口等信息
 - 局部变量表存储了编译期可知的各种基本数据类型、对象引用
 - 线程请求的栈深度不够会报 StackOverflowError 异常
 - 栈动态扩展的容量不够会报 OutOfMemoryError 异常
 - 虚拟机栈是线程隔离的，即每个线程都有自己独立的虚拟机栈
- 本地方法: Native Stack
 - 本地方法栈类似于虚拟机栈，只不过本地方法栈使用的是本地方法
- 堆: Heap
 - 几乎所有的对象实例都在堆上分配内存



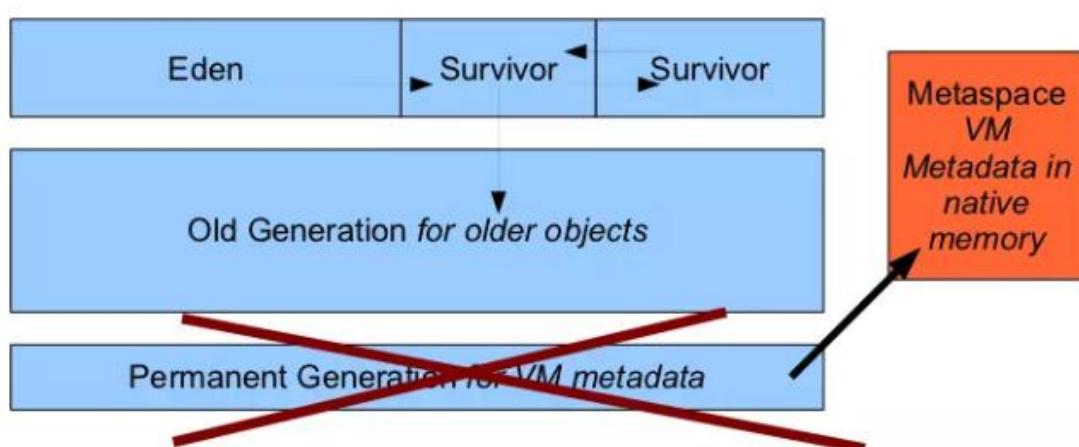
2、堆

所有的对象实例以及数组都要在堆上分配。堆是垃圾收集器管理的主要区域，也被称为“GC堆”；也是我们优化最多考虑的地方。

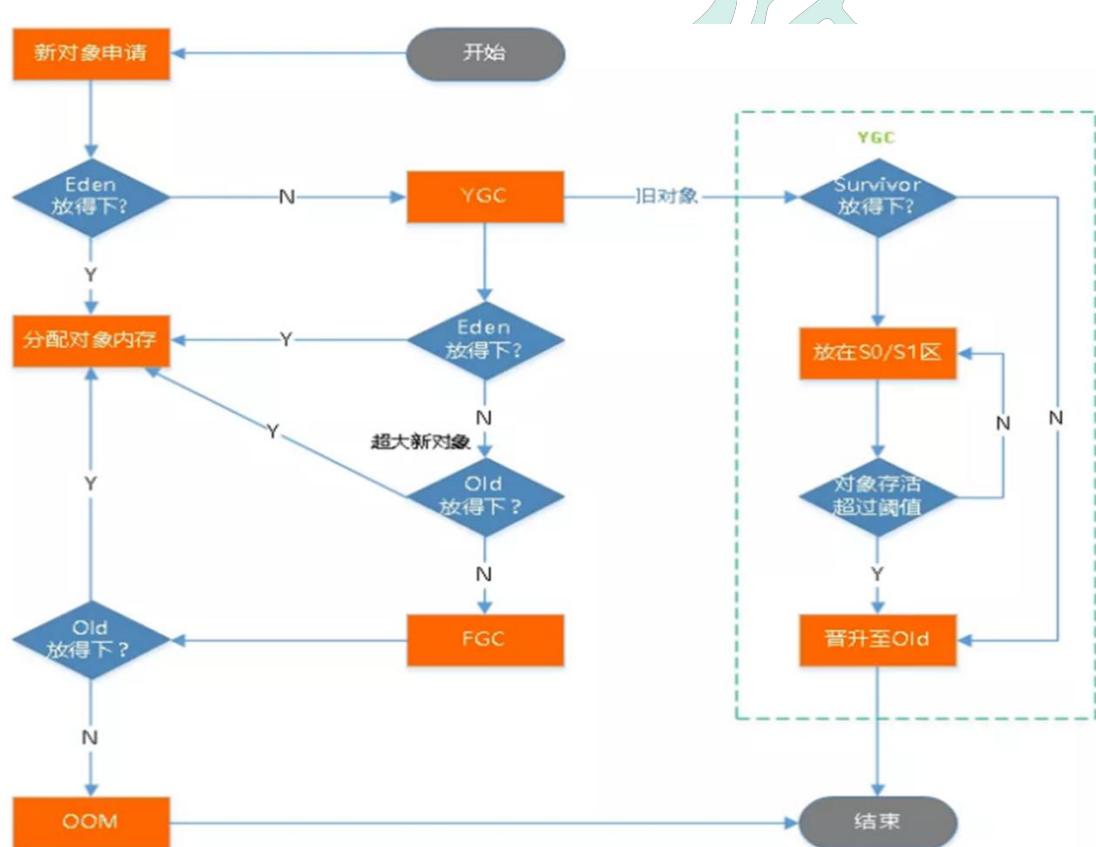
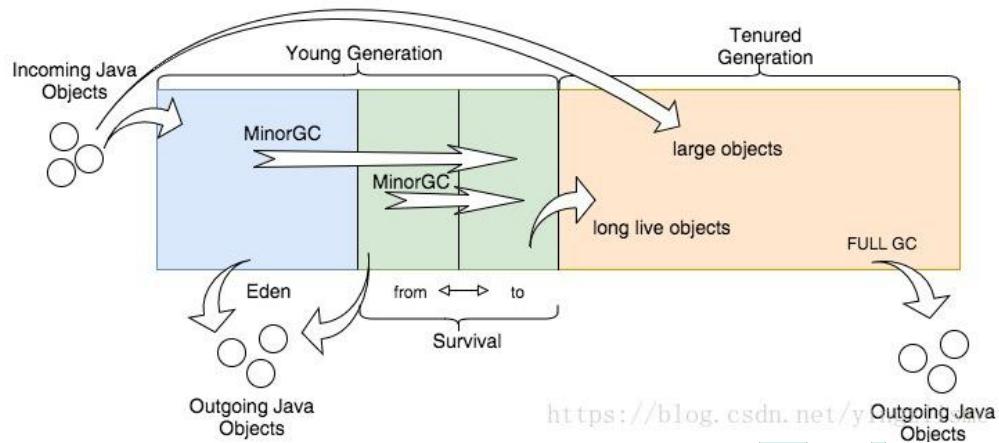
堆可以细分为：

- 新生代
 - Eden 空间
 - From Survivor 空间
 - To Survivor 空间
- 老年代
- 永久代/元空间
 - Java8 以前永久代，受 jvm 管理，java8 以后元空间，直接使用物理内存。因此，默认情况下，元空间的大小仅受本地内存限制。

垃圾回收



从 Java8 开始，HotSpot 已经完全将永久代（Permanent Generation）移除，取而代之的是一个新的区域—元空间（Metaspace）



3、jconsole 与 jvisualvm

Jdk 的两个小工具 jconsole、jvisualvm (升级版的 jconsole);通过命令行启动，可监控本地和远程应用。远程应用需要配置

1、jvisualvm 能干什么

监控内存泄露，跟踪垃圾回收，执行时内存、cpu 分析，线程分析...



运行：正在运行的

休眠：sleep

等待：wait

驻留：线程池里面的空闲线程

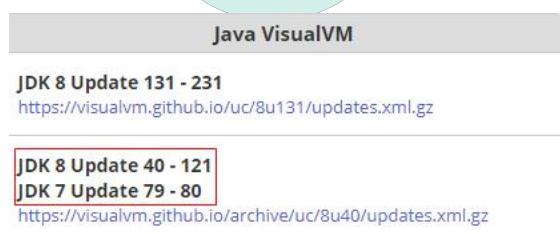
监视：阻塞的线程，正在等待锁

2、安装插件方便查看 gc

- Cmd 启动 jvisualvm
- 工具->插件



- 如果 503 错误解决：
 - 打开网址 <https://visualvm.github.io/pluginscenters.html>
 - cmd 查看自己的 jdk 版本，找到对应的



- 复制下面查询出来的链接。并重新设置上即可



4、监控指标

1、中间件指标

常用的中间件例如Tomcat、Weblogic等指标主要包括JVM, ThreadPool, JDBC,具体如下：

一级指标	二级指标	单位	解释
GC	GC频率	每秒多少次	java虚拟机垃圾部分回收频率
	Full GC频率	每小时多少次	java虚拟机垃圾完全回收频率
	Full GC平均时长	秒	用于垃圾完全回收的平均时长
	Full GC最大时长	秒	用于垃圾完全回收的最大时长
	堆使用率	百分比	堆使用率
ThreadPool	Active Thread Count	个	活动的线程数
	Pending User Request	个	处于排队的用户请求个数
JDBC	JDBC Active Connection	个	JDBC活动连接数

- 当前正在运行的线程数不能超过设定的最大值。一般情况下系统性能较好的情况下，线程数最小值设置 50 和最大值设置 200 比较合适。
- 当前运行的 JDBC 连接数不能超过设定的最大值。一般情况下系统性能较好的情况下， JDBC 最小值设置 50 和最大值设置 200 比较合适。
- G C 频率不能频繁，特别是 FULL GC 更不能频繁，一般情况下系统性能较好的情况下， JVM 最小堆大小和最大堆大小分别设置 1024M 比较合适。

2、数据库指标

常用的数据库例如MySQL指标主要包括SQL、吞吐量、缓存命中率、连接数等，具体如下：

一级指标	二级指标	单位	解释
吞吐量	耗时	微秒	执行SQL耗时
	QPS	个	每秒查询次数
	TPS	个	每秒事务次数
命中率	Key Buffer命中率	百分之	索引缓冲区命中率
	InnoDB Buffer命中率	百分之	InnoDB缓冲区命中率
	Query Cache命中率	百分之	查询缓存命中率
	Table Cache命中率	百分之	表缓存命中率
	Thread Cache命中率	百分之	线程缓存命中率
锁	等待次数	次	锁等待次数
	等待时间	微秒	锁等待时间

- SQL 耗时越小越好，一般情况下微秒级别。
- 命中率越高越好，一般情况下不能低于 95%。
- 锁等待次数越低越好，等待时间越短越好。

压测内容	压测线程数	吞吐量/s	90%响应时间	99%响应时间
Nginx	50	2335	11	944
Gateway	50	10367	8	31
简单服务	50	11341	8	17
首页一级菜单渲染	50	270(db,thymeleaf)	267	365
首页渲染(开缓存)	50	290	251	365
首页渲染(开缓存、优化数据库、关日志)	50	700	105	183
三级分类数据获取	50	2(db)/8(加索引)
三级分类(优化业务)	50	111	571	896
三级分类(使用redis作为缓存)	50	411	153	217
首页全量数据获取	50	7(静态资源)		
Nginx+Gateway	50			

Gateway+简单服务	50	3126	30	125
全链路	50	800	88	310

- 中间件越多，性能损失越大，大多都损失在网络交互了；
- 业务：
 - Db（MySQL 优化）
 - 模板的渲染速度（缓存）
 - 静态资源

5、JVM 分析&调优

jvm 调优，调的是稳定，并不能带给你性能的大幅提升。服务稳定的重要性就不用多说了，保证服务的稳定，gc 永远会是 Java 程序员需要考虑的不稳定因素之一。复杂和高并发下的服务，必须保证每次 gc 不会出现性能下降，各种性能指标不会出现波动，gc 回收规律而且干净，找到合适的 jvm 设置。Full gc 最会影响性能，根据代码问题，避免 full gc 频率。可以适当调大年轻代容量，让大对象可以在年轻代触发 young gc，调整大对象在年轻代的回收频次，尽可能保证大对象在年轻代回收，减小老年代缩短回收时间；

1、几个常用工具

jstack	查看 jvm 线程运行状态，是否有死锁现象等等信息
jinfo	可以输出并修改运行时的 java 进程的 opts。
jps	与 unix 上的 ps 类似，用来显示本地的 java 进程，可以查看本地运行着几个 java 程序，并显示他们的进程号。
jstat	一个极强的监视 VM 内存工具。可以用来监视 VM 内存内的各种堆和非堆的大小及其内存使用量。
jmap	打印出某个 java 进程（使用 pid）内存内的所有'对象'的情况（如：产生那些对象，及其数量）

2、命令示例

jstat 工具特别强大，有众多的可选项，详细查看堆内各个部分的使用量，以及加载类的数量。使用时，需加上查看进程的进程 id，和所选参数。

jstat -class pid	显示加载 class 的数量，及所占空间等信息
jstat -compiler pid	显示 VM 实时编译的数量等信息。
jstat -gc pid	可以显示 gc 的信息，查看 gc 的次数，及时间
jstat -gccapacity pid	堆内存统计，三代（young,old,perm）内存使用和占用大小
jstat -gcnew pid	新生代垃圾回收统计

jstat -gcnewcapacity pid	新生代内存统计
jstat -gcold pid	老年代垃圾回收统计
除了以上一个参数外，还可以同时加上 两个数字，如：jstat -printcompilation 3024 250 6 是每 250 毫秒打印一次，一共打印 6 次，还可以加上-h3 每三行显示一下标题。	
jstat -gcutil pid 1000 100 :1000ms 统计一次 gc 情况统计 100 次；	

在使用这些工具前，先用 JPS 命令获取当前的每个 JVM 进程号，然后选择要查看的 JVM。

jinfo 是 JDK 自带的命令，可以用来查看正在运行的 java 应用程序的扩展参数，包括 Java System 属性和 JVM 命令行参数；也可以动态的修改正在运行的 JVM 一些参数。当系统崩溃时，jinfo 可以从 core 文件里面知道崩溃的 Java 应用程序的配置信息	
jinfo pid	输出当前 jvm 进程的全部参数和系统属性
jinfo -flag name pid	可以查看指定的 jvm 参数的值；打印结果：-无此参数，+有
jinfo -flag [+ -]name pid	开启或者关闭对应名称的参数（无需重启虚拟机）
jinfo -flag name=value pid	修改指定参数的值
jinfo -flags pid	输出全部的参数
jinfo -sysprops pid	输出当前 jvm 进行的全部的系统属性

jmap 可以生成 heap dump 文件，也可以查看堆内对象分析内存信息等，如果不使用这个命令，还可以使用-XX:+HeapDumpOnOutOfMemoryError 参数来让虚拟机出现 OOM 的时候自动生成 dump 文件。

jmap -dump:live,format=b,file=dump.hprof pid	dump 堆到文件，format 指定输出格式，live 指明是活着的对象，file 指定文件名。eclipse 可以打开这个文件
jmap -heap pid	打印 heap 的概要信息，GC 使用的算法，heap 的配置和使用情况，可以用此来判断内存目前的使用情况以及垃圾回收情况
jmap -finalizerinfo pid	打印等待回收的对象信息
jmap -histo:live pid	打印堆的对象统计，包括对象数、内存大小等。 jmap -histo:live 这个命令执行， JVM 会先触发 gc ，然后再统计信息
jmap -clstats pid	打印 Java 类加载器的智能统计信息，对于每个类加载器而言，对于每个类加载器而言，它的名称，活跃度，地址，父类加载器，它所加载的类的数量和大小都会被打印。此外，包含的字符串数量和大小也会被打印。
-F	强制模式。如果指定的 pid 没有响应，请使用 jmap -dump 或 jmap -histo 选项。此模式下，不支持 live 子选项。
jmap -F -histo pid	
jstack	是 jdk 自带的线程堆栈分析工具，使用该命令可以查看或导出 Java 应用程序中线程堆栈信息。
jstack pid	输出当前 jvm 进程的全部参数和系统属性

3、调优项

官方文档: <https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html#BGBCIEFC>

二、压力测试

压力测试考察当前软硬件环境下系统所能承受的最大负荷并帮助找出系统瓶颈所在。压测都是为了系统在线上的处理能力和稳定性维持在一个标准范围内，做到心中有数。

使用压力测试，我们有希望找到很多种用其他测试方法更难发现的错误。有两种错误类型是：**内存泄漏，并发与同步。**

有效的压力测试系统将应用以下这些关键条件:**重复，并发，量级，随机变化。**

1、性能指标

- 响应时间（Response Time: RT）
响应时间指用户从客户端发起一个请求开始，到客户端接收到从服务器端返回的响应结束，整个过程所耗费的时间。
- HPS（Hits Per Second）：每秒点击次数，单位是次/秒。
- TPS（Transaction per Second）：系统每秒处理交易数，单位是笔/秒。
- QPS（Query per Second）：系统每秒处理查询次数，单位是次/秒。
对于互联网业务中，如果某些业务有且仅有一个请求连接，那么 $TPS=QPS=HPS$ ，一般情况下用 **TPS** 来衡量整个业务流程，用 **QPS** 来衡量接口查询次数，用 **HPS** 来表示对服务器单击请求。
- 无论 **TPS**、**QPS**、**HPS**,此指标是衡量系统处理能力非常重要的指标，越大越好，根据经验，一般情况下：
 - 金融行业：1000TPS~50000TPS，不包括互联网化的活动
 - 保险行业：100TPS~100000TPS，不包括互联网化的活动
 - 制造行业：10TPS~5000TPS
 - 互联网电子商务：10000TPS~1000000TPS
 - 互联网中型网站：1000TPS~50000TPS
 - 互联网小型网站：500TPS~10000TPS
- 最大响应时间（Max Response Time）指用户发出请求或者指令到系统做出反应（响应）的最大时间。
- 最少响应时间（Mininum ResponseTime）指用户发出请求或者指令到系统做出反应（响应）的最少时间。
- 90%响应时间（90% Response Time）是指所有用户的响应时间进行排序，第 90% 的响

应时间。

- 从外部看，性能测试主要关注如下三个指标

吞吐量：每秒钟系统能够处理的请求数、任务数。

响应时间：服务处理一个请求或一个任务的耗时。

错误率：一批请求中结果出错的请求所占比例。

2、JMeter

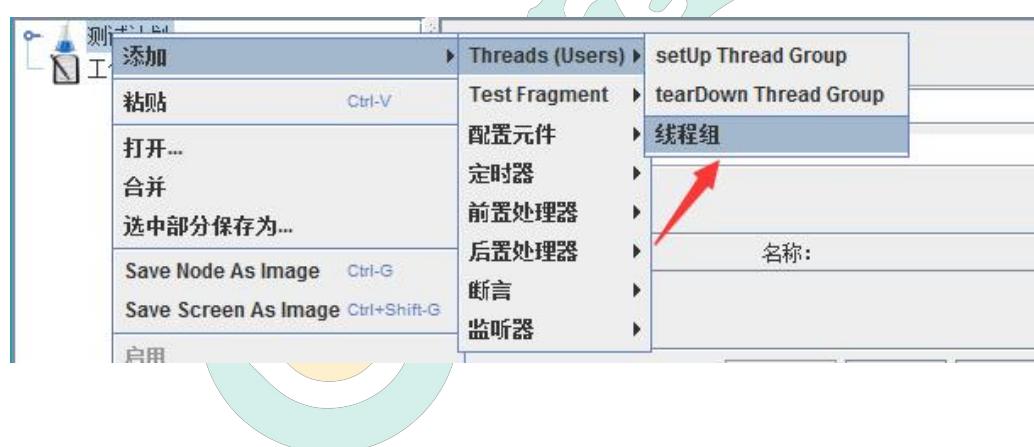
1、JMeter 安装

https://jmeter.apache.org/download_jmeter.cgi

下载对应的压缩包，解压运行 jmeter.bat 即可

2、JMeter 压测示例

1、添加线程组



线程组

名称:	线程组
注释:	在取样器错误后要执行的动作
<input checked="" type="radio"/> 继续 <input type="radio"/> 启动下一进程循环 <input type="radio"/> 停止线程	
线程属性	
线程数:	100
Ramp-Up时间(秒):	1
循环次数	<input type="checkbox"/> 永远 100
<input type="checkbox"/> 延迟创建线程直到需要	
<input type="checkbox"/> 调度器	
调度器配置	
⚠ If Loop Count is not -1 or Forever, duration will be min(Duration, Loop Count * iteration duration)	
持续时间(秒)	
启动延迟(秒)	

线程组参数详解:

- 线程数: 虚拟用户数。一个虚拟用户占用一个进程或线程。设置多少虚拟用户数在这里也就是设置多少个线程数。
- Ramp-Up Period(in seconds)准备时长: 设置的虚拟用户数需要多长时间全部启动。如果线程数为 10, 准备时长为 2, 那么需要 2 秒钟启动 10 个线程, 也就是每秒钟启动 5 个线程。
- 循环次数: 每个线程发送请求的次数。如果线程数为 10, 循环次数为 100, 那么每个线程发送 100 次请求。总请求数为 $10*100=1000$ 。如果勾选了“永远”, 那么所有线程会一直发送请求, 一到选择停止运行脚本。
- Delay Thread creation until needed: 直到需要时延迟线程的创建。
- 调度器: 设置线程组启动的开始时间和结束时间(配置调度器时, 需要勾选循环次数为永远)
- 持续时间(秒): 测试持续时间, 会覆盖结束时间
- 启动延迟(秒): 测试延迟启动时间, 会覆盖启动时间
- 启动时间: 测试启动时间, 启动延迟会覆盖它。当启动时间已过, 手动只需测试时当前时间也会覆盖它。
- 结束时间: 测试结束时间, 持续时间会覆盖它。

2、添加 HTTP 请求

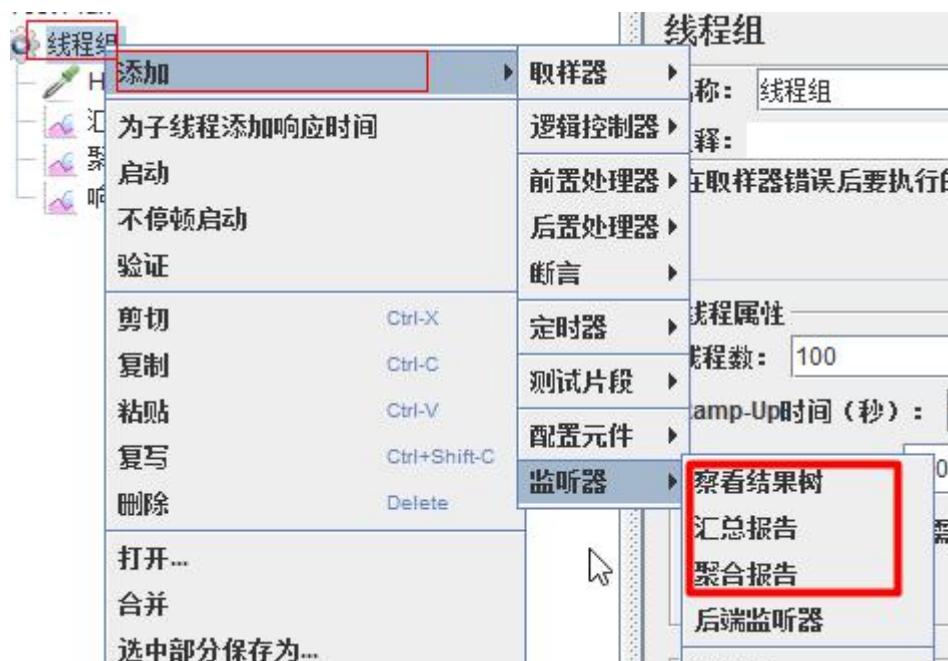
The screenshot shows the JMeter interface with the '线程组' (Thread Group) selected in the tree view. A context menu is open over the '添加' (Add) option, specifically under the 'Sampler' section of the '逻辑控制器' (Logic Controller) submenu. The 'HTTP请求' (HTTP Request) item is highlighted with a red arrow.

HTTP Request configuration:

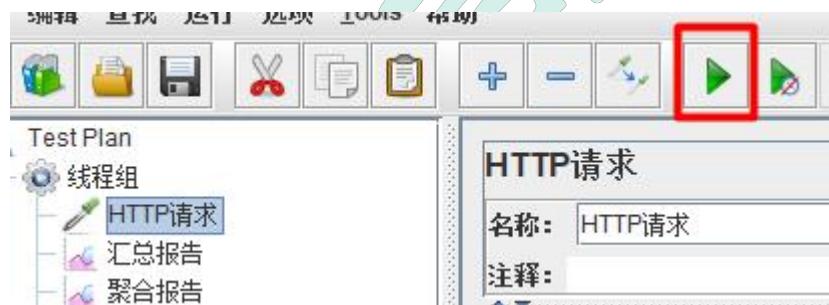
- Basic Tab:**
 - Protocol [http]: http
 - Server Name or IP: 192.168.56.10
 - Port Number: 9200
- Advanced Tab:**
 - Method: GET
 - Path: (empty)
 - Content encoding: (empty)
 - Checkboxes: Redirect Automatically, Follow Redirects, Use KeepAlive, Use multipart/form-data, Browser-compatible headers
- Parameters Tab:** Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equa
(empty)	(empty)	(empty)	(empty)	(empty)

3、添加监听器



4、启动压测&查看分析结果



结果分析

- 有错误率同开发确认，确定是否允许错误的发生或者错误率允许在多大的范围内；
- Throughput 吞吐量每秒请求的数大于并发数，则可以慢慢的往上面增加；若在压测的机器性能很好的情况下，出现吞吐量小于并发数，说明并发数不能再增加了，可以慢慢的往下减，找到最佳的并发数；
- 压测结束，登陆相应的 web 服务器查看 CPU 等性能指标，进行数据的分析；
- 最大的 tps，不断的增加并发数，加到 tps 达到一定值开始出现下降，那么那个值就是最大的 tps。
- 最大的并发数：最大的并发数和最大的 tps 是不同的概率，一般不断增加并发数，达到一个值后，服务器出现请求超时，则可认为该值为最大的并发数。
- 压测过程出现性能瓶颈，若压力机任务管理器查看到的 cpu、网络和 cpu 都正常，未达

到 90%以上，则可以说明服务器有问题，压力机没有问题。

- 影响性能考虑点包括：

数据库、应用程序、中间件（tomcat、Nginx）、网络和操作系统等方面

- 首先考虑自己的应用属于 **CPU 密集型** 还是 **IO 密集型**

3、JMeter Address Already in use 错误解决

windows 本身提供的端口访问机制的问题。

Windows 提供给 TCP/IP 链接的端口为 1024-5000，并且要四分钟来循环回收他们。就导致我们在短时间内跑大量的请求时将端口占满了。

1.cmd 中，用 regedit 命令打开注册表

2.在 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters 下，

1.右击 parameters，添加一个新的 DWORD，名字为 MaxUserPort

2.然后双击 MaxUserPort，输入数值数据为 65534，基数选择十进制（如果是分布式运行的话，控制机器和负载机器都需要这样操作哦）

3. 修改配置完毕之后记得重启机器才会生效

<https://support.microsoft.com/zh-cn/help/196271/when-you-try-to-connect-from-tcp-ports-greater-than-5000-you-receive-t>

TCPTimedWaitDelay: 30

Gulimall

缓存与分布式锁



一、缓存

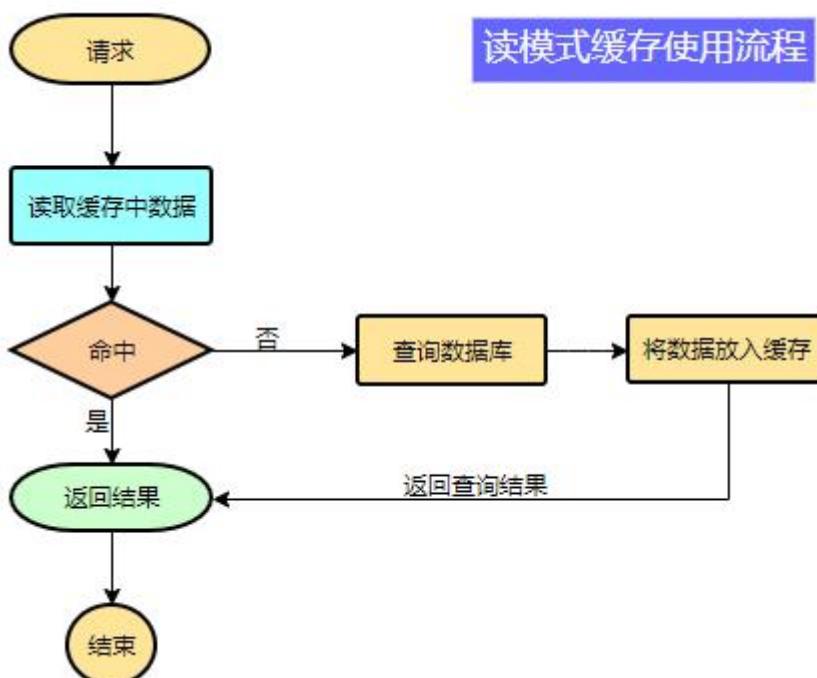
1、缓存使用

为了系统性能的提升，我们一般都会将部分数据放入缓存中，加速访问。而 db 承担数据落盘工作。

哪些数据适合放入缓存？

- 即时性、数据一致性要求不高的
- 访问量大且更新频率不高的数据（读多，写少）

举例：电商类应用，商品分类，商品列表等适合缓存并加一个失效时间(根据数据更新频率来定)，后台如果发布一个商品，买家需要 5 分钟才能看到新的商品一般还是可以接受的。



```
data = cache.load(id); //从缓存加载数据
if(data == null){
    data = db.load(id); //从数据库加载数据
    cache.put(id,data); //保存到 cache 中
}
return data;
```

注意：在开发中，凡是放入缓存中的数据我们都应该指定过期时间，使其可以在系统即使没有主动更新数据也能自动触发数据加载进缓存的流程。避免业务崩溃导致的数据永久不一致问题。

2、整合 redis 作为缓存

1、引入 redis-starter

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

2、配置 redis

```
spring:
  redis:
    host: 192.168.56.10
    port: 6379
```

3、使用 RedisTemplate 操作 redis

```
@Autowired
StringRedisTemplate stringRedisTemplate;
@Test
public void testStringRedisTemplate(){
    ValueOperations<String, String> ops = stringRedisTemplate.opsForValue();
    ops.set("hello", "world_" + UUID.randomUUID().toString());
    String hello = ops.get("hello");
    System.out.println(hello);
}
```

4、切换使用 jedis

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <exclusions>
        <exclusion>
            <groupId>io.lettuce</groupId>
            <artifactId>lettuce-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

二、缓存失效问题

先来解决大并发读情况下的缓存失效问题：

1、缓存穿透

- 缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中，将去查询数据库，但是数据库也无此记录，我们没有将这次查询的 null 写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。
- 在流量大时，可能 DB 就挂掉了，要是有人利用不存在的 key 频繁攻击我们的应用，这就是漏洞。
- 解决：
缓存空结果、并且设置短的过期时间。

2、缓存雪崩

- 缓存雪崩是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到 DB，DB 瞬时压力过重雪崩。
- 解决：
原有的失效时间基础上增加一个随机值，比如 1-5 分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

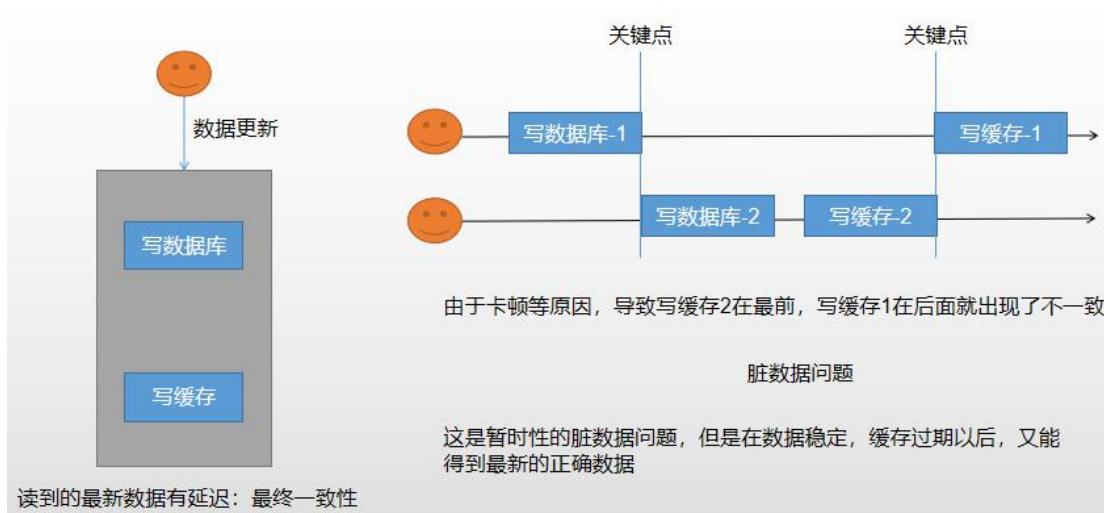
3、缓存击穿

- 对于一些设置了过期时间的 key，如果这些 key 可能在某些时间点被超高并发地访问，是一种非常“热点”的数据。
- 这个时候，需要考虑一个问题：如果这个 key 在大量请求同时进来前正好失效，那么所有对这个 key 的数据查询都落到 db，我们称为缓存击穿。
- 解决：
加锁

三、缓存数据一致性

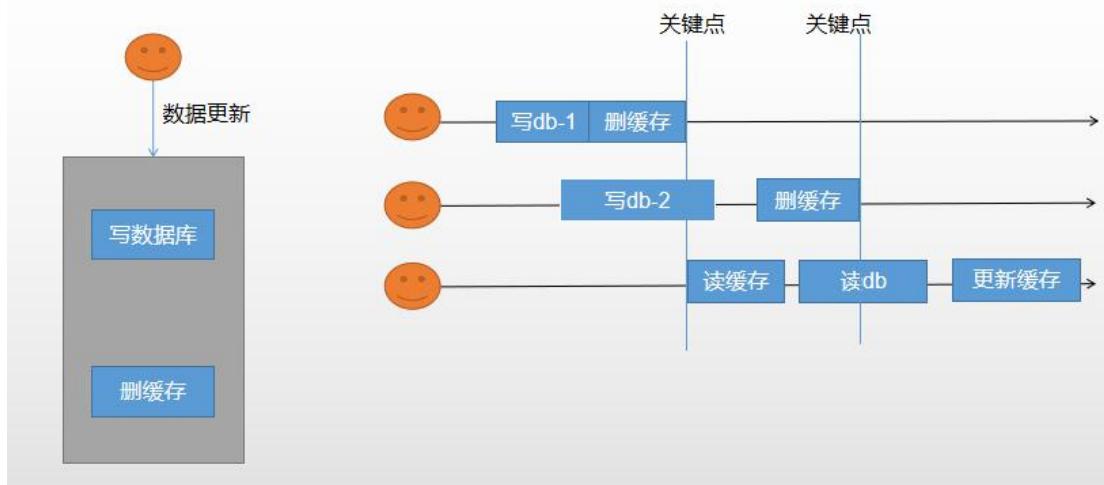
1、保证一致性模式

1、双写模式



2、失效模式

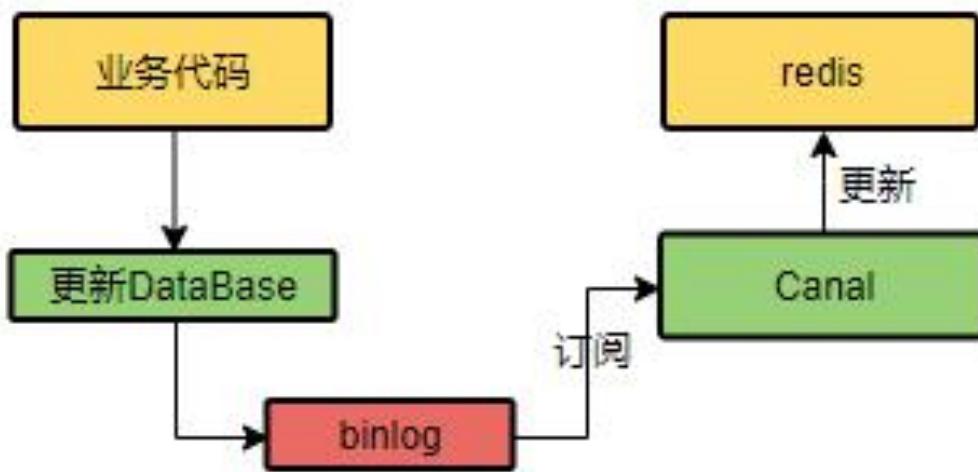
缓存一致性-失效模式



3、改进方法 1-分布式读写锁

分布式读写锁。读数据等待写数据整个操作完成

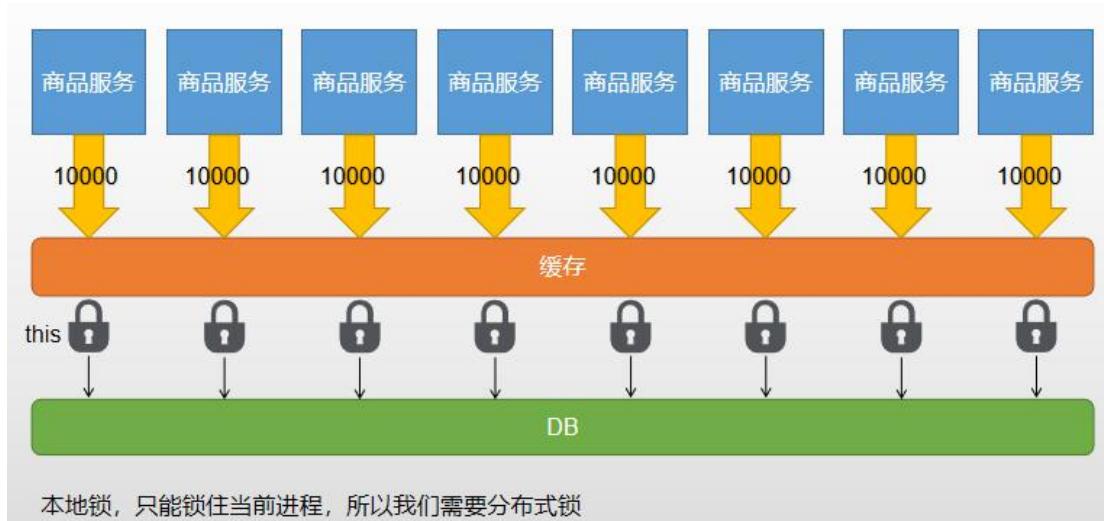
4、改进方法 2-使用 canal



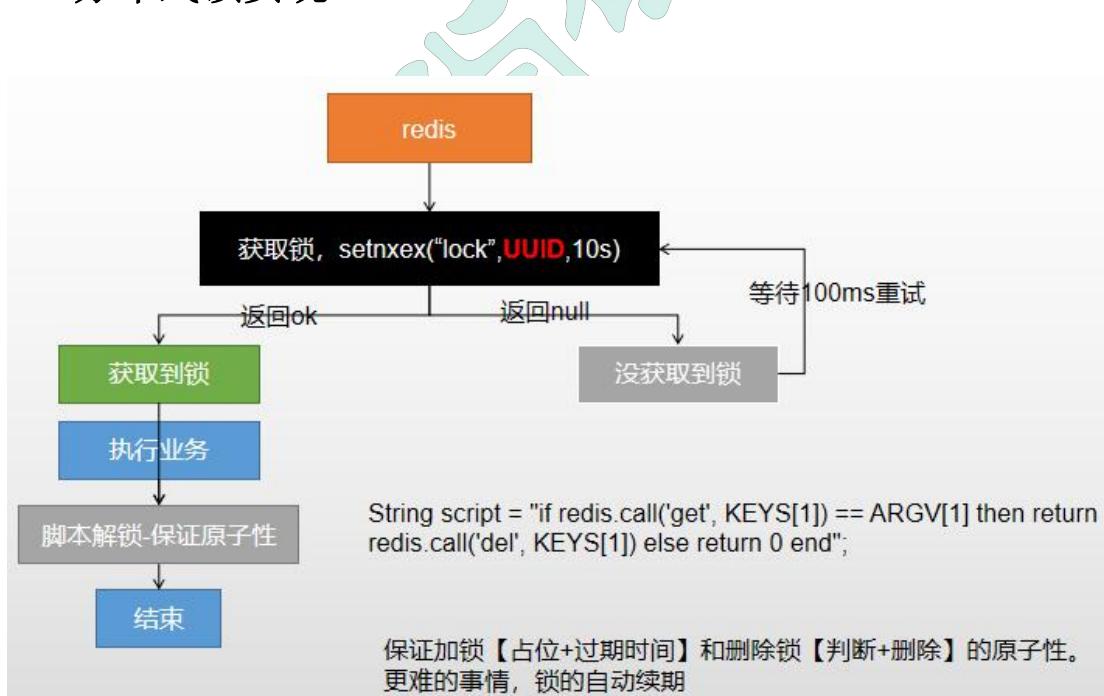
尚硅谷

四、分布式锁

1、分布式锁与本地锁



2、分布式锁实现



使用 RedisTemplate 操作分布式锁

```
public Map<String, List<Catalog2Vo>> getCatalogJsonFromDbWithRedisLock() {
```

```
//1、占分布式锁。去redis 占坑
```

```
String uuid = UUID.randomUUID().toString();
Boolean lock =
redisTemplate.opsForValue().setIfAbsent("lock",uuid,300,TimeUnit.SECONDS);
if(lock){
    System.out.println("获取分布式锁成功...");
    //加锁成功... 执行业务
    //2、设置过期时间，必须和加锁是同步的，原子的
    //redisTemplate.expire("lock",30,TimeUnit.SECONDS);
    Map<String, List<Catalog2Vo>> dataFromDb;
    try{
        dataFromDb = getDataFromDb();
    }finally {
        String script = "if redis.call('get', KEYS[1]) == ARGV[1] then
return redis.call('del', KEYS[1]) else return 0 end";
        //删除锁
        Long lock1 = redisTemplate.execute(new
DefaultRedisScript<Long>(script, Long.class)
        , Arrays.asList("lock"), uuid);
    }

    //获取值对比+对比成功删除=原子操作 Lua 脚本解锁
    //String lockValue = redisTemplate.opsForValue().get("lock");
    //if(uuid.equals(lockValue)){
    //    //删除我自己的锁
    //    redisTemplate.delete("lock");//删除锁
    //}
    return dataFromDb;
}else {
    //加锁失败...重试。synchronized ()
    //休眠 100ms 重试
    System.out.println("获取分布式锁失败...等待重试");
    try{
        Thread.sleep(200);
    }catch (Exception e){

    }

    return getCatalogJsonFromDbWithRedisLock();//自旋的方式
}

}
```

3、Redisson 完成分布式锁

1、简介

Redisson 是架设在 Redis 基础上的一个 Java 驻内存数据网格（In-Memory Data Grid）。充分的利用了 Redis 键值数据库提供的一系列优势，基于 Java 实用工具包中常用接口，为使用者提供了一系列具有分布式特性的常用工具类。使得原本作为协调单机多线程并发程序的工具包获得了协调分布式多机多线程并发系统的能力，大大降低了设计和研发大规模分布式系统的难度。同时结合各富特色的分布式服务，更进一步简化了分布式环境中程序相互之间的协作。

官方文档：<https://github.com/redisson/redisson/wiki/%E7%9B%AE%E5%BD%95>

2、配置

```
// 默认连接地址 127.0.0.1:6379
RedissonClient redisson = Redisson.create();

Config config = new Config();
//redis://127.0.0.1:7181
//可以用"rediss://"来启用 SSL 连接
config.useSingleServer().setAddress("redis://192.168.56.10:6379");
RedissonClient redisson = Redisson.create(config);
```

3、使用分布式锁

```
RLock lock = redisson.getLock("anyLock");// 最常见的使用方法
lock.lock();
// 加锁以后 10 秒钟自动解锁// 无需调用 unlock 方法手动解锁
lock.lock(10, TimeUnit.SECONDS);
// 尝试加锁，最多等待 100 秒，上锁以后 10 秒自动解锁 boolean res = lock.tryLock(100,
10, TimeUnit.SECONDS);
if (res) {
    try {
        ...
    } finally {
        lock.unlock();
    }
}
```



让天下没有难学的技术

4、使用其他

参照官方文档进行测试

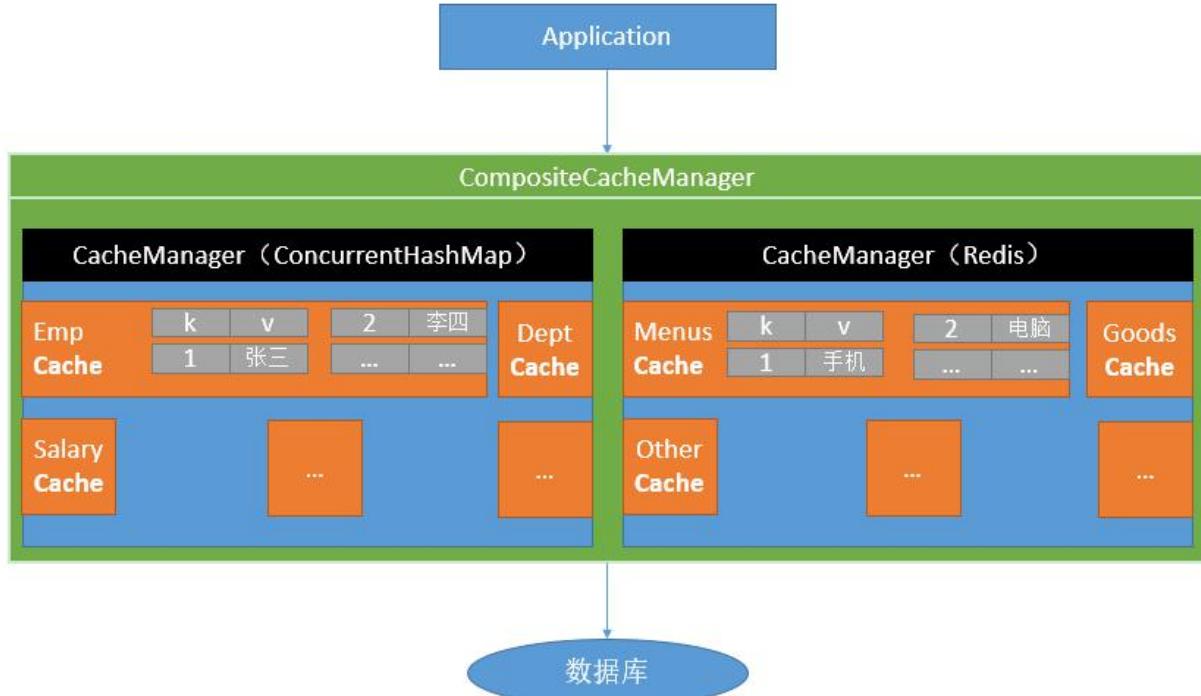


五、Spring Cache

1、简介

- Spring 从 3.1 开始定义了 `org.springframework.cache.Cache` 和 `org.springframework.cache.CacheManager` 接口来统一不同的缓存技术；并支持使用 JCache (JSR-107) 注解简化我们开发；
- `Cache` 接口为缓存的组件规范定义，包含缓存的各种操作集合；`Cache` 接口下 Spring 提供了各种 `xxxCache` 的实现；如 `RedisCache`，`EhCacheCache`，`ConcurrentMapCache` 等；
- 每次调用需要缓存功能的方法时，Spring 会检查指定参数的指定的目标方法是否已经被调用过；如果有就直接从缓存中获取方法调用后的结果，如果没有就调用方法并缓存结果后返回给用户。下次调用直接从缓存中获取。
- 使用 Spring 缓存抽象时我们需要关注以下两点：
 - 1、确定方法需要被缓存以及他们的缓存策略
 - 2、从缓存中读取之前缓存存储的数据

2、基础概念



3、注解

Cache	缓存接口，定义缓存操作。实现有：RedisCache、EhCacheCache、ConcurrentMapCache等
CacheManager	缓存管理器，管理各种缓存（Cache）组件
@Cacheable	主要针对方法配置，能够根据方法的请求参数对其结果进行缓存
@CacheEvict	清空缓存
@CachePut	保证方法被调用，又希望结果被缓存。
@EnableCaching	开启基于注解的缓存
keyGenerator	缓存数据时key生成策略
serialize	缓存数据时value序列化策略

@Cacheable/@CachePut/@CacheEvict 主要的参数

value	缓存的名称，在spring配置文件中定义，必须指定至少一个	例如： @Cacheable(value="mycache") 或者 @Cacheable(value={"cache1","cache2"})
key	缓存的key，可以为空，如果指定要按照SpEL表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	例如： @Cacheable(value="testcache",key="#userName")
condition	缓存的条件，可以为空，使用SpEL编写，返回true或者false，只有为true才进行缓存/清除缓存，在调用方法之前之后都能判断	例如： @Cacheable(value="testcache",condition="#userName.length()>2")
allEntries (@CacheEvict)	是否清空所有缓存内容，缺省为false，如果指定为true，则方法调用后将立即清空所有缓存	例如： @CacheEvict(value="testcache",allEntries=true)
beforeInvocation (@CacheEvict)	是否在方法执行前就清空，缺省为false，如果指定为true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存	例如： @CacheEvict(value="testcache",beforeInvocation=true)
unless (@CachePut) (@Cacheable)	用于否决缓存的，不像condition，该表达式只在方法执行之后判断，此时可以拿到返回值result进行判断。条件为true不会缓存，fasle才缓存	例如： @Cacheable(value="testcache",unless="#result == null")

4、表达式语法

Cache SpEL available metadata

名字	位置	描述	示例
methodName	root object	当前被调用的方法名	#root.methodName
method	root object	当前被调用的方法	#root.method.name
target	root object	当前被调用的目标对象	#root.target
targetClass	root object	当前被调用的目标对象类	#root.targetClass
args	root object	当前被调用的方法的参数列表	#root.args[0]
caches	root object	当前方法调用使用的缓存列表（如@Cacheable(value={"cache1", "cache2"}))，则有两个cache	#root.caches[0].name
argument name	evaluation context	方法参数的名字，可以直接 #参数名，也可以使用 #p0或#a0 的形式，0代表参数的索引；	
result	evaluation context	方法执行后的返回值（仅当方法执行之后的判断有效，如‘unless’，‘cache put’的表达式 ‘cache evict’的表达式 beforeInvocation=false）	#result

5、缓存穿透问题解决



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

允许 null 值缓存





让天下没有难学的技术

谷粒商城

ElasticSearch-全文检索



简介

<https://www.elastic.co/cn/what-is/elasticsearch>

全文搜索属于最常见的需求，开源的 Elasticsearch 是目前全文搜索引擎的首选。它可以快速地储存、搜索和分析海量数据。维基百科、Stack Overflow、Github 都采用它



Elastic 的底层是开源库 Lucene。但是，你没法直接用 Lucene，必须自己写代码去调用它的接口。Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。

REST API：天然的跨平台。

官方文档：<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

官方中文：https://www.elastic.co/guide/cn/elasticsearch/guide/current/foreword_id.html

社区中文：

<https://es.xiaoleilu.com/index.html>

<http://doc.codingdict.com/elasticsearch/0/>

一、基本概念

1、Index（索引）

动词，相当于 MySQL 中的 insert;

名词，相当于 MySQL 中的 Database

2、Type（类型）

在 Index（索引）中，可以定义一个或多个类型。

类似于 MySQL 中的 Table；每一种类型的数据放在一起；

3、Document（文档）

保存在某个索引（Index）下，某种类型（Type）的一个数据（Document），文档是 JSON 格式的，Document 就像是 MySQL 中的某个 Table 里面的内容；

4、倒排索引机制

词	记录
红海	1,2,3,4,5
行动	1,2,3
探索	2,5
特别	3,5
记录篇	4
特工	5

二、Docker 安装 Es

1、下载镜像文件

docker pull elasticsearch:7.4.2 存储和检索数据
docker pull kibana:7.4.2 可视化检索数据

2、创建实例

1、ElasticSearch

```
mkdir -p /mydata/elasticsearch/config  
mkdir -p /mydata/elasticsearch/data  
echo "http.host: 0.0.0.0" >> /mydata/elasticsearch/config/elasticsearch.yml
```

```
chmod -R 777 /mydata/elasticsearch/ 保证权限  
docker run --name elasticsearch -p 9200:9200 -p 9300:9300 \  
-e "discovery.type=single-node" \  
-v /mydata/elasticsearch/config:/etc/elasticsearch/config  
-v /mydata/elasticsearch/data:/data/elasticsearch/data
```



```
-e ES_JAVA_OPTS="-Xms64m -Xmx512m" \
-v /mydata/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml \
-v /mydata/elasticsearch/data:/usr/share/elasticsearch/data \
-v /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins \
-d elasticsearch:7.4.2
```

以后再外面装好插件重启即可；

特别注意：

-e ES_JAVA_OPTS="-Xms64m -Xmx256m" \ 测试环境下，设置 ES 的初始内存和最大内存，否则导致过大启动不了 ES

2、Kibana

```
docker run --name kibana -e ELASTICSEARCH_HOSTS=http://192.168.56.10:9200 -p 5601:5601 \
-d kibana:7.4.2
```

http://192.168.56.10:9200 一定改为自己虚拟机的地址

三、初步检索

1、_cat

GET /_cat/nodes: 查看所有节点
GET /_cat/health: 查看 es 健康状况
GET /_cat/master: 查看主节点
GET /_cat/indices: 查看所有索引 show databases;

2、索引一个文档（保存）

保存一个数据，保存在哪个索引的哪个类型下，指定用哪个唯一标识

PUT customer/external/1; 在 customer 索引下的 external 类型下保存 1 号数据为

```
PUT customer/external/1
```

```
{  
    "name": "John Doe"  
}
```

PUT 和 POST 都可以，
POST 新增。如果不指定 id，会自动生成 id。指定 id 就会修改这个数据，并新增版本号

PUT 可以新增可以修改。PUT 必须指定 id；由于 PUT 需要指定 id，我们一般都用来做修改操作，不指定 id 会报错。

3、查询文档

```
GET customer/external/1
```

结果：

```
{  
    "_index": "customer",      // 在哪个索引  
    "_type": "external",       // 在哪个类型  
    "_id": "1",                // 记录 id  
    "_version": 2,              // 版本号  
    "_seq_no": 1,               // 并发控制字段，每次更新就会+1，用来做乐观锁  
    "_primary_term": 1,         // 同上，主分片重新分配，如重启，就会变化  
    "found": true,  
    "_source": {  
        "name": "John Doe"  
    }  
}
```

更新携带 ?if_seq_no=0&if_primary_term=1

4、更新文档

```
POST customer/external/1/_update  
{  
    "doc":{  
        "name": "John Doew"  
    }  
}
```

或者

```
POST customer/external/1  
{  
    "name": "John Doe2"  
}
```

或者

```
PUT customer/external/1
{
  "name": "John Doe"
}
```

- 不同：POST 操作会对比源文档数据，如果相同不会有操作，**文档 version 不增加**；
PUT 操作总会将数据重新保存并增加 version 版本；

带_update 对比元数据如果一样就不进行任何操作。

看场景：

对于大并发更新，不带 update；

对于大并发查询偶尔更新，带 update；对比更新，重新计算分配规则。

- 更新同时增加属性

```
POST customer/external/1/_update
{
  "doc": { "name": "Jane Doe", "age": 20 }
}
```

PUT 和 POST 不带_update 也可以

5、删除文档&索引

```
DELETE customer/external/1
```

```
DELETE customer
```

6、bulk 批量 API

```
POST customer/external/_bulk
{"index":{"_id":"1"}}
{"name": "John Doe" }
{"index":{"_id":"2"}}
 {"name": "Jane Doe" }
```

语法格式：

```
{ action: { metadata }}\n{ request body }\\n
```

```
{ action: { metadata }}\n{ request body }\\n
```

复杂实例：

```
POST /_bulk
{ "delete": { "_index": "website", "_type": "blog", "_id": "123" }}
{ "create": { "_index": "website", "_type": "blog", "_id": "123" }}
```

```
{ "title": "My first blog post" }
{ "index": { "_index": "website", "_type": "blog" }}
{ "title": "My second blog post" }
{ "update": { "_index": "website", "_type": "blog", "_id": "123", "_retry_on_conflict": 3} }
{ "doc": {"title": "My updated blog post"} }
```

bulk API 以此按顺序执行所有的 action (动作)。如果一个单个的动作因任何原因而失败，它将继续处理它后面剩余的动作。当 bulk API 返回时，它将提供每个动作的状态 (与发送的顺序相同)，所以您可以检查是否一个指定的动作是不是失败了。

7、样本测试数据

我准备了一份顾客银行账户信息的虚构的 JSON 文档样本。每个文档都有下列的 schema (模式)：

```
{
  "account_number": 0,
  "balance": 16623,
  "firstname": "Bradshaw",
  "lastname": "Mckenzie",
  "age": 29,
  "gender": "F",
  "address": "244 Columbus Place",
  "employer": "Euron",
  "email": "bradshawmckenzie@euron.com",
  "city": "Hobucken",
  "state": "CO"
}
```

<https://github.com/elastic/elasticsearch/blob/master/docs/src/test/resources/accounts.json?raw=true> 导入测试数据

POST bank/account/_bulk

测试数据

四、进阶检索

1、SearchAPI

ES 支持两种基本方式检索：

- 一个是通过使用 REST request URI 发送搜索参数 (uri+检索参数)
- 另一个是通过使用 REST request body 来发送它们 (uri+请求体)

1) 、检索信息

- 一切检索从 _search 开始

GET bank/_search	检索 bank 下所有信息，包括 type 和 docs
GET bank/_search?q=* &sort=account_number:asc	请求参数方式检索

响应结果解释：

took - Elasticsearch 执行搜索的时间（毫秒）

time_out - 告诉我们搜索是否超时

_shards - 告诉我们多少个分片被搜索了，以及统计了成功/失败的搜索分片

hits - 搜索结果

hits.total - 搜索结果

hits.hits - 实际的搜索结果数组（默认为前 10 的文档）

sort - 结果的排序 key（键）（没有则按 score 排序）

score 和 max_score - 相关性得分和最高得分（全文检索用）

- uri+请求体进行检索

GET bank/_search	
<pre>{ "query": { "match_all": {} }, "sort": [{ "account_number": { "order": "desc" } }] }</pre>	

HTTP 客户端工具（POSTMAN），get 请求不能携带请求体，我们变为 post 也是一样的
我们 POST 一个 JSON 风格的查询请求体到 _search API。

需要了解，一旦搜索的结果被返回，Elasticsearch 就完成了这次请求，并且不会维护任何服务端的资源或者结果的 cursor（游标）

2、Query DSL

1) 、基本语法格式

Elasticsearch 提供了一个可以执行查询的 **Json** 风格的 **DSL** (**domain-specific language** 领域特定语言)。这个被称为 **Query DSL**。该查询语言非常全面，并且刚开始的时候感觉有点复杂，真正学好它的方法是从一些基础的示例开始的。

- 一个查询语句 的典型结构

```
{  
    QUERY_NAME: {  
        ARGUMENT: VALUE,  
        ARGUMENT: VALUE,...  
    }  
}  
● 如果是针对某个字段，那么它的结构如下：  
{  
    QUERY_NAME: {  
        FIELD_NAME: {  
            ARGUMENT: VALUE,  
            ARGUMENT: VALUE,...  
        }  
    }  
}
```

```
GET bank/_search  
{  
    "query": {  
        "match_all": {}  
    },  
    "from": 0,  
    "size": 5,  
    "sort": [  
        {  
            "account_number": {  
                "order": "desc"  
            }  
        }  
    ]  
}
```

- `query` 定义如何查询，
- `match_all` 查询类型【代表查询所有的所有】，es 中可以在 `query` 中组合非常多的查询类型完成复杂查询
- 除了 `query` 参数之外，我们也可以传递其它的参数以改变查询结果。如 `sort`, `size`
- `from+size` 限定，完成分页功能
- `sort` 排序，多字段排序，会在前序字段相等时后续字段内部排序，否则以前序为准

2) 、返回部分字段

```
GET bank/_search  
{  
    "query": {
```

```
"match_all": {}  
},  
"from": 0,  
"size": 5,  
"_source": ["age","balance"]  
}
```

3) 、match 【匹配查询】

- 基本类型（非字符串），精确匹配

```
GET bank/_search  
{  
  "query": {  
    "match": {  
      "account_number": "20"  
    }  
  }  
}
```

match 返回 account_number=20 的

- 字符串，全文检索

```
GET bank/_search  
{  
  "query": {  
    "match": {  
      "address": "mill"  
    }  
  }  
}
```

最终查询出 address 中包含 mill 单词的所有记录

match 当搜索字符串类型的时候，会进行全文检索，并且每条记录有相关性得分。

- 字符串，多个单词（分词+全文检索）

```
GET bank/_search  
{  
  "query": {  
    "match": {  
      "address": "mill road"  
    }  
  }  
}
```

最终查询出 address 中包含 mill 或者 road 或者 mill road 的所有记录，并给出相关性得分

4) 、match_phrase 【短语匹配】

将需要匹配的值当成一个整体单词（不分词）进行检索

```
GET bank/_search
{
  "query": {
    "match_phrase": {
      "address": "mill road"
    }
  }
}
```

查出 address 中包含 mill road 的所有记录，并给出相关性得分

5) 、multi_match 【多字段匹配】

```
GET bank/_search
{
  "query": {
    "multi_match": {
      "query": "mill",
      "fields": ["state", "address"]
    }
  }
}
```

state 或者 address 包含 mill

6) 、bool 【复合查询】

bool 用来做复合查询：

复合语句可以合并 任何 其它查询语句，包括复合语句，了解这一点是很重要的。这就意味着，复合语句之间可以互相嵌套，可以表达非常复杂的逻辑。

- **must:** 必须达到 must 列举的所有条件

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "name": "Bank of America" } }
      ]
    }
  }
}
```

```
        { "match": { "gender": "M" } }
    ]
}
}
}
```

- **should:** 应该达到 **should** 列举的条件，如果达到会增加相关文档的评分，并不会改变查询的结果。如果 query 中只有 should 且只有一种匹配规则，那么 should 的条件就会被作为默认匹配条件而去改变查询结果

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "gender": "M" } }
      ],
      "should": [
        {"match": { "address": "lane" }}
      ]
    }
  }
}
```

- **must_not** 必须不是指定的情况

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "gender": "M" } }
      ],
      "should": [
        {"match": { "address": "lane" }}
      ],
      "must_not": [
        {"match": { "email": "baluba.com" }}
      ]
    }
  }
}
```

}

address 包含 mill，并且 gender 是 M，如果 address 里面有 lane 最好不过，但是 email 必须不包含 baluba.com

事件	描述
must	子句（查询）必须出现在匹配的文档中，并将有助于得分。
filter	子句（查询）必须出现在匹配的文档中。然而不像 must 此查询的分数将被忽略。
should	子句（查询）应出现在匹配文档中。在布尔查询中不包含 must 或 filter 子句，一个或多个 should 子句必须有相匹配的文件。匹配 should 条件的最小数目可通过设置 minimum_should_match 参数。
must_not	子句（查询）不能出现在匹配的文档中。

7)、filter【结果过滤】

并不是所有的查询都需要产生分数，特别是那些仅用于“**filtering**”（过滤）的文档。为了不计算分数 **Elasticsearch** 会自动检查场景并且优化查询的执行。

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": { "address": "mill"}}
      ],
      "filter": {
        "range": {
          "balance": {
            "gte": 10000,
            "lte": 20000
          }
        }
      }
    }
  }
}
```

8)、term

和 match 一样。匹配某个属性的值。全文检索字段用 **match**，其他非 **text** 字段匹配用 **term**。

Avoid using the **term** query for **text** fields.

By default, Elasticsearch changes the values of **text** fields as part of **analysis**. This can make finding exact matches for **text** field values difficult.

To search **text** field values, use the **match** query instead.

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {"term": {
          "age": {
            "value": "28"
          }
        }},
        {"match": {
          "address": "990 Mill Road"
        }}
      ]
    }
  }
}
```

9)、aggregations（执行聚合）

聚合提供了从数据中分组和提取数据的能力。最简单的聚合方法大致等于 **SQL GROUP BY** 和 **SQL 聚合函数**。在 **Elasticsearch** 中，您有执行搜索返回 **hits**（命中结果），并且同时返回聚合结果，把一个响应中的所有 **hits**（命中结果）分隔开的能力。这是非常强大且有效的，您可以执行查询和多个聚合，并且在一次使用中得到各自的（任何一个的）返回结果，使用一次简洁和简化的 **API** 来避免网络往返。

- 搜索 **address** 中包含 **mill** 的所有人的年龄分布以及平均年龄，但不显示这些人的详情。

```
GET bank/_search
{
  "query": {
    "match": {
      "address": "mill"
    }
  },
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "age"
      }
    },
    "avg_age": {
      "avg": {
        "field": "age"
      }
    }
  }
}
```

```
        "field": "age"
    }
}
},
"size": 0
}

size: 0 不显示搜索数据
aggs: 执行聚合。聚合语法如下
"aggs": {
    "aggs_name 这次聚合的名字，方便展示在结果集中": {
        "AGG_TYPE 聚合的类型 (avg,term,terms)": {}
    }
},
}
```

复杂：

按照年龄聚合，并且请求这些年段的这些人的平均薪资

```
GET bank/account/_search
{
    "query": {
        "match_all": {}
    },
    "aggs": {
        "age_avg": {
            "terms": {
                "field": "age",
                "size": 1000
            },
            "aggs": {
                "banlances_avg": {
                    "avg": {
                        "field": "balance"
                    }
                }
            }
        }
    },
    "size": 1000
}
```

复杂：查出所有年龄分布，并且这些年段中 M 的平均薪资和 F 的平均薪资以及这个年龄段的总体平均薪资

```
GET bank/account/_search
```

```
{  
    "query": {  
        "match_all": {}  
    },  
    "aggs": {  
        "age_agg": {  
            "terms": {  
                "field": "age",  
                "size": 100  
            },  
            "aggs": {  
                "gender_agg": {  
                    "terms": {  
                        "field": "gender.keyword",  
                        "size": 100  
                    },  
                    "aggs": {  
                        "balance_avg": {  
                            "avg": {  
                                "field": "balance"  
                            }  
                        }  
                    }  
                },  
                "balance_avg": {  
                    "avg": {  
                        "field": "balance"  
                    }  
                }  
            }  
        },  
        "size": 1000  
    }  
}
```

3、Mapping

1) 、字段类型

<p>核心类型</p> <p>字符串 (string) <i>text, keyword</i></p> <p>数字类型 (Numeric) <i>long, integer, short, byte, double, float, half_float, scaled_float</i></p> <p>日期类型 (Date) <i>date</i></p> <p>布尔类型 (Boolean) <i>boolean</i></p> <p>二进制类型 (binary) <i>binary</i></p>	<p>复合类型</p> <p>数组类型 (Array) <i>Array</i> 支持不针对特定的类型</p> <p>对象类型 (Object) <i>object</i> 用于单JSON对象</p> <p>嵌套类型 (Nested) <i>nested</i> 用于JSON对象数组</p> <p>地理类型 (Geo) 地理坐标 (Geo-points) <i>geo_point</i> 用于描述 经纬度坐标 地理图形 (Geo-Shape) <i>geo_shape</i> 用于描述复杂形状, 如多边形</p> 
<p>特定类型</p> <p>IP 类型 <i>ip</i> 用于描述 ipv4 和 ipv6 地址</p> <p>补全类型 (Completion) <i>completion</i> 提供自动完成提示</p> <p>令牌计数类型 (Token count) <i>token_count</i> 用于统计字符串中的词条数量</p> <p>附件类型 (attachment) 参考 <i>mapper-attachments</i> 插件, 支持将附件如Microsoft Office格式, Open Document格式, ePub, HTML等等索引为 <i>attachment</i> 数据类型。</p> <p>抽取类型 (Percolator) 接受特定领域查询语言 (query-dsl) 的查询</p> 	
<p>多字段</p> <p>通常用于为不同目的用不同的方法索引同一个字段。例如, <i>string</i> 字段可以映射为一个 <i>text</i> 字段用于全文检索, 同样可以映射为一个 <i>keyword</i> 字段用于排序和聚合。另外, 你可以使用 <i>standard analyzer</i>, <i>english analyzer</i>, <i>french analyzer</i> 来索引一个 <i>text</i> 字段</p> <p>这就是 <i>multi-fields</i> 的目的。大多数的数据类型通过fields参数来支持 <i>multi-fields</i>。</p> 	

2) 、映射

Mapping (映射)

Mapping 是用来定义一个文档 (**document**) , 以及它所包含的属性 (**field**) 是如何存储和索引的。比如, 使用 **mapping** 来定义:

- 哪些字符串属性应该被看做全文本属性 (**full text fields**) 。
- 哪些属性包含数字, 日期或者地理位置。
- 文档中的所有属性是否都能被索引 (**_all** 配置) 。
- 日期的格式。
- 自定义映射规则来执行动态添加属性。

- 查看 mapping 信息:

```
GET bank/_mapping
```

- 修改 mapping 信息

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html>

自动猜测的映射类型

JSON type	域 type
布尔型: true 或者 false	boolean
整数: 123	long
浮点数: 123.45	double
字符串, 有效日期: 2014-09-15	date
字符串: foo bar	string

3) 、新版本改变

Es7 及以上移除了 type 的概念。

- 关系型数据库中两个数据表示是独立的，即使他们里面有相同名称的列也不影响使用，但 ES 中不是这样的。elasticsearch 是基于 Lucene 开发的搜索引擎，而 ES 中不同 type 下名称相同的 filed 最终在 Lucene 中的处理方式是一样的。
 - 两个不同 type 下的两个 user_name，在 ES 同一个索引下其实被认为是同一个 filed，你必须在两个不同的 type 中定义相同的 filed 映射。否则，不同 type 中的相同字段名称就会在处理中出现冲突的情况，导致 Lucene 处理效率下降。
 - 去掉 type 就是为了提高 ES 处理数据的效率。

Elasticsearch 7.x

- URL 中的 type 参数为可选。比如，索引一个文档不再要求提供文档类型。

Elasticsearch 8.x

- 不再支持 URL 中的 type 参数。

解决：

- 1) 、将索引从多类型迁移到单类型，每种类型文档一个独立索引
- 2) 、将已存在的索引下的类型数据，全部迁移到指定位置即可。详见数据迁移

1、创建映射

1、创建索引并指定映射

```
PUT /my-index
```

```
{
```

```
  "mappings": {
```

```
    "properties": {
```

```
"age": { "type": "integer" },
"email": { "type": "keyword" },
"name": { "type": "text" }
}
}
}
```

2、添加新的字段映射

```
PUT /my-index/_mapping
{
  "properties": {
    "employee-id": {
      "type": "keyword",
      "index": false
    }
  }
}
```

3、更新映射

对于已经存在的映射字段，我们不能更新。更新必须创建新的索引进行数据迁移

4、数据迁移

先创建出 **new_twitter** 的正确映射。然后使用如下方式进行数据迁移

```
POST _reindex [固定写法]
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

将旧索引的 **type** 下的数据进行迁移

```
POST _reindex
{
  "source": {
```

```
"index": "twitter",
  "type": "tweet"
},
"dest": {
  "index": "tweets"
}
}
```

4、分词

一个 **tokenizer**（分词器）接收一个字符流，将之分割为独立的 **tokens**（词元，通常是独立的单词），然后输出 **tokens** 流。

例如，**whitespace tokenizer** 遇到空白字符时分割文本。它会将文本 "Quick brown fox!" 分割为 [Quick, brown, fox!]。

该 **tokenizer**（分词器）还负责记录各个 **term**（词条）的顺序或 **position** 位置（用于 **phrase** 短语和 **word proximity** 词近邻查询），以及 **term**（词条）所代表的原始 **word**（单词）的 **start**（起始）和 **end**（结束）的 **character offsets**（字符偏移量）（用于高亮显示搜索的内容）。**Elasticsearch** 提供了很多内置的分词器，可以用来构建 **custom analyzers**（自定义分词器）。

1) 安装 ik 分词器

注意：不能用默认 `elasticsearch-plugin install xxx.zip` 进行自动安装

<https://github.com/medcl/elasticsearch-analysis-ik/releases?after=v6.4.2> 对应 es 版本安装

进入 es 容器内部 `plugins` 目录

`docker exec -it 容器 id /bin/bash`

`wget`

<https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v7.4.2/elasticsearch-analysis-ik-7.4.2.zip>

`unzip` 下载的文件

`rm -rf *.zip`

`mv elasticsearch/ ik`

可以确认是否安装好了分词器

`cd .. /bin`

`elasticsearch plugin list:` 即可列出系统的分词器

2) 测试分词器

使用默认

```
POST _analyze
{
    "text": "我是中国人"
}
```

请观察结果

使用分词器

```
POST _analyze
{ "analyzer": "ik_smart",
  "text": "我是中国人"
}
```

请观察结果

另外一个分词器

ik_max_word

```
POST _analyze
{ "analyzer": "ik_max_word",
  "text": "我是中国人"
}
```

请观察结果

能够看出不同的分词器，分词有明显的区别，所以以后定义一个索引不能再使用默认的 mapping 了，要手工建立 mapping，因为要选择分词器。

3) 自定义词库

修改/usr/share/elasticsearch/plugins/ik/config/中的 IKAnalyzer.cfg.xml
/usr/share/elasticsearch/plugins/ik/config

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict"></entry>
    <!--用户可以在这里配置自己的扩展停止词字典-->
    <entry key="ext_stopwords"></entry>
    <!--用户可以在这里配置远程扩展字典 -->
    <entry key="remote_ext_dict">http://192.168.128.130/fenci/myword.txt</entry>
    <!--用户可以在这里配置远程扩展停止词字典-->
    <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

原来的 xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
```

```

<!--用户可以在这里配置自己的扩展字典 -->
<entry key="ext_dict"></entry>
<!--用户可以在这里配置自己的扩展停止词字典-->
<entry key="ext_stopwords"></entry>
<!--用户可以在这里配置远程扩展字典 -->
<!-- <entry key="remote_ext_dict">words_location</entry> -->
<!--用户可以在这里配置远程扩展停止词字典-->
<!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>

```

按照标红的路径利用 nginx 发布静态资源,按照请求路径, 创建对应的文件夹以及文件, 放在 nginx 的 html 下



然后重启 es 服务器, 重启 nginx。

在 kibana 中测试分词效果

A screenshot of the Kibana interface showing a search request and its results.

Request:

```
GET movie_index/_analyze
{
  "analyzer": "ik_max_word",
  "text": "尚硅谷技术真牛"
}
```

Response:

```

1  [
2   "tokens": [
3     {
4       "token": "尚硅谷技术",
5       "start_offset": 0,
6       "end_offset": 5,
7       "type": "CN_WORD",
8       "position": 0
9     },
10    {
11      "token": "尚硅谷",
12      "start_offset": 0,
13      "end_offset": 3,
14      "type": "CN_WORD",
15      "position": 1
16    },
17    {
18      "token": "硅谷",
19      "start_offset": 1,
20      "end_offset": 3,
21      "type": "CN_WORD",
22      "position": 2
23    }
]

```

更新完成后, es 只会对新增的数据用新词分词。历史数据是不会重新分词的。如果想要历史数据重新分词。需要执行:

```
POST my_index/_update_by_query?conflicts=proceed
```

五、Elasticsearch-Rest-Client

- 1) 、9300: TCP
- spring-data-elasticsearch:transport-api.jar;
 - springboot 版本不同, **transport-api.jar** 不同, 不能适配 es 版本

- 7.x 已经不建议使用，8以后就要废弃
- 2)、9200: HTTP
 - JestClient: 非官方, 更新慢
 - RestTemplate: 模拟发 HTTP 请求, ES很多操作需要自己封装, 麻烦
 - HttpClient: 同上
 - Elasticsearch-Rest-Client: 官方 RestClient, 封装了 ES 操作, API 层次分明, 上手简单

最终选择 Elasticsearch-Rest-Client (elasticsearch-rest-high-level-client)

<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high.html>

1、SpringBoot 整合

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>7.4.2</version>
</dependency>
```

2、配置

```
@Bean
RestHighLevelClient client() {
    RestClientBuilder builder = RestClient.builder(new HttpHost("192.168.56.10", 9200,
    "http"));
    return new RestHighLevelClient(builder);
}
```

3、使用

参照官方文档:

```
@Test
void test1() throws IOException {
    Product product = new Product();
    product.setSpuName("华为");
    product.setId(10L);

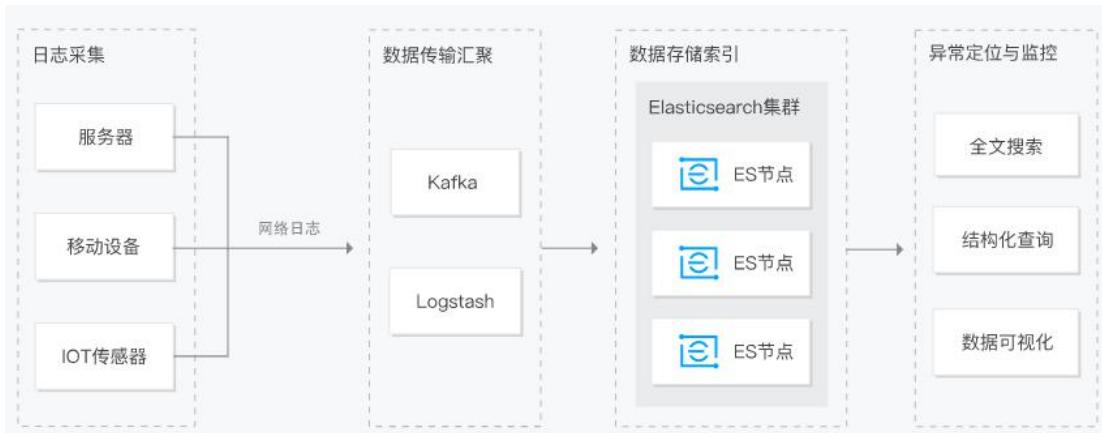
    IndexRequest request = new IndexRequest("product").id("20")
        .source("spuName", "华为", "id", 20L);
    try {
        IndexResponse response = client.index(request, RequestOptions.DEFAULT);
        System.out.println(response.toString());
    }
}
```

```

        IndexResponse response2 = client.index(request, RequestOptions.DEFAULT);
    } catch (ElasticsearchException e) {
        if (e.status() == RestStatus.CONFLICT) {

        }
    }
}

```



六、附录-安装 nginx

- 随便启动一个 nginx 实例，只是为了复制出配置
 - docker run -p 80:80 --name nginx -d nginx:1.10
- 将容器内的配置文件拷贝到当前目录: docker container cp nginx:/etc/nginx .
 - 别忘了后面的点
- 修改文件名称: mv nginx.conf 把这个 conf 移动到/mydata/nginx 下
- 终止原容器: docker stop nginx
- 执行命令删除原容器: docker rm \$ContainerId
- 创建新的 nginx; 执行以下命令

```

docker run -p 80:80 --name nginx \
-v /mydata/nginx/html:/usr/share/nginx/html \
-v /mydata/nginx/logs:/var/log/nginx \
-v /mydata/nginx/conf:/etc/nginx \
-d nginx:1.10

```

```

[root@10 nginx]# pwd
/mydata/nginx
[root@10 nginx]# ls
conf  html  logs

```

- 给 nginx 的 html 下面放的所有资源可以直接访问;

Gulimall

异步&线程池



一、线程回顾

1、初始化线程的 4 种方式

- 1)、继承 Thread
- 2)、实现 Runnable 接口
- 3)、实现 Callable 接口 + FutureTask (可以拿到返回结果，可以处理异常)
- 4)、线程池

方式 1 和方式 2：主进程无法获取线程的运算结果。不适合当前场景

方式 3：主进程可以获取线程的运算结果，但是不利于控制服务器中的线程资源。可以导致服务器资源耗尽。

方式 4：通过如下两种方式初始化线程池

```
Executors.newFixedThreadPool(3);  
//或者  
new ThreadPoolExecutor(corePoolSize, maximumPoolSize, keepAliveTime, TimeUnit unit,  
workQueue, threadFactory, handler);
```

通过线程池性能稳定，也可以获取执行结果，并捕获异常。但是，在业务复杂情况下，一个异步调用可能会依赖于另一个异步调用的执行结果。

2、线程池的七大参数

* `corePoolSize` the number of threads to keep in the pool, even if they are idle, unless `allowCoreThreadTimeOut` is set

池中一直保持的线程的数量，即使线程空闲。除非设置了 `allowCoreThreadTimeOut`

* `maximumPoolSize` the maximum number of threads to allow in the pool

池中允许的最大的线程数

* `keepAliveTime` when the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating.

当线程数大于核心线程数的时候，线程在最大多长时间没有接到新任务就会终止释放，最终线程池维持在 `corePoolSize` 大小

* `unit` the time unit for the `keepAliveTime` argument
时间单位

* `workQueue` the queue to use for holding tasks before they are

```
*      executed. This queue will hold only the {@code Runnable}
*      tasks submitted by the {@code execute} method.
阻塞队列，用来存储等待执行的任务，如果当前对线程的需求超过了 corePoolSize 大小，就会放在这里等待空闲线程执行。
```

```
* @param threadFactory the factory to use when the executor
*           creates a new thread
创建线程的工厂，比如指定线程名等
* @param handler the handler to use when execution is blocked
*           because the thread bounds and queue capacities are reached
拒绝策略，如果线程满了，线程池就会使用拒绝策略。
```

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler)
```

运行流程：

- 1、线程池创建，准备好 core 数量的核心线程，准备接受任务
- 2、新的任务进来，用 core 准备好的空闲线程执行。
 - (1) 、core 满了，就将再进来的任务放入阻塞队列中。空闲的 core 就会自己去阻塞队列获取任务执行
 - (2) 、阻塞队列满了，就直接开新线程执行，最大只能开到 max 指定的数量
 - (3) 、max 都执行好了。Max-core 数量空闲的线程会在 keepAliveTime 指定的时间后自动销毁。最终保持到 core 大小
 - (4) 、如果线程数开到了 max 的数量，还有新任务进来，就会使用 reject 指定的拒绝策略进行处理
- 3、所有的线程创建都是由指定的 factory 创建的。

面试：

一个线程池 core 7; max 20 , queue: 50, 100 并发进来怎么分配的；
先有 7 个能直接得到执行，接下来 50 个进入队列排队，在多开 13 个继续执行。现在 70 个被安排上了。剩下 30 个默认拒绝策略。

3、常见的 4 种线程池

- newCachedThreadPool
 - 创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。
- newFixedThreadPool
 - 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

- newScheduledThreadPool
 - 创建一个定长线程池，支持定时及周期性任务执行。
- newSingleThreadExecutor
 - 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

4、开发中为什么使用线程池

- 降低资源的消耗
 - 通过重复利用已经创建好的线程降低线程的创建和销毁带来的损耗
- 提高响应速度
 - 因为线程池中的线程数没有超过线程池的最大上限时，有的线程处于等待分配任务的状态，当任务来时无需创建新的线程就能执行
- 提高线程的可管理性
 - 线程池会根据当前系统特点对池内的线程进行优化处理，减少创建和销毁线程带来的系统开销。无限的创建和销毁线程不仅消耗系统资源，还降低系统的稳定性，使用线程池进行统一分配

二、CompletableFuture 异步编排

业务场景：

查询商品详情页的逻辑比较复杂，有些数据还需要远程调用，必然需要花费更多的时间。

```
// 1. 获取sku的基本信息    0.5s  
// 2. 获取sku的图片信息    0.5s  
// 3. 获取sku的促销信息    1s  
// 4. 获取spu的所有销售属性  1s  
// 5. 获取规格参数组及组下的规格参数  1.5s  
// 6. spu详情      1s
```

假如商品详情页的每个查询，需要如下标注的时间才能完成

那么，用户需要 5.5s 后才能看到商品详情页的内容。很显然是不能接受的。

如果有多个线程同时完成这 6 步操作，也许只需要 1.5s 即可完成响应。

Future 是 Java 5 添加的类，用来描述一个异步计算的结果。你可以使用 `isDone` 方法检查计

算是否完成，或者使用`get`阻塞住调用线程，直到计算完成返回结果，你也可以使用`cancel`方法停止任务的执行。

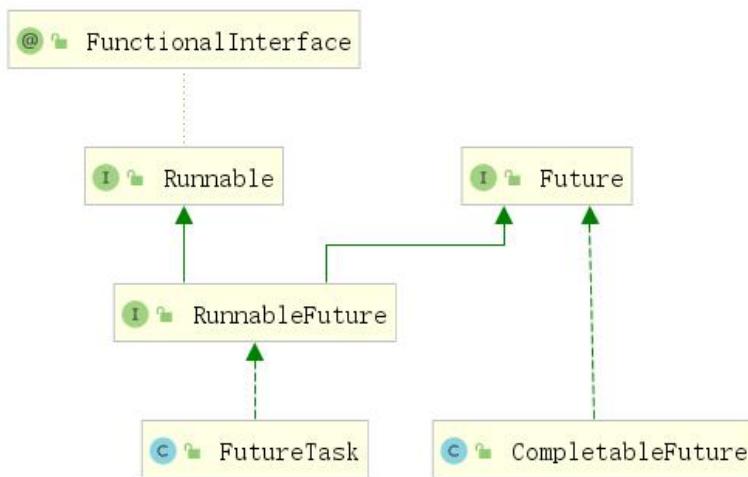
虽然`Future`以及相关使用方法提供了异步执行任务的能力，但是对于结果的获取却是很方便，只能通过阻塞或者轮询的方式得到任务的结果。阻塞的方式显然和我们的异步编程的初衷相违背，轮询的方式又会耗费无谓的CPU资源，而且也不能及时地得到计算结果，为什么不能用观察者设计模式当计算结果完成及时通知监听者呢？

很多语言，比如Node.js，采用回调的方式实现异步编程。Java的一些框架，比如Netty，自己扩展了Java的`Future`接口，提供了`addListener`等多个扩展方法；Google guava也提供了通用的扩展Future；Scala也提供了简单易用且功能强大的Future/Promise异步编程模式。

作为正统的Java类库，是不是应该做点什么，加强一下自身库的功能呢？

在Java 8中，新增加了一个包含50个方法左右的类：`CompletableFuture`，提供了非常强大的Future的扩展功能，可以帮助我们简化异步编程的复杂性，提供了函数式编程的能力，可以通过回调的方式处理计算结果，并且提供了转换和组合`CompletableFuture`的方法。
`CompletableFuture`类实现了`Future`接口，所以你还是可以像以前一样通过`get`方法阻塞或者轮询的方式获得结果，但是这种方式不推荐使用。

`CompletableFuture`和`FutureTask`同属于`Future`接口的实现类，都可以获取线程的执行结果。



1、创建异步对象

`CompletableFuture`提供了四个静态方法来创建一个异步操作。

```
static CompletableFuture<Void> runAsync(Runnable runnable)
public static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor)
public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier)
public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor
executor)
```

- 1、runXxx 都是没有返回结果的，supplyXxx 都是可以获取返回结果的
- 2、可以传入自定义的线程池，否则就用默认的线程池；

2、计算完成时回调方法

```
public CompletableFuture<T> whenComplete(BiConsumer<? super T, ? super Throwable> action);
public CompletableFuture<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable>
action);
public CompletableFuture<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable>
action, Executor executor);

public CompletableFuture<T> exceptionally(Function<Throwable, ? extends T> fn);
```

whenComplete 可以处理正常和异常的计算结果，exceptionally 处理异常情况。

whenComplete 和 whenCompleteAsync 的区别：

whenComplete：是执行当前任务的线程执行继续执行 whenComplete 的任务。

whenCompleteAsync：是执行把 whenCompleteAsync 这个任务继续提交给线程池来进行执行。

方法不以 Async 结尾，意味着 Action 使用相同的线程执行，而 Async 可能会使用其他线程执行（如果是使用相同的线程池，也可能被同一个线程选中执行）

```
public class CompletableFutureDemo {

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        CompletableFuture future = CompletableFuture.supplyAsync(new Supplier<Object>() {
            @Override
            public Object get() {
                System.out.println(Thread.currentThread().getName() + "\t"
completableFuture");
                int i = 10 / 0;
                return 1024;
            }
        }).whenComplete(new BiConsumer<Object, Throwable>() {
            @Override
            public void accept(Object o, Throwable throwable) {
                System.out.println("-----o=" + o.toString());
                System.out.println("-----throwable=" + throwable);
            }
        });
    }
}
```

```
}).exceptionally(new Function<Throwable, Object>() {
    @Override
    public Object apply(Throwable throwable) {
        System.out.println("throwable=" + throwable);
        return 6666;
    }
});
System.out.println(future.get());
}
```

3、handle 方法

```
public <U> CompletionStage<U> handle(BiFunction<? super T, Throwable, ? extends U> fn);
public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn);
public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn, Executor executor);
```

和 complete 一样，可对结果做最后的处理（可处理异常），可改变返回值。

4、线程串行化方法

```
public <U> CompletableFuture<U> thenApply(Function<? super T, ? extends U> fn)
public <U> CompletableFuture<U> thenApplyAsync(Function<? super T, ? extends U> fn)
public <U> CompletableFuture<U> thenApplyAsync(Function<? super T, ? extends U> fn, Executor executor)

public CompletionStage<Void> thenAccept(Consumer<? super T> action);
public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action);
public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action, Executor executor);

public CompletionStage<Void> thenRun(Runnable action);
public CompletionStage<Void> thenRunAsync(Runnable action);
public CompletionStage<Void> thenRunAsync(Runnable action, Executor executor);
```

thenApply 方法：当一个线程依赖另一个线程时，获取上一个任务返回的结果，并返回当前任务的返回值。

thenAccept 方法：消费处理结果。接收任务的处理结果，并消费处理，无返回结果。

thenRun 方法：只要上面的任务执行完成，就开始执行 thenRun，只是处理完任务后，执行 thenRun 的后续操作

带有 Async 默认是异步执行的。同之前。

以上都要前置任务成功完成。

Function<? super T, ? extends U>

T：上一个任务返回结果的类型

U: 当前任务的返回值类型

5、两任务组合 - 都要完成

```
public <U,V> CompletableFuture<V> thenCombine(
    CompletionStage<? extends U> other,
    BiFunction<? super T,> super U,> ? extends V> fn);

public <U,V> CompletableFuture<V> thenCombineAsync(
    CompletionStage<? extends U> other,
    BiFunction<? super T,> super U,> ? extends V> fn);

public <U,V> CompletableFuture<V> thenCombineAsync(
    CompletionStage<? extends U> other,
    BiFunction<? super T,> super U,> ? extends V> fn, Executor executor);

public <U> CompletableFuture<Void> thenAcceptBoth(
    CompletionStage<? extends U> other,
    BiConsumer<? super T, ? super U> action);

public <U> CompletableFuture<Void> thenAcceptBothAsync(
    CompletionStage<? extends U> other,
    BiConsumer<? super T, ? super U> action);

public <U> CompletableFuture<Void> thenAcceptBothAsync(
    CompletionStage<? extends U> other,
    BiConsumer<? super T, ? super U> action, Executor executor);

public CompletableFuture<Void> runAfterBoth(CompletionStage<?> other,
                                              Runnable action);

public CompletableFuture<Void> runAfterBothAsync(CompletionStage<?> other,
                                                 Runnable action);

public CompletableFuture<Void> runAfterBothAsync(CompletionStage<?> other,
                                                 Runnable action,
                                                 Executor executor);
```

两个任务必须都完成，触发该任务。

thenCombine: 组合两个 future，获取两个 future 的返回结果，并返回当前任务的返回值
thenAcceptBoth: 组合两个 future，获取两个 future 任务的返回结果，然后处理任务，没有返回值。

runAfterBoth: 组合两个 future，不需要获取 future 的结果，只需两个 future 处理完任务后，处理该任务。

6、两任务组合 - 一个完成

```
public <U> CompletableFuture<U> applyToEither(
    CompletionStage<? extends T> other, Function<? super T, U> fn);

public <U> CompletableFuture<U> applyToEitherAsync(
    CompletionStage<? extends T> other, Function<? super T, U> fn);

public <U> CompletableFuture<U> applyToEitherAsync(
    CompletionStage<? extends T> other, Function<? super T, U> fn,
    Executor executor);

public CompletableFuture<Void> acceptEither(
    CompletionStage<? extends T> other, Consumer<? super T> action);

public CompletableFuture<Void> acceptEitherAsync(
    CompletionStage<? extends T> other, Consumer<? super T> action);

public CompletableFuture<Void> acceptEitherAsync(
    CompletionStage<? extends T> other, Consumer<? super T> action,
    Executor executor);

public CompletableFuture<Void> runAfterEither(CompletionStage<?> other,
    Runnable action);

public CompletableFuture<Void> runAfterEitherAsync(CompletionStage<?> other,
    Runnable action);

public CompletableFuture<Void> runAfterEitherAsync(CompletionStage<?> other,
    Runnable action,
    Executor executor);
```

当两个任务中，任意一个 future 任务完成的时候，执行任务。

applyToEither: 两个任务有一个执行完成，获取它的返回值，处理任务并有新的返回值。
acceptEither: 两个任务有一个执行完成，获取它的返回值，处理任务，没有新的返回值。
runAfterEither: 两个任务有一个执行完成，不需要获取 future 的结果，处理任务，也没有返回值。

7、多任务组合

```
public static CompletableFuture<Void> allOf(CompletableFuture<?>... cfs);

public static CompletableFuture<Object> anyOf(CompletableFuture<?>... cfs);

allOf: 等待所有任务完成
anyOf: 只要有一个任务完成
```



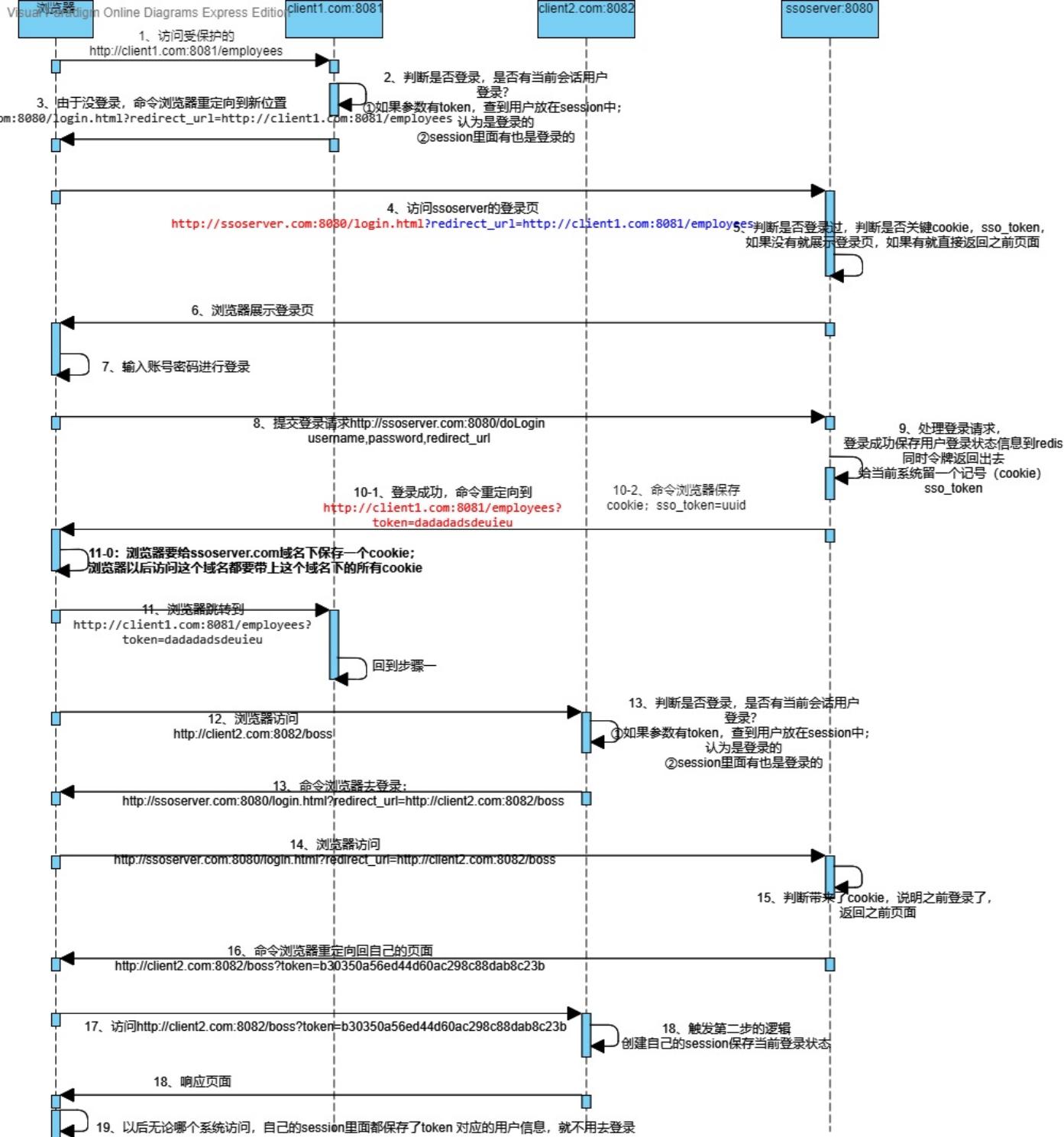
让天下没有难学的技术

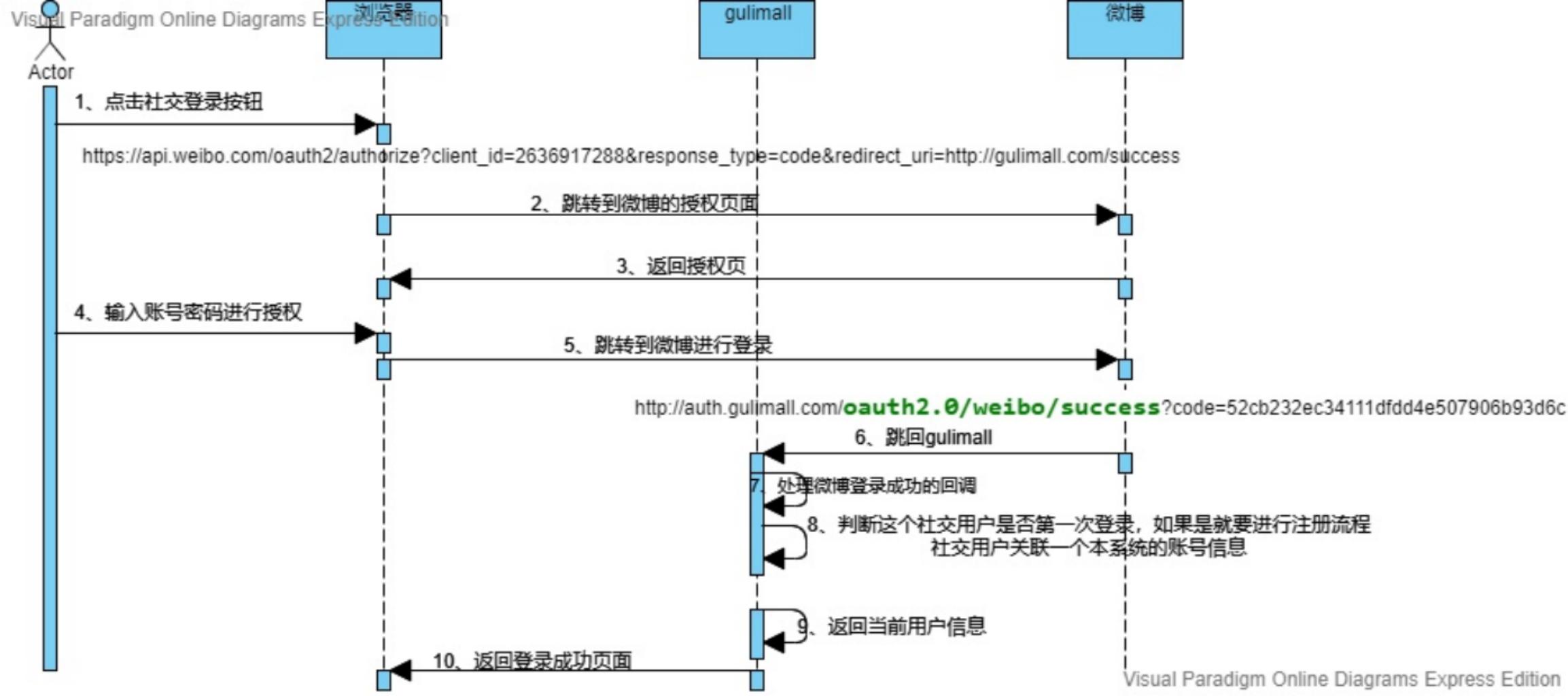


Gulimall

社交登陆&单点登陆







一、社交登陆



QQ、微博、github 等网站的用户量非常大，别的网站为了简化自我网站的登陆与注册逻辑，引入社交登陆功能；

步骤：

- 1)、用户点击 QQ 按钮
- 2)、引导跳转到 QQ 授权页

- 3)、用户主动点击授权，跳回之前网页。

1、OAuth2.0

- **OAuth:** OAuth（开放授权）是一个开放标准，允许用户授权第三方网站访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方网站或分享他们数据的所有内容。
- **OAuth2.0:** 对于用户相关的 OpenAPI（例如获取用户信息，动态同步，照片，日志，分享等），为了保护用户数据的安全和隐私，第三方网站访问用户数据前都需要显式的向用户征求授权。
- 官方版流程：

关于OAuth2.0协议的授权流程可以参考下面的流程图，其中Client指第三方应用，Resource Owner指用户，Authorization Server是我们的授权服务器，Resource Server是API服务器。



-
- (A) 用户打开客户端以后，客户端要求用户给予授权。
 - (B) 用户同意给予客户端授权。
 - (C) 客户端使用上一步获得的授权，向认证服务器申请令牌。
 - (D) 认证服务器对客户端进行认证以后，确认无误，同意发放令牌。
 - (E) 客户端使用令牌，向资源服务器申请获取资源。
 - (F) 资源服务器确认令牌无误，同意向客户端开放资源。

2、微博登陆准备工作

1、进入微博开放平台



微博开放平台-新浪大数据应用平台

优惠: 我们将为您提供试用账号 品牌: 新浪舆情通 特色: 专属多维度的舆情监测 详情: 舆情全网每日采集上亿+条数据,新浪微博授权全量数据,数据类型共涵盖全网大信息源.
www.yqt365.com 2019-03 - V3 - 评价 广告

新浪微博开放平台-首页

微博开放平台为移动应用提供了便捷的合作模式,满足了多元化移动终端用户随时随地快速登录、分享信息的需求,助力实现移动Apps、健康设备、智能家居、车载等多类型终端的社...
<https://open.weibo.com/> - 百度快照 - 90%好评

微博登录

微博登录介绍 微博登录包括身份认证、用
户关系以及内容传播。允许用户使

微博组件

新浪微博签名档可以放置在你的博客、论
坛,或是其它可以引用网上图片的位

2、登陆微博，进入微连接，选择网站接入



3、选择立即接入



4、创建自己的应用



创建新应用

应用名称: guli_shop 该名称也用于来源显示, 不超过10个汉字或20个字母

应用分类: 网页应用

我已阅读并接受 《微博开发者协议》

申请SAE应用托管服务

创建

5、我们可以在开发阶段进行测试了



应用状态

申请 > 开发 > 审核 > 上线 提交审核

尚未提交审核, 您还需要:
· 通过开发者身份认证审核
· 完善应用信息

应用基本信息

应用类型: 普通应用 - 网页应用

应用名称: guli_shop 该名称也用于来源显示, 不超过10个汉字或2个字母

记住自己的 app key 和 app secret 我们一会儿用

6、进入高级信息，填写授权回调页的地址



guli_shop

控制台

应用信息

基本信息

高级信息

测试信息

OAuth2.0 授权设置

授权回调页: http://www.gulishop.com/success

取消授权回调页: http://www.gulishop.com/cancel

当用户在我的应用管理页中取消对你的应用授权时，开放平台会回调该地址。查看回调参数

提交 取消

修改后立即生效

安全设置 编辑

7、添加测试账号（选做）



guli_shop

控制台

应用信息

基本信息

高级信息

测试信息

测试帐号 (0) 编辑

您的应用还未通过审核，不能大范围推广。您可以设置测试帐号来测试您尚在开发中的应用。

已关联测试帐号 (0/15 人)

您尚未添加任何测试帐号

8、进入文档，按照流程测试社交登陆



微博·开放平台

微连接 微服务 文档 支持 推广 我的应用

for_sun

更多可以帮助到你的文档

新手引导

在您接入网站和开发应用之前，需要了解如何成为平台的开发者，以及之后如何创建应用、提交审核、最终让微博用户在微博应用广场上找到您发布的应用。

API 文档

大部分API的访问如发表微博、获取私信、关注都需要用户身份。目前微博开放平台用户身份鉴权使用OAuth2.0协议，同时提供对Web、桌面和移动应用程序的支持。

SDK 下载

1、微博登录

基于OAuth2.0协议，使用微博 Open API 进行开发，即可用微博帐号登录你的网站，让你的网站降低新用户注册成本，快速获取大量用户。

[了解微博登录](#)

3、微博登陆测试

1、引导用户到如下地址

https://api.weibo.com/oauth2/authorize?client_id=YOUR_CLIENT_ID&response_type=code&redirect_uri=YOUR_REGISTERED_REDIRECT_URI



2、用户同意授权，页面跳转至 `xxx/?code=CODE`

<http://www.gulishop.com/success?code=fef987b3f9ad1169955840b467bfc661>

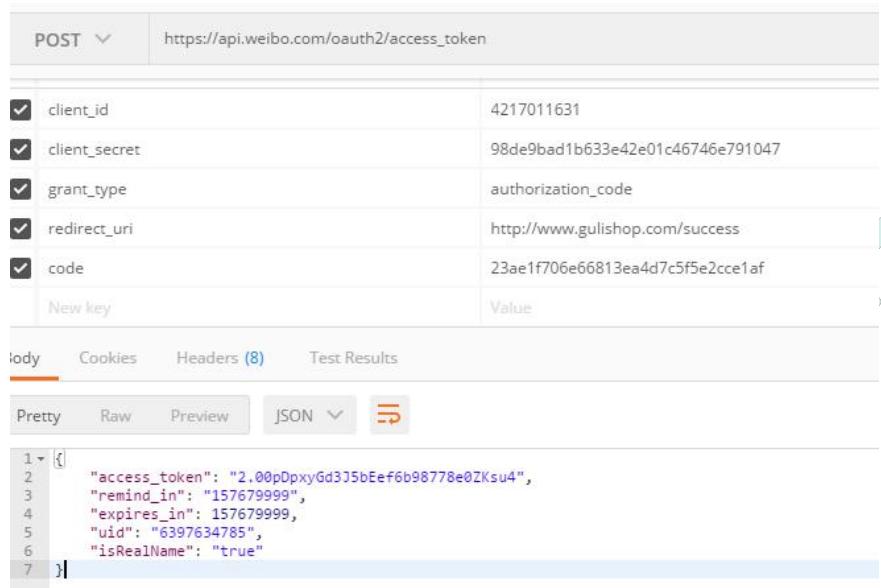
3、使用返回的 code，换取 access token

https://api.weibo.com/oauth2/access_token?client_id=YOUR_CLIENT_ID&client_secret=YOUR_CLIENT_SECRET&grant_type=authorization_code&redirect_uri=YOUR_REGISTERED_REDIRECT_URI&code=CODE

https://api.weibo.com/oauth2/access_token?client_id=4217011631&client_secret=98de9bad1b633e42e01c46746e791047&grant_type=authorization_code&redirect_uri=http://www.gulishop.com/success&code=fef987b3f9ad1169955840b467bfc661

注意，上面这个是 post 请求

```
{  
    "access_token": "2.00pDpxyGd3J5bEef6b98778e0ZKsu4",  
    "remind_in": "157679999",  
    "expires_in": 157679999,  
    "uid": "6397634785",  
    "isRealName": "true"  
}
```



The screenshot shows a Postman interface with a POST request to https://api.weibo.com/oauth2/access_token. The request body contains the following parameters:

参数	值
client_id	4217011631
client_secret	98de9bad1b633e42e01c46746e791047
grant_type	authorization_code
redirect_uri	http://www.gulishop.com/success
code	23ae1f706e66813ea4d7c5f5e2cce1af

Below the body, there is a JSON preview:

```
1  {  
2      "access_token": "2.00pDpxyGd3J5bEef6b98778e0ZKsu4",  
3      "remind_in": "157679999",  
4      "expires_in": 157679999,  
5      "uid": "6397634785",  
6      "isRealName": "true"  
7  }
```

4、使用 AccessToken 调用开发 API 获取用户信息



The screenshot shows the Weibo Open Platform interface. On the left, there is a sidebar with the following navigation items:

- 控制台
- 应用信息 >
- 数据统计 >
- 接口管理 > **已有权限** (highlighted with a red box)
- 申请权限
- 授权机制
- 管理应用 >

The main content area displays a list of existing interfaces (18 groups):

- 微博普通读取接口
- 评论普通读取接口
- 评论普通写入接口
- 用户普通读取接口** (highlighted with a red box)
 - users/show
 - users/domain_show
 - users/counts
- 关系普通读取接口
- 账户普通读取接口

至此微博登陆调试完成。

Oauth2.0：授权通过后，使用 code 换取 access_token，然后去访问任何开放 API

- 1)、code 用后即毁
- 2)、access_token 在几天内是一样的
- 3)、uid 永久固定

二、SSO（单点登录）

Single Sign On 一处登陆、处处可用

0、前置概念：

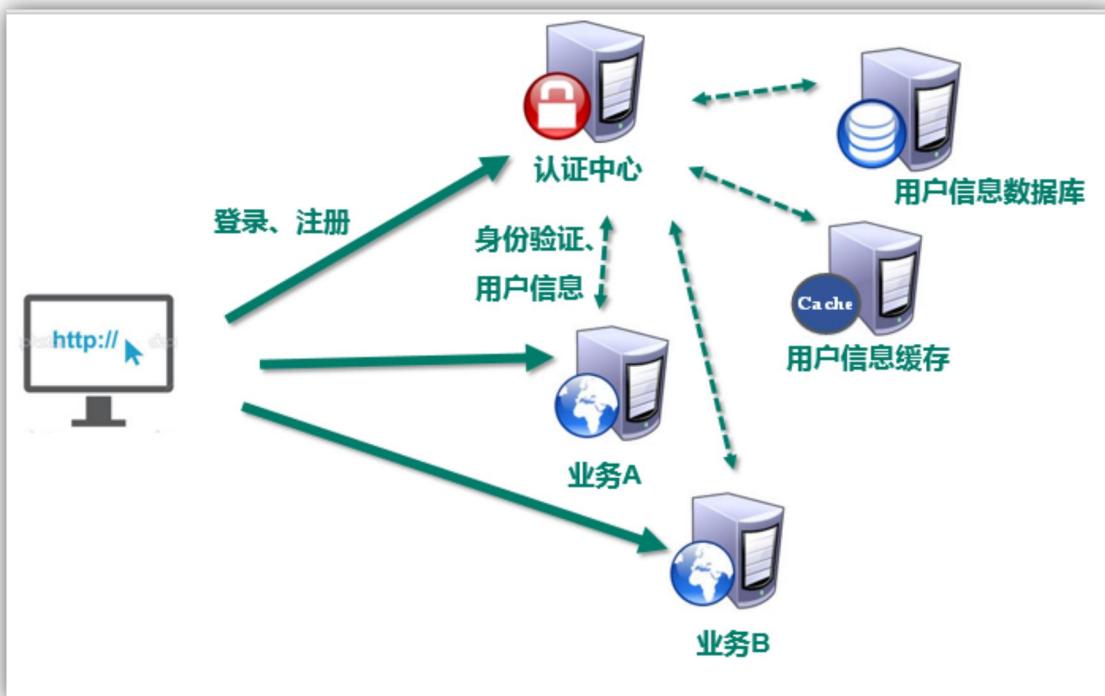
1)、单点登录业务介绍

早期单一服务器，用户认证。



缺点：单点性能压力，无法扩展

分布式，SSO(single sign on)模式



解决：

用户身份信息独立管理，更好的分布式管理。

可以自己扩展安全策略

跨域不是问题

缺点：

认证服务器访问压力较大。

2)、几个基本概念

2.1 什么是跨域 Web SSO。

域名通过“.”号切分后，从右往左看，不包含“.”的是顶级域名，包含一个“.”的一级域名，包含两个“.”的是二级域名，以此类推。

例如对网址 <http://www.cnblogs.com/baibaomen>，域名部分是 www.cnblogs.com。用“.”拆分后从右往左看：

cookie.setDomain(".cnblogs.com");//最多设置到本域的一级域名这里

cookie.setDomain(".baidu.com");//最多设置到本域的一级域名这里

“com”不包含“.”，是顶级域名；“cnblogs.com”包含一个“.”，是一级域名；

www.cnblogs.com 包含两个“.”，是二级域名。

blog.cnblogs.com

news.cnblogs.com

跨域 Web SSO 指的是针对 Web 站点，各级域名不同都能处理的单点登录方案。

2.2 浏览器读写 cookie 的安全性限制：一级或顶级域名不同的网站，无法读到彼此写的 cookie。

所以 baidu.com 无法读到 cnblogs.com 写的 cookie。

一级域名相同，只是二级或更高级域名不同的站点，可以通过设置 domain 参数共享 cookie 读写。这种场景可以选择不跨域的 SSO 方案。

域名相同，只是 https 和 http 协议不同的 URL，默认 cookie 可以共享。知道这一点对处理 SSO 服务中心要登出

2.3 http 协议是无状态协议。浏览器访问服务器时，要让服务器知道你是谁，只有两种方式：



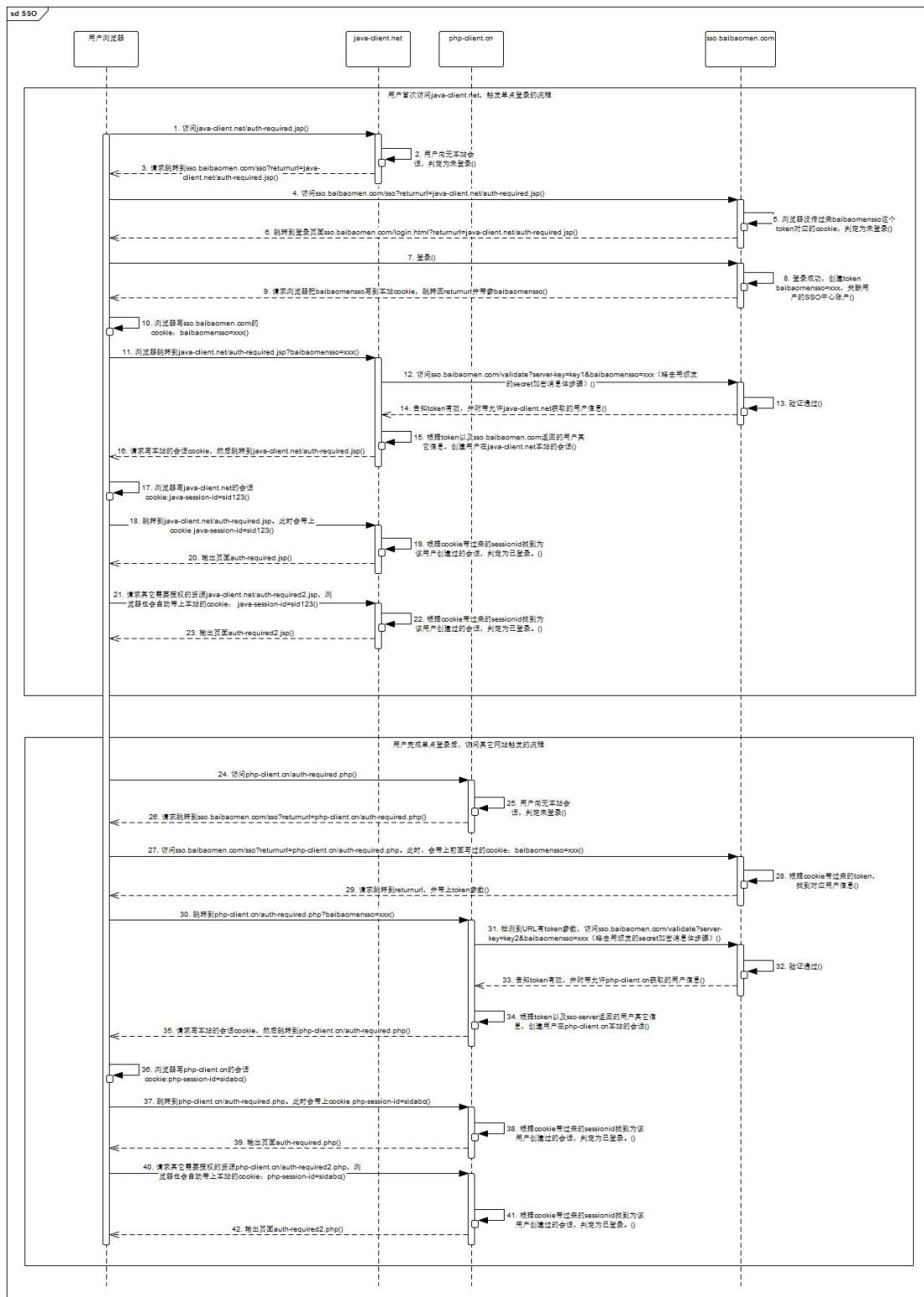
方式一：把“你是谁”写入 cookie。它会随每次 HTTP 请求带到服务端；

方式二：在 URL、表单数据中带上你的用户信息（也可能在 HTTP 头部）。这种方式依赖于从特定的网页入口进入，因为只有走特定的入口，才有机会拼装出相应的信息，提交到服务端。

大部分 SSO 需求都希望不依赖特定的网页入口（集成门户除外），所以后一种方式有局限性。适应性强的方式是第一种，即在浏览器通过 cookie 保存用户信息相关凭据，随每次请求传递到服务端。我们采用的方案是第一种。



1、Cookie 接入方式



2、Token 接入方式

类似社交登陆

3、有状态登录

为了保证客户端 cookie 的安全性，服务端需要记录每次会话的客户端信息，从而识别客户端身份，根据用户身份进行请求的处理，典型的设计如 tomcat 中的 session。

例如登录：用户登录后，我们把登录者的信息保存在服务端 session 中，并且给用户一个 cookie 值，记录对应的 session。然后下次请求，用户携带 cookie 值来，我们就能识别到对应 session，从而找到用户的信息。

缺点是什么？

- 服务端保存大量数据，增加服务端压力
- 服务端保存用户状态，无法进行水平扩展
- 客户端请求依赖服务端，多次请求必须访问同一台服务器

即使使用 redis 保存用户的信息，也会损耗服务器资源。

4、无状态登录

微服务集群中的每个服务，对外提供的都是 Rest 风格的接口。而 Rest 风格的一个最重要的规范就是：服务的无状态性，即：

- 服务端不保存任何客户端请求者信息
- 客户端的每次请求必须具备自描述信息，通过这些信息识别客户端身份

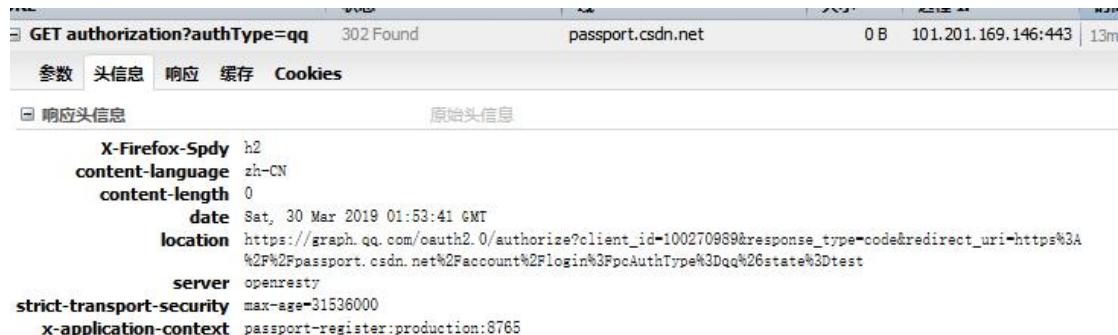
带来的好处是什么呢？

- 客户端请求不依赖服务端的信息，任何多次请求不需要必须访问到同一台服务
- 服务端的集群和状态对客户端透明
- 服务端可以任意的迁移和伸缩
- 减小服务端存储压力

5、集成社交登陆

1)、用户点击不同的社交登陆按钮，先来我们自己的服务 器

<https://passport.csdn.net/v1/register/authorization?authType=qq /sina>



2)、命令浏览器重定向到用户授权页

用户确认授权

<https://graph.qq.com/oauth2.0/authorize>



3)、qq 返回的响应，会命令用户重定向到指定位置

4)、服务器的这个位置就可以收到我们的 code 码

收到 code 码，服务器自己用 code 交换 access_token 令牌，并获取到用户的信息。给浏览器只给用户的信息即可；

access_token=UUID

浏览器访问带 UUID_token 而不是 access_token；

三、JWT

1、简介

JWT，全称是 Json Web Token，是 JSON 风格轻量级的授权和身份认证规范，可实现无状态、分布式的 Web 应用授权；官网：<https://jwt.io>

GitHub 上 jwt 的 java 客户端：<https://github.com/jwt/jjwt>

我们最终可以利用 jwt 实现无状态登录

2、数据格式

JWT 包含三部分数据：

- **Header:** 头部，通常头部有两部分信息：

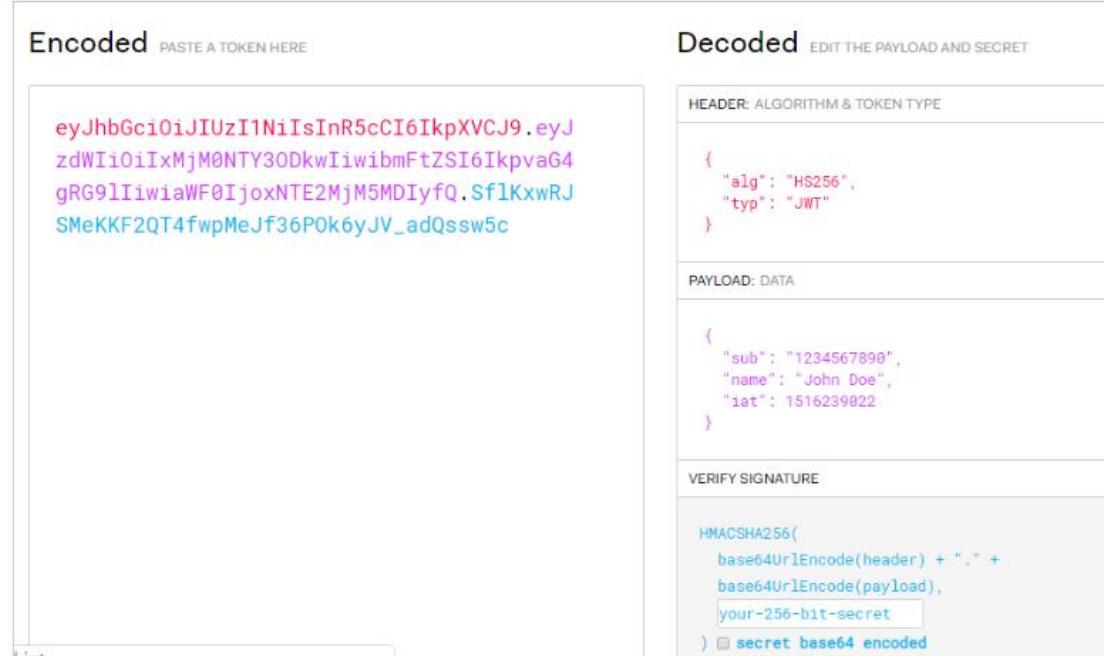
- token 类型：JWT
- 加密方式：base64 (HS256)

- **Payload:** 载荷，就是有效数据，一般包含下面信息：

- 用户身份信息（注意，这里因为采用 base64 编码，可解码，因此不要存放敏感信息）
- 注册声明：如 token 的签发时间，过期时间，签发人等

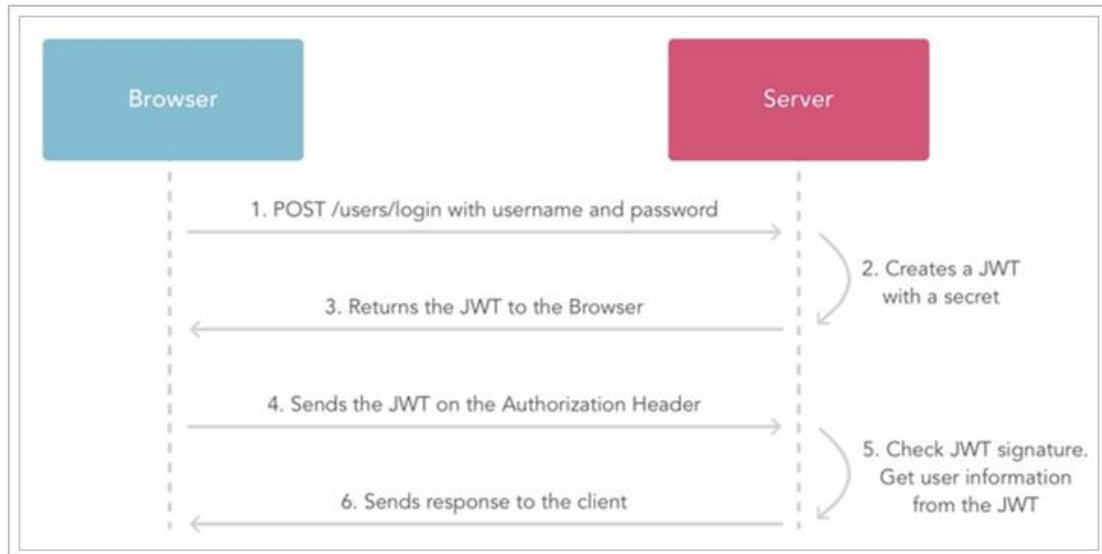
这部分也会采用 base64 编码，得到第二部分数据

- **Signature:** 签名，是整个数据的认证信息。根据前两步的数据，再加上指定的密钥 (secret)（不要泄漏，最好周期性更换），通过 base64 编码生成。用于验证整个数据完整和可靠性



The image shows a screenshot of a JWT Decoder tool. On the left, under 'Encoded' (PASTE A TOKEN HERE), there is a large input field containing a long base64-encoded string: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c. On the right, under 'Decoded' (EDIT THE PAYLOAD AND SECRET), the token is broken down into three parts: 1. HEADER: ALGORITHM & TOKEN TYPE: { "alg": "HS256", "typ": "JWT" } 2. PAYLOAD: DATA: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 } 3. VERIFY SIGNATURE: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) secret base64 encoded

3、交互流程

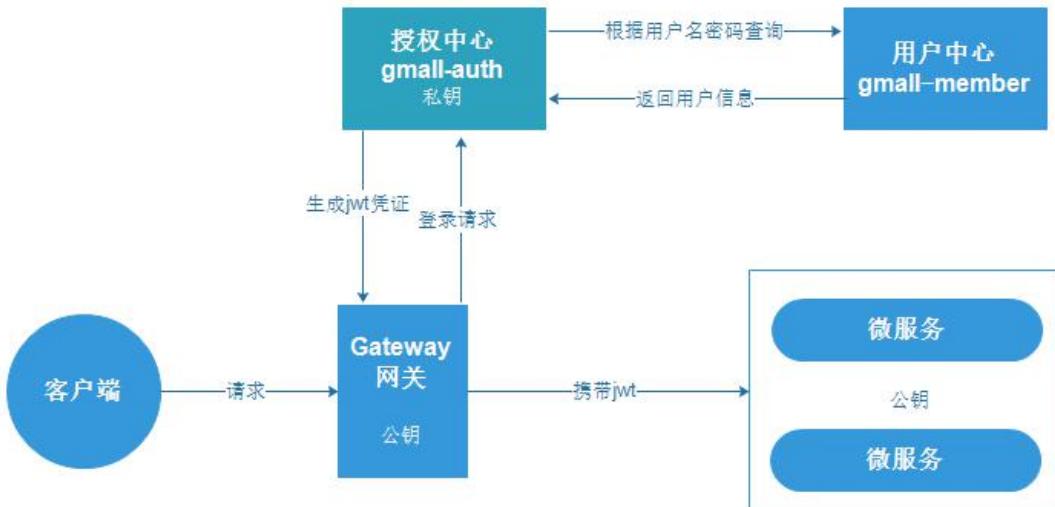


步骤：

- 1、用户登录
- 2、服务的认证，通过后根据 secret 生成 token
- 3、将生成的 token 返回给浏览器
- 4、用户每次请求携带 token
- 5、服务端利用秘钥解读 jwt 签名，判断签名有效后，从 Payload 中获取用户信息
- 6、处理请求，返回响应结果

因为 JWT 签发的 token 中已经包含了用户的身份信息，并且每次请求都会携带，这样服务的就无需保存用户信息，甚至无需去数据库查询，完全符合了 Rest 的无状态规范。

4、授权中心流程



5、JWT优势

- 易于水平扩展
 - 在 cookie-session 方案中，cookie 内仅包含一个 session 标识符，而诸如用户信息、授权列表等都保存在服务端的 session 中。如果把 session 中的认证信息都保存在 JWT 中，在服务端就没有 session 存在的必要了。当服务端水平扩展的时候，就不需要处理 session 复制（session replication）/ session 黏连（sticky session）或是引入外部 session 存储了[实际上 spring-session 和 hazelcast 能完美解决这个问题]。
- 防护 CSRF（跨站请求伪造）攻击
 - 访问某个网站会携带这个域名下的 cookie。所以可能导致攻击。但是我们可以把 jwt 放在请求头中发送。
 - Jwt 放在请求头中，就必须把 jwt 保存在 cookie 或者 localStorage 中。保存这里 js 就会读写，又会导致 xss 攻击。可以设置 cookie，`httpOnly=true` 来防止 xss
- 安全
 - 只是 base64 编码了，cookie+session 直接将数据保存在服务端，看都看不见，请问哪个更安全？
 -

6、使用 JWT 带来的问题

- 我们不建议使用 jwt+cookie 代替 session+cookie 机制，jwt 更适合 restful api
- jwt token 泄露了怎么办？
 - 这个问题可以不考虑，因为 session+cookie 同样泄露了 cookie 的 jsessionid 也会有这个问题
 - 我们可以遵循以下规范减少风险
 - ◆ 使用 https 加密应用
 - ◆ 返回 jwt 给客户端时设置 `httpOnly=true` 并且使用 cookie 而不是 localStorage 存储 jwt，防止 XSS 攻击和 CSRF 攻击
- secret 如果泄露会导致大面积风险
 - 定期更新
 - Secret 设计可以和用户关联起来，每个用户不一样。防止全用一个 secret
- 注销和修改密码
 - 传统的 session+cookie 方案用户点击注销，服务端清空 session 即可，因为状态保存在服务端。我们不害怕注销后的假登录
 - Jwt 会有问题。用户如果注销了或者修改密码了。恶意用户还使用之前非法盗取来的 token，可以在不重新登录的情况下继续使用
 - ◆ 可以按程度使用如下设计，减少一定的风险
 - 清空客户端的 cookie，这样用户访问时就不会携带 jwt，服务端就认为用户需要重新登录。这是一个典型的假注销，对于用户表现出退出的行为，实际上这个时候携带对应的 jwt 依旧可以访问系统。
 - 清空或修改服务端的用户对应的 secret，这样在用户注销后，jwt 本身不变，但是由于 secret 不存在或改变，则无法完成校验。这也是为什么将

`secret` 设计成和用户相关的原因

- 借助第三方存储，管理 `jwt` 的状态，可以以 `jwt` 为 `key`，去 `redis` 校验存在性。但这样，就把无状态的 `jwt` 硬生生变成了有状态了，违背了 `jwt` 的初衷。实际上这个方案和 `session` 都差不多了。
- 修改密码则略微有些不同，假设号被到了，修改密码（是用户密码，不是 `jwt` 的 `secret`）之后，盗号者在原 `jwt` 有效期之内依旧可以继续访问系统，所以仅仅清空 `cookie` 自然是不够的，这时，需要强制性的修改 `secret`

■ 续签问题

- ◆ 传统的 `cookie` 续签方案一般都是框架自带的，`session` 有效期 30 分钟，30 分钟内如果有访问，`session` 有效期被刷新至 30 分钟。而 `jwt` 本身的 `payload` 之中也有一个 `exp` 过期时间参数，来代表一个 `jwt` 的时效性，而 `jwt` 想延期这个 `exp` 就有点身不由己了，因为 `payload` 是参与签名的，一旦过期时间被修改，整个 `jwt` 串就变了，`jwt` 的特性天然不支持续签！
- ◆ 可如下解决，但都不是完美方案
 - 每次请求刷新 `jwt`: 简单暴力，性能低下，浪费资源。
 - 只要快要过期的时候刷新 `jwt`: `jwt` 最后的几分钟，换新一下。但是如果用户连续操作了 27 分钟，只有最后的 3 分钟没有操作，导致未刷新 `jwt`，就很难受。
 - 完善 `refreshToken`: 借鉴 `oauth2` 的设计，返回给客户端一个 `refreshToken`，允许客户端主动刷新 `jwt`。这样做，还不如用 `oauth2`
 - 使用 `redis` 记录独立的过期时间:`jwt` 作为 `key`，在 `redis` 中保存过期时间，每次使用在 `redis` 中续期，如果 `redis` 没有就认为过期。但是这样做，还不如用 `session+cookie`

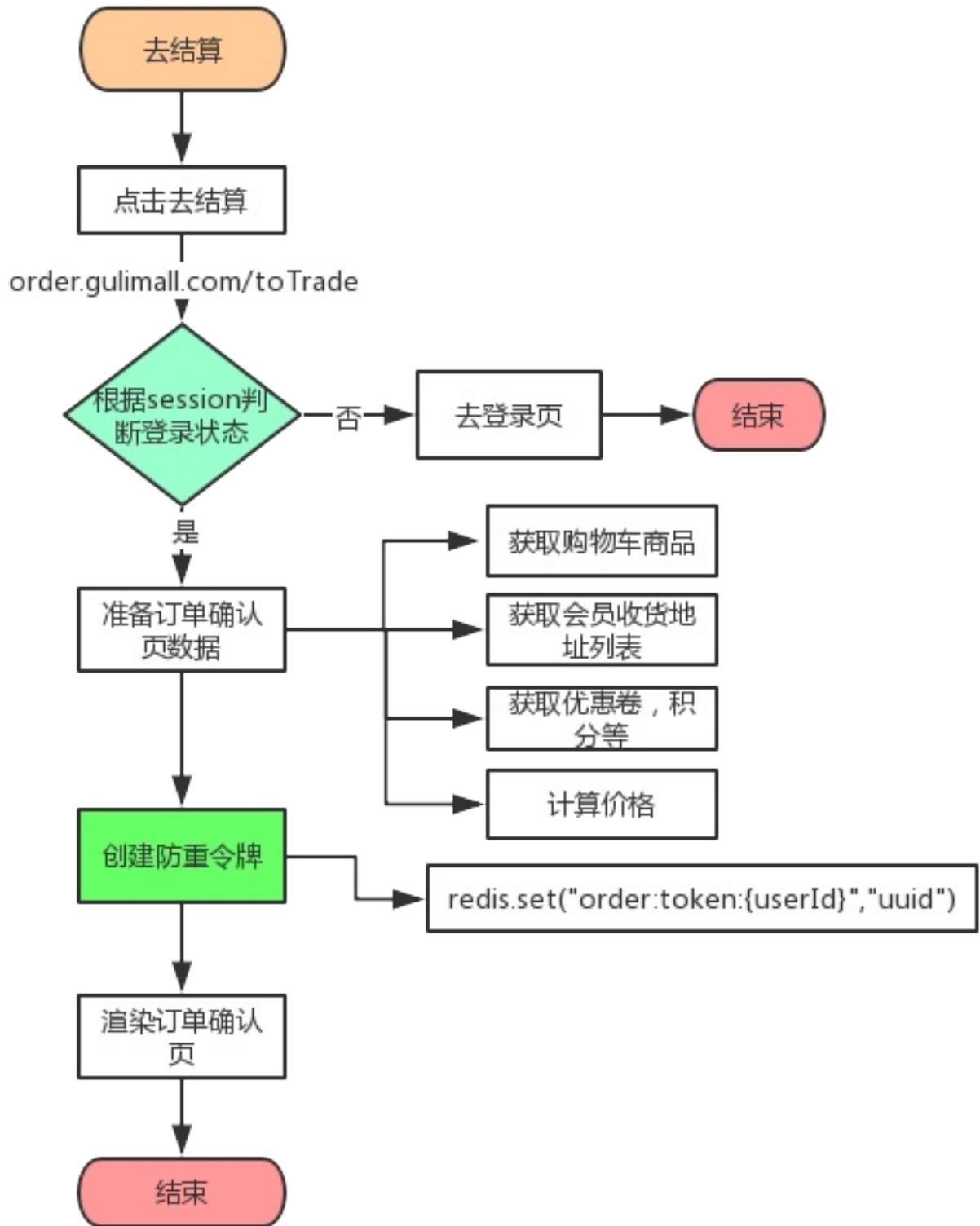
■ 总结

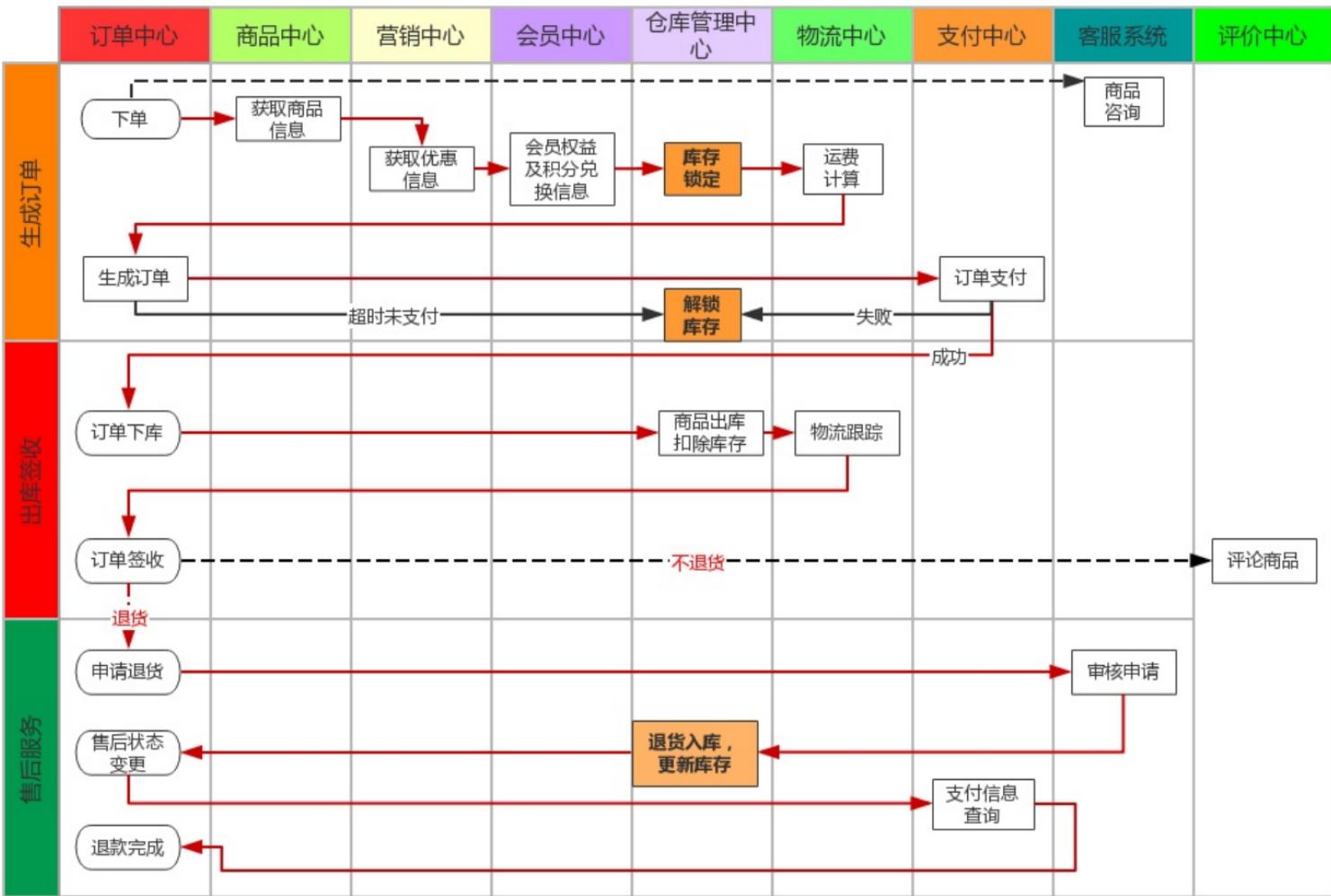
- ◆ 在 Web 应用中，别再把 JWT 当做 session 使用，绝大多数情况下，传统的 `cookie-session` 机制工作得更好
- ◆ JWT 适合一次性的命令认证，颁发一个有效期极短的 JWT，即使暴露了危险也很小，由于每次操作都会生成新的 JWT，因此也没必要保存 JWT，真正实现无状态。

Gulimall

商城业务







一、商品上架

上架的商品才可以在网站展示。

上架的商品需要可以被检索。

1、商品 Mapping

分析：商品上架在 es 中是存 sku 还是 spu？

- 1) 检索的时候输入名字，是需要按照 sku 的 title 进行全文检索的
- 2) 检索使用商品规格，规格是 spu 的公共属性，每个 spu 是一样的
- 3) 按照分类 id 进去的都是直接列出 spu 的，还可以切换。
- 4) 我们如果将 sku 的全量信息保存到 es 中（包括 spu 属性）就太多量字段了。
- 5) 我们如果将 spu 以及他包含的 sku 信息保存到 es 中，也可以方便检索。但是 sku 属于 spu 的级联对象，在 es 中需要 nested 模型，这种性能差点。
- 6) 但是存储与检索我们必须性能折中。
- 7) 如果我们分拆存储，spu 和 attr 一个索引，sku 单独一个索引可能涉及的问题。

检索商品的名字，如“手机”，对应的 spu 有很多，我们要分析出这些 spu 的所有关联属性，再做一次查询，就必须将所有 spu_id 都发出去。假设 1 万个数据，数据传输一次就 $10000 \times 4 = 4\text{MB}$ ；并发情况下假设 1000 检索请求，那就是 4GB 的数据，，传输阻塞时间会很长，业务更加无法继续。

所以，我们如下设计，这样才是文档区别于关系型数据库的地方，宽表设计，不能去考虑数据库范式。

1) PUT product

```
product 的 mapping
{
  "mappings": {
    "properties": {
      "skuid": {
        "type": "long"
      },
      "spuid": {
        "type": "keyword"
      },
      "skuTitle": {
        "type": "text",
        "analyzer": "ik_smart"
      },
      "skuPrice": {
        "type": "keyword"
      },
      "skulmg": {
        "type": "keyword"
      }
    }
  }
}
```

```
"index": false,  
"doc_values": false  
},  
"saleCount": {  
    "type": "long"  
},  
"hasStock": {  
    "type": "boolean"  
},  
"hotScore": {  
    "type": "long"  
},  
"brandId": {  
    "type": "long"  
},  
"catalogId": {  
    "type": "long"  
},  
"brandName": {  
    "type": "keyword",  
    "index": false,  
    "doc_values": false  
},  
"brandImg": {  
    "type": "keyword",  
    "index": false,  
    "doc_values": false  
},  
"catalogName": {  
    "type": "keyword",  
    "index": false,  
    "doc_values": false  
},  
"attrs": {  
    "type": "nested",  
    "properties": {  
        "attrId": {  
            "type": "long"  
        },  
        "attrName": {  
            "type": "keyword",  
            "index": false,  
            "doc_values": false  
        },  
    },  
}
```

```
        "attrValue": {  
            "type": "keyword"  
        }  
    }  
}  
}  
}  
}  
}
```

index:

默认 true，如果为 false，表示该字段不会被索引，但是检索结果里面有，但字段本身不能当做检索条件。

doc_values:

默认 true，设置为 false，表示不可以做排序、聚合以及脚本操作，这样更节省磁盘空间。

还可以通过设定 doc_values 为 true，index 为 false 来让字段不能被搜索但可以用于排序、聚合以及脚本操作：

2、上架细节

上架是将后台的商品放在 es 中可以提供检索和查询功能

- 1) 、hasStock：代表是否有库存。默认上架的商品都有库存。如果库存无货的时候才需要更新一下 es
- 2) 、库存补上以后，也需要重新更新一下 es
- 3) 、hotScore 是热度值，我们只模拟使用点击率更新热度。点击率增加到一定程度才更新热度值。
- 4) 、下架就是从 es 中移除检索项，以及修改 mysql 状态

商品上架步骤：

- 1) 、先在 es 中按照之前的 mapping 信息，建立 product 索引。
- 2) 、点击上架，查询出所有 sku 的信息，保存到 es 中
- 3) 、es 保存成功返回，更新数据库的上架状态信息。



让天下没有难学的技术

3、数据一致性

- 1) 、商品无库存的时候需要更新 es 的库存信息
- 2) 、商品有库存也要更新 es 的信息



二、商品检索

1、检索业务分析

商品检索三个入口：

1) 选择分类进入商品检索



2) 输入检索关键字展示检索页



华为 | 小米手机 | 小米9 | 小米电视 | oppo | vivo | 手机 | 荣耀 | 小米8 | 苹果 | 魅族



3) 选择筛选条件进入

品牌:	小米	黑鲨科技	TCL	2
	多亲 (QIN)	朵唯 (DOOV)	JBL by HARMAN	S

分类:	手机	平板电视	耳机/耳麦	
分类:	智能门铃	智能猫眼	专业/家庭音响	便携音箱

检索条件&排序条件

- 全文检索: skuTitle
- 排序: saleCount、hotScore、skuPrice
- 过滤: hasStock、skuPrice 区间、brandId、catalogId、attrs
- 聚合: attrs

完整的 url 参数

keyword=小米&sort=saleCount_desc/asc&hasStock=0/1&skuPrice=400_1900&brandId=1&catalogId=1&attrs=1_3G:4G:5G&attrs=2_骁龙 845&attrs=4_高清屏

2、检索语句构建

1、请求参数模型

```
@Data
public class SearchParam {

    private String keyword;//页面传递过来的全文匹配关键字 v
    private Long catalog3Id;//三级分类id v

    /**
     * sort=saleCount_asc/desc
     * sort=skuPrice_asc/desc
     * sort=hotScore_asc/desc
     */
    private String sort;//排序条件 v

    /**
     *过滤条件
     * hasStock(是否有货)、skuPrice 区间、brandId、catalog3Id、attrs
     * hasStock=0/1
     * skuPrice=1_500/_500/500_
     * brandId=1
     * attrs=2_5 存:6 材
     *
     */
    private Integer hasStock;//是否只显示有货 v 0 (无库存) 1 (有库存)
    private String skuPrice;//价格区间查询 v
    private List<Long> brandId;//按照品牌进行查询，可以多选 v
    private List<String> attrs;//按照属性进行筛选 v
    private Integer pageNum = 1;//页码

    private String _queryString;//原生的所有查询条件
}
```

2、构建参数

```
private SearchRequest buildSearchRequest(SearchParam param) {
    SearchSourceBuilder sourceBuilder = new SearchSourceBuilder(); //构建
DSL 语句的

    /**
     * 查询：过滤（按照属性，分类，品牌，价格区间，库存）
     */
```

```
//1、构建bool - query
BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
//1.1、must- 模糊匹配,
if (!StringUtils.isEmpty(param.getKeyword())) {
    boolQuery.must(QueryBuilders.matchQuery("skuTitle",
param.getKeyword()));
}
//1.2、bool - filter - 按照三级分类id查询
if (param.getCatalog3Id() != null) {
    boolQuery.filter(QueryBuilders.termQuery("catalogId",
param.getCatalog3Id()));
}
//1.2、bool - filter - 按照品牌id查询
if (param.getBrandId() != null && param.getBrandId().size() > 0) {
    boolQuery.filter(QueryBuilders.termsQuery("brandId",
param.getBrandId()));
}
//1.2、bool - filter - 按照所有指定的属性进行查询
if (paramAttrs() != null && paramAttrs().size() > 0) {
    for (String attrStr : paramAttrs()) {
        //attrs=1_5寸:8寸&attrs=2_16G:8G
        BoolQueryBuilder nestedboolQuery = QueryBuilders.boolQuery();
        //attr = 1_5寸:8寸
        String[] s = attrStr.split("_");
        String attrId = s[0]; //检索的属性id
        String[] attrValues = s[1].split(":"); //这个属性的检索用的值
        nestedboolQuery.must(QueryBuilders.termQuery("attrs.attrId",
attrId));
        nestedboolQuery.must(QueryBuilders.termsQuery("attrs.attrValue",
attrValues));
        //每一个必须都得生成一个nested 查询
        NestedQueryBuilder nestedQuery =
QueryBuilders.nestedQuery("attrs", nestedboolQuery, ScoreMode.None);
        boolQuery.filter(nestedQuery);
    }
}

//1.2、bool - filter - 按照库存是否有进行查询
if(param.getHasStock() != null){
    boolQuery.filter(QueryBuilders.termQuery("hasStock",
param.getHasStock() == 1));
}
```

```
}

//1.2、bool - filter - 按照价格区间
if (!StringUtils.isEmpty(param.getSkuPrice())) {
    //1_500/_500/500_
    /**
     * "range": {
     *     "skuPrice": {
     *         "gte": 0,
     *         "lte": 6000
     *     }
     */
    RangeQueryBuilder rangeQuery =
    QueryBuilders.rangeQuery("skuPrice");

    String[] s = param.getSkuPrice().split("_");
    if (s.length == 2) {
        //区间
        rangeQuery.gte(s[0]).lte(s[1]);
    } else if (s.length == 1) {
        if (param.getSkuPrice().startsWith("_")) {
            rangeQuery.lte(s[0]);
        }

        if (param.getSkuPrice().endsWith("_")) {
            rangeQuery.gte(s[0]);
        }
    }

    boolQuery.filter(rangeQuery);
}

//把以前的所有条件都拿来进封装
sourceBuilder.query(boolQuery);

/**
 * 排序, 分页, 高亮,
 */
//2.1、排序
```

```
if (!StringUtils.isEmpty(param.getSort())) {
    String sort = param.getSort();
    //sort=hotScore_asc/desc
    String[] s = sort.split("_");
    SortOrder order = s[1].equalsIgnoreCase("asc") ? SortOrder.ASC : SortOrder.DESC;
    sourceBuilder.sort(s[0], order);
}
//2.2、分页 pageSize:5
// pageNum:1 from:0 size:5 [0,1,2,3,4]
// pageNum:2 from:5 size:5
//from = (pageNum-1)*size
sourceBuilder.from((param.getPageNum() - 1) * EsConstant.PRODUCT_PAGESIZE);
sourceBuilder.size(EsConstant.PRODUCT_PAGESIZE);

//2.3、高亮
if (!StringUtils.isEmpty(param.getKeyword())) {
    HighlightBuilder builder = new HighlightBuilder();
    builder.field("skuTitle");
    builder.preTags("<b style='color:red'>");
    builder.postTags("</b>");
    sourceBuilder.highlighter(builder);
}

/**
 * 聚合分析
 */
//1、品牌聚合
TermsAggregationBuilder brand_agg =
AggregationBuilders.terms("brand_agg");
brand_agg.field("brandId").size(50);
//品牌聚合的子聚合

brand_agg.subAggregation(AggregationBuilders.terms("brand_name_agg").field("brandName").size(1));

brand_agg.subAggregation(AggregationBuilders.terms("brand_img_agg").field("brandImg").size(1));
//TODO 1、聚合brand
sourceBuilder.aggregation(brand_agg);
//2、分类聚合 catalog_agg
```

```
    TermsAggregationBuilder catalog_agg =
AggregationBuilders.terms("catalog_agg").field("catalogId").size(20);

catalog_agg.subAggregation(AggregationBuilders.terms("catalog_name_agg")
).field("catalogName").size(1));
    //TODO 2、聚合catalog
    sourceBuilder.aggregation(catalog_agg);
    //3、属性聚合 attr_agg
    NestedAggregationBuilder attr_agg =
AggregationBuilders.nested("attr_agg", "attrs");

    //聚合出当前所有的 attrId
    TermsAggregationBuilder attr_id_agg =
AggregationBuilders.terms("attr_id_agg").field("attrs.attrId");
    //聚合分析出当前 attr_id 对应的名字

attr_id_agg.subAggregation(AggregationBuilders.terms("attr_name_agg")
.field("attrs.attrName").size(1));
    //聚合分析出当前 attr_id 对应的所有可能的属性值 attrValue

attr_id_agg.subAggregation(AggregationBuilders.terms("attr_value_agg")
.field("attrs.attrValue").size(50));
    attr_agg.subAggregation(attr_id_agg);
    //TODO 3、聚合attr
    sourceBuilder.aggregation(attr_agg);

    String s = sourceBuilder.toString();
    System.out.println("构建的 DSL" + s);

    SearchRequest searchRequest = new SearchRequest(new
String[]{EsConstant.PRODUCT_INDEX}, sourceBuilder);
    return searchRequest;
}
```

3、结果提取封装

1、响应数据模型

```
@Data
public class SearchResult {

    //查询到的所有商品信息
    private List<SkuEsModel> products;

    /**
     * 以下是分页信息
     */
    private Integer pageNum;//当前页码
    private Long total;//总记录数
    private Integer totalPages;//总页码
    private List<Integer> pageNavs;

    private List<BrandVo> brands;//当前查询到的结果，所有涉及到的品牌
    private List<CatalogVo> catalogs;//当前查询到的结果，所有涉及到的所有分类
    private List<AttrVo> attrs;//当前查询到的结果，所有涉及到的所有属性

    //=====以上是返回给页面的所有信息=====

    //面包屑导航数据
    private List<NavVo> navs = new ArrayList<>();
    private List<Long> attrIds = new ArrayList<>();

    @Data
    public static class NavVo{
        private String navName;
        private String navValue;
        private String link;
    }

    @Data
    public static class BrandVo{}
```

```
    private Long brandId;
    private String brandName;

    private String brandImg;
}

@Data
public static class CatalogVo{
    private Long catalogId;
    private String catalogName;

}

@Data
public static class AttrVo{
    private Long attrId;
    private String attrName;

    private List<String> attrValue;
}
}
```

2、响应结果封装

```
private SearchResult buildSearchResult(SearchResponse response,
SearchParams param) {

    SearchResult result = new SearchResult();
    //1. 返回的所有查询到的商品
    SearchHits hits = response.getHits();
    List<SkuEsModel> esModels = new ArrayList<>();
    if (hits.getHits() != null && hits.getHits().length > 0) {
        for (SearchHit hit : hits.getHits()) {
            String sourceAsString = hit.getSourceAsString();
            SkuEsModel esModel = JSON.parseObject(sourceAsString,
SkuEsModel.class);
            if (!StringUtils.isEmpty(param.getKeyword())){
                HighlightField skuTitle =
hit.getHighlightFields().get("skuTitle");
                String string = skuTitle.getFragments()[0].string();
                esModel.setSkuTitle(string);
            }
        }
    }
}
```

```
        }
        esModels.add(esModel);
    }
;
}
result.setProducts(esModels);

// //2、当前所有商品涉及到的所有属性信息
List<SearchResult.AttrVo> attrVos = new ArrayList<>();
ParsedNested attr_agg =
response.getAggregations().get("attr_agg");
ParsedLongTerms attr_id_agg =
attr_agg.getAggregations().get("attr_id_agg");
for (Terms.Bucket bucket : attr_id_agg.getBuckets()) {
    SearchResult.AttrVo attrVo = new SearchResult.AttrVo();
    //1、得到属性的id
    long attrId = bucket.getKeyAsNumber().longValue();
    //2、得到属性的名字
    String attrName = ((ParsedStringTerms)
bucket.getAggregations().get("attr_name_agg")).getBuckets().get(0).getKeyAsString();

    //3、得到属性的所有值
    List<String> attrValues = ((ParsedStringTerms)
bucket.getAggregations().get("attr_value_agg")).getBuckets().stream().map(item -> {
        String keyAsString = ((Terms.Bucket)
item.getKeyAsString());
        return keyAsString;
    }).collect(Collectors.toList());

    attrVo.setAttrId(attrId);
    attrVo.setAttrName(attrName);
    attrVo.setAttrValue(attrValues);

    attrVos.add(attrVo);
}
```

```
result.setAttrs(attrVos);
//      //3、当前所有商品涉及到的所有品牌信息
List<SearchResult.BrandVo> brandVos = new ArrayList<>();
ParsedLongTerms brand_agg =
response.getAggregations().get("brand_agg");
for (Terms.Bucket bucket : brand_agg.getBuckets()) {
    SearchResult.BrandVo brandVo = new SearchResult.BrandVo();
    //1、得到品牌的id
    long brandId = bucket.getKeyAsNumber().longValue();
    //2、得到品牌的名
    String brandName = ((ParsedStringTerms)
bucket.getAggregations().get("brand_name_agg")).getBuckets().get(0).getKeyAsString();
    //3、得到品牌的图片
    String brandImg = ((ParsedStringTerms)
bucket.getAggregations().get("brand_img_agg")).getBuckets().get(0).getKeyAsString();
    brandVo.setBrandId(brandId);
    brandVo.setBrandName(brandName);
    brandVo.setBrandImg(brandImg);
    brandVos.add(brandVo);
}

result.setBrands(brandVos);
//      //4、当前所有商品涉及到的所有分类信息
ParsedLongTerms catalog_agg =
response.getAggregations().get("catalog_agg");
List<SearchResult.CatalogVo> catalogVos = new ArrayList<>();
List<? extends Terms.Bucket> buckets = catalog_agg.getBuckets();
for (Terms.Bucket bucket : buckets) {
    SearchResult.CatalogVo catalogVo = new
SearchResult.CatalogVo();
    //得到分类id
    String keyAsString = bucket.getKeyAsString();
    catalogVo.setCatalogId(Long.parseLong(keyAsString));

    //得到分类名
    ParsedStringTerms catalog_name_agg =
bucket.getAggregations().get("catalog_name_agg");
    String catalog_name =
catalog_name_agg.getBuckets().get(0).getKeyAsString();
    catalogVo.setCatalogName(catalog_name);
    catalogVos.add(catalogVo);
```

```
        }

        result.setCatalogs(catalogVos);
//      =====以上从聚合信息中获取=====

//      //5、分页信息-页码
        result.setPageNum(param.getPageNum());
//      //5、分页信息-总记录数
        long total = hits.getTotalHits().value;
        result.setTotal(total);
//      //5、分页信息-总页码-计算 11/2 = 5 .. 1
        int totalPages = (int) total % EsConstant.PRODUCT_PAGESIZE == 0 ?
(int) total / EsConstant.PRODUCT_PAGESIZE : ((int) total /
EsConstant.PRODUCT_PAGESIZE + 1);
        result.setTotalPages(totalPages);

        List<Integer> pageNavs = new ArrayList<>();
        for (int i=1;i<=totalPages;i++){
            pageNavs.add(i);
        }
        result.setPageNavs(pageNavs);

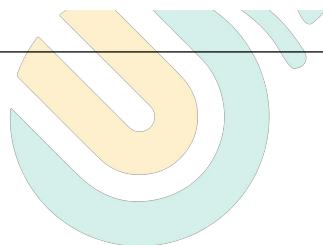
//6、构建面包屑导航功能
if(param.getAttrs()!=null && param.getAttrs().size()>0){
    List<SearchResult.NavVo> collect =
param.getAttrs().stream().map(attr -> {
    //1、分析每个 attrs 传过来的查询参数值。
    SearchResult.NavVo navVo = new SearchResult.NavVo();
//    attrs=2_5 存:6 个
    String[] s = attr.split("_");
    navVo.setNavValue(s[1]);
    R r = productFeignService.attrInfo(Long.parseLong(s[0]));
    result.getAttributeIds().add(Long.parseLong(s[0]));
    if(r.getCode() == 0){
        AttrResponseVo data = r.getData("attr", new
TypeReference<AttrResponseVo>() {
            });
        navVo.setNavName( data.getAttributeName());
    }else{
        navVo.setNavName(s[0]);
    }
})
```

```
//  
    //2、取消了这个面包屑以后，我们要跳转到那个地方。将请求地址的url  
    里面的当前置空  
    //拿到所有的查询条件，去掉当前。  
    //attrs= 15_海思(Hisilicon)  
    String replace = replaceQueryString(param, attr, "attrs");  
  
    navVo.setLink("http://search.gulimall.com/list.html?"+replace);  
  
    return navVo;  
}).collect(Collectors.toList());  
  
    result.setNavs(collect);  
}  
  
//品牌，分类  
if(param.getBrandId()!=null && param.getBrandId().size()>0){  
    List<SearchResult.NavVo> navs = result.getNavs();  
    SearchResult.NavVo navVo = new SearchResult.NavVo();  
  
    navVo.setNavName("品牌");  
    //TODO 远程查询所有品牌  
    R r = productFeignService.brandsInfo(param.getBrandId());  
    if(r.getCode() == 0){  
        List<BrandVo> brand = r.getData("brand", new  
TypeReference<List<BrandVo>>() {  
    });  
        StringBuffer buffer = new StringBuffer();  
        String replace = "";  
        for (BrandVo brandVo : brand) {  
            buffer.append(brandVo.getBrandName()+";");  
            replace = replaceQueryString(param,  
brandVo.getBrandId()+"", "brandId");  
        }  
        navVo.setNavValue(buffer.toString());  
  
navVo.setLink("http://search.gulimall.com/list.html?"+replace);  
    }  
  
    navs.add(navVo);  
}
```

```
//TODO 分类: 不需要导航取消

    return result;
}

private String replaceQueryString(SearchParam param, String
value, String key) {
    String encode = null;
    try {
        encode = URLEncoder.encode(value, "UTF-8");
        encode = encode.replace("+", "%20"); // 浏览器对空格编码和 java 不一样
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return param.get_queryString().replace("&" + key + "=" + encode, "");
}
```



三、商品详情

1、详情数据

```
// 1. 获取sku的基本信息    0.5s  
// 2. 获取sku的图片信息    0.5s  
// 3. 获取sku的促销信息    1s  
// 4. 获取spu的所有销售属性   1s  
// 5. 获取规格参数组及组下的规格参数   1.5s  
// 6. spu详情      1s
```

2、查询详情

```
@Override
public SkuItemVo item(Long skuId) throws ExecutionException, InterruptedException {
    SkuItemVo skuItemVo = new SkuItemVo();

    CompletableFuture<SkuInfoEntity> infoFuture = CompletableFuture.supplyAsync(() ->
{
    //1、sku 基本信息获取 pms_sku_info
    SkuInfoEntity info = getById(skuId);
    skuItemVo.setInfo(info);
    return info;
}, executor);

    CompletableFuture<Void> saleAttrFuture = infoFuture.thenAcceptAsync(res -> {
        //3、获取spu 的销售属性组合。
        List<SkuItemSaleAttrVo> saleAttrVos =
skuSaleAttrValueService.getSaleAttrsBySpuId(res.getSpuId());
        skuItemVo.setSaleAttr(saleAttrVos);
    }, executor);

    CompletableFuture<Void> descFuture = infoFuture.thenAcceptAsync(res -> {
        //4、获取spu 的介绍 pms_spu_info_desc
        SpuInfoDescEntity spuInfoDescEntity =
spuInfoDescService.getById(res.getSpuId());
        skuItemVo.setDesp(spuInfoDescEntity);
    }, executor);

    CompletableFuture<Void> baseAttrFuture = infoFuture.thenAcceptAsync(res -> {
        //5、获取spu 的规格参数信息。
        List<SpuItemAttrGroupVo> attrGroupVos =
attrGroupService.getAttrGroupWithAttrsBySpuId(res.getSpuId(), res.getCatalogId());
        skuItemVo.setGroupAttrs(attrGroupVos);
    }, executor);

    //2、sku 的图片信息 pms_sku_images
    CompletableFuture<Void> imageFuture = CompletableFuture.runAsync(() -> {
        List<SkuImagesEntity> images = imagesService.getImagesBySkuId(skuId);
        skuItemVo.setImages(images);
    }, executor);
}
```

```
//等到所有任务都完成
CompletableFuture.allOf(saleAttrFuture,descFuture,baseAttrFuture,imageFuture).get()
;

return skuItemVo;
}
```

3、sku 组合切换

页面遍历

```
<div class="box-attr clear" th:each="attr:${item.saleAttr}">
    <dl>
        <dt>选择[[${attr.attrName}]]</dt>
        <dd th:each="vals:${attr.attrValues}">
            >
            <a class="sku_attr_value"
                th:attr="skus=${vals.skuIds},class=${#lists.contains(#strings.listSplit(vals.skuIds,',','), item.info.skuId.toString())?'sku_attr_value checked':'sku_attr_value'}"
                href="#">[[${vals.attrValue}]]</a>
            <!--
                 -->
            </a>
        </dd>
    </dl>
</div>
```

动态切换

```
$(".sku_attr_value").click(function(){
    //1、点击的元素先添加上自定义的属性。为了识别我们是刚才被点击的
    var skus = new Array();
    $(this).addClass("clicked");
    var curr = $(this).attr("skus").split(",");
    //当前被点击的所有 sku 组合数组放进去
    skus.push(curr);
    //去掉同一行的所有的 checked

    $(this).parent().parent().find(".sku_attr_value").removeClass("checked");

    $("a[class='sku_attr_value checked']").each(function(){
```

```
skus.push($(this).attr("skus").split(","));
});

console.log(skus);

//2、取出他们的交集，得到skuId
var filterEle = skus[0];
for(var i = 1;i<skus.length;i++){
    filterEle = $(filterEle).filter(skus[i]);
}

console.log(filterEle[0]);
location.href = "http://item.gulimall.com/" + filterEle[0] + ".html";

//4、跳转
});
```

关键 sql

```
SELECT
    ssav.`attr_id` attr_id,
    ssav.`attr_name` attr_name,
    ssav.`attr_value` ,
    GROUP_CONCAT(DISTINCT info.`sku_id`) sku_ids
FROM `pms_sku_info` info
LEFT JOIN `pms_sku_sale_attr_value` ssav ON
ssav.`sku_id`=info.`sku_id`
WHERE info.`spu_id`=#{spuId}
GROUP BY ssav.`attr_id`,ssav.`attr_name`,ssav.`attr_value`
```

	attr_id	attr_name	attr_value	sku_ids
	9	颜色	亮黑色	3,4
	9	颜色	星河银	1,2
	9	颜色	罗兰紫	7,8
	9	颜色	碧冷翠	5,6
	12	版本	8GB+128GB	2,4,6,8
	12	版本	8GB+256GB	1,3,5,7

四、购物车

1、购物车需求

1)、需求描述：

- 用户可以在**登录状态**下将商品添加到购物车 【**用户购物车/在线购物车**】

- 放入数据库

- **mongodb**

- **放入 redis (采用)**

 登录以后，会将**临时购物车**的数据全部合并过来，并清空**临时购物车**；

- 用户可以在**未登录状态**下将商品添加到购物车 【**游客购物车/离线购物车/临时购物车**】

- 放入 localstorage (客户端存储，后台不存)

- cookie

- WebsQL

- **放入 redis (采用)**

 浏览器即使关闭，下次进入，**临时购物车**数据都在

- 用户可以使用购物车一起结算下单

- 给购物车添加商品

- 用户可以查询自己的购物车

- 用户可以在购物车中修改购买商品的数量。

- 用户可以在购物车中删除商品。

- 选中不选中商品

- 在购物车中展示商品优惠信息

- 提示购物车商品价格变化

2) 、数据结构

您还没有登录！登录后购物车的商品将保存到您的账号中 [立即登录](#)

全部商品 1 免息 购物车中1件商品已享白条免息 [查看](#) 配送至：北京昌平区北七家镇

<input checked="" type="checkbox"/> 全选	商品	单价	数量	小计	操作
<input checked="" type="checkbox"/> 京东自营	三星 Galaxy A90 5G (SM-A9080) 全息黑 疾速5G 骁龙855 8GB+128GB 全8GB+128GB	¥3299.00	<input type="button" value="1"/> +	¥3299.00	删除 移到我的关注
选服务					

全选 [删除选中商品](#) [移到关注](#) [清理购物车](#) 已选择 1 件商品 ^ 总价：¥3299.00 [去结算](#)

因此每一个购物项信息，都是一个对象，基本字段包括：

```
{
  skuid: 2131241,
  check: true,
  title: "Apple iphone.....",
  defaultImage: "...",
  price: 4999,
  count: 1,
  totalPrice: 4999,
  skuSaleVO: {...}
}
```

另外，购物车中不止一条数据，因此最终会是对象的数组。即：

```
[...], [...], [...]
```

Redis 有 5 种不同数据结构，这里选择哪一种比较合适呢？`Map<String, List<String>>`

- 首先不同用户应该有独立的购物车，因此购物车应该以用户的作为 key 来存储，Value 是用户的所有购物车信息。这样看来基本的`k-v`结构就可以了。
- 但是，我们对购物车中的商品进行增、删、改操作，基本都需要根据商品 id 进行判断，为了方便后期处理，我们的购物车也应该是`k-v`结构，key 是商品 id，value 才是这个商品的购物车信息。

综上所述，我们的购物车结构是一个双层 Map：`Map<String, Map<String, String>>`

- 第一层 Map，Key 是用户 id
- 第二层 Map，Key 是购物车中商品 id，值是购物项数据

3) 、流程

参照京东

Network	Performance	Memory	Application	Security	Audits
Name	Value	Domain	Path	Expires / Max-Age	
pinld	Q4Pws5W9mrY	jd.com	/	2020-10-03T11:59:29.064Z	
shshshfp	e5bbcf831bb9add04de0705be032fc6	jd.com	/	2047-02-19T11:59:52.000Z	
shshshfp_a	ad7299f0-dd22-0e8b-bf4c-858c331d9836-1532...	jd.com	/	2047-02-19T11:59:53.000Z	
shshshfp_b	000a2d7f6300517c64196f460034d1fa399a563...	jd.com	/	2047-02-19T11:59:53.000Z	
shshshfp_D	517f8fd086c2c50cc4e3ba59c3cd172c_5_1570190...	jd.com	/	2019-10-04T12:29:52.000Z	
thor	5E127CCA29B7876F145FC0B2E463FFB232633B...	jd.com	/	N/A	
unick	%E6%8B%9C%E5%95%8A%E6%8B%9C_818	jd.com	/	2019-11-03T11:59:29.064Z	
unpl	V2_ZnNbUpREUlmWkFRKrwLUmJWF1RLUUUU...	jd.com	/	2019-11-01T07:18:54.839Z	
user-key	2e9c2130-c40c-4008-849e-630296df0ea	jd.com	/	2019-10-31T12:01:10.646Z	
wifstk_smdl	6ousken85bb2fkj547iqmwn8082k01f7	jd.com	/	N/A	

user-key 是随机生成的 id，不管有没有登录都会有这个 cookie 信息。

两个功能：新增商品到购物车、查询购物车。

新增商品：判断是否登录

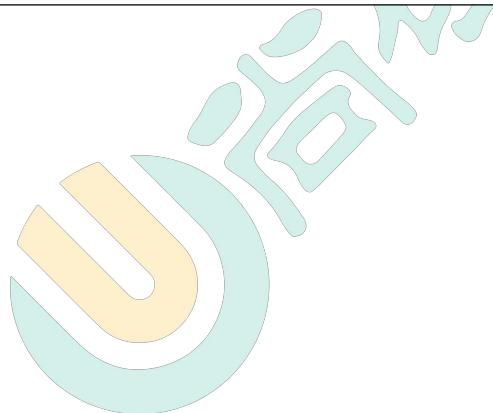
- 是：则添加商品到后台 Redis 中，把 user 的唯一标识符作为 key。
- 否：则添加商品到后台 redis 中，使用随机生成的 user-key 作为 key。

查询购物车列表：判断是否登录

- 否：直接根据 user-key 查询 redis 中数据并展示
- 是：已登录，则需要先根据 user-key 查询 redis 是否有数据。
 - 有：需要提交到后台添加到 redis，合并数据，而后查询。
 - 否：直接去后台查询 redis，而后返回。

2、临时购物车

```
/**  
 * 获取到我们要操作的购物车  
 * @return  
 */  
  
private BoundHashOperations<String, Object, Object> getCartOps() {  
    UserInfoTo userInfoTo = CartInterceptor.threadLocal.get();  
    String cartKey = "";  
    if (userInfoTo.getUserId() != null) {  
        //gulimall:cart:1  
        cartKey = CART_PREFIX + userInfoTo.getUserId();  
    } else {  
        cartKey = CART_PREFIX + userInfoTo.getUserKey();  
    }  
  
    BoundHashOperations<String, Object, Object> operations =  
    redisTemplate.boundHashOps(cartKey);  
    return operations;  
}
```



3、登录购物车

```
@Override
public Cart getCart() throws ExecutionException, InterruptedException {

    Cart cart = new Cart();
    UserInfoTo userInfoTo = CartInterceptor.threadLocal.get();
    if(userInfoTo.getUserId()!=null){
        //1、登录
        String cartKey =CART_PREFIX+ userInfoTo.getUserId();
        //2、如果临时购物车的数据还没有进行合并【合并购物车】
        String tempCartKey = CART_PREFIX + userInfoTo.getUserKey();
        List<CartItem> tempCartItems = getCartItems(tempCartKey);
        if(tempCartItems!=null){
            //临时购物车有数据，需要合并
            for (CartItem item : tempCartItems) {
                addToCart(item.getSkuId(),item.getCount());
            }
            //清除临时购物车的数据
            clearCart(tempCartKey);
        }

        //3、获取登录后的购物车的数据【包含合并过来的临时购物车的数据，和登录后的购物车的数据】
        List<CartItem> cartItems = getCartItems(cartKey);
        cart.setItems(cartItems);

    }else{
        //2、没登录
        String cartKey =CART_PREFIX+ userInfoTo.getUserKey();
        //获取临时购物车的所有购物项
        List<CartItem> cartItems = getCartItems(cartKey);
        cart.setItems(cartItems);

    }
    return cart;
}
```

五、订单

1、订单中心

电商系统涉及到3流，分别是信息流，资金流，物流，而订单系统作为中枢将三者有机的集合起来。

订单模块是电商系统的枢纽，在订单这个环节上需求获取多个模块的数据和信息，同时对这些信息进行加工处理后流向下一个环节，这一系列就构成了订单的信息流通。

1、订单构成



1、用户信息

用户信息包括用户账号、用户等级、用户的收货地址、收货人、收货人电话等组成，用户账户需要绑定手机号码，但是用户绑定的手机号码不一定是收货信息上的电话。用户可以添加多个收货信息，用户等级信息可以用来和促销系统进行匹配，获取商品折扣，同时用户等级还可以获取积分的奖励等。

2、订单基础信息

订单基础信息是订单流转的核心，其包括订单类型、父/子订单、订单编号、订单状态、订单流转的时间等。

(1) 订单类型包括实体商品订单和虚拟订单商品等，这个根据商城商品和服务类型进行区分。

(2) 同时订单都需要做父子订单处理，之前在初创公司一直只有一个订单，没有做父子订单处理后期需要进行拆单的时候就比较麻烦，尤其是多商户商场，和不同仓库商品的时候，父子订单就是为后期做拆单准备的。

(3) 订单编号不多说了，需要强调的一点是父子订单都需要有订单编号，需要完善的时候可以对订单编号的每个字段进行统一定义和诠释。

(4) 订单状态记录订单每次流转过程，后面会对订单状态进行单独的说明。

(5) 订单流转时间需要记录下单时间，支付时间，发货时间，结束时间/关闭时间等等

3. 商品信息

商品信息从商品库中获取商品的 SKU 信息、图片、名称、属性规格、商品单价、商户信息等，从用户下单行为记录的用户下单数量，商品合计价格等。

4. 优惠信息

优惠信息记录用户参与的优惠活动，包括优惠促销活动，比如满减、满赠、秒杀等，用户使用的优惠券信息，优惠券满足条件的优惠券需要默认展示出来，具体方式已在之前的优惠券篇章做过详细介绍，另外还虚拟币抵扣信息等进行记录。

为什么把优惠信息单独拿出来而不放在支付信息里面呢？

因为优惠信息只是记录用户使用的条目，而支付信息需要加入数据进行计算，所以做为区分。

5. 支付信息

(1) 支付流水单号，这个流水单号是在唤起网关支付后支付通道返回给电商业务平台的支付流水号，财务通过订单号和流水单号与支付通道进行对账使用。

(2) 支付方式用户使用的支付方式，比如微信支付、支付宝支付、钱包支付、快捷支付等。支付方式有时候可能有两个——余额支付+第三方支付。

(3) 商品总金额，每个商品加总后的金额；运费，物流产生的费用；优惠总金额，包括促销活动的优惠金额，优惠券优惠金额，虚拟积分或者虚拟币抵扣的金额，会员折扣的金额等之和；实付金额，用户实际需要付款的金额。

$$\text{用户实付金额} = \text{商品总金额} + \text{运费} - \text{优惠总金额}$$

6. 物流信息

物流信息包括配送方式，物流公司，物流单号，物流状态，物流状态可以通过第三方接口来获取和向用户展示物流每个状态节点。

2、订单状态

1. 待付款

用户提交订单后，订单进行预下单，目前主流电商网站都会唤起支付，便于用户快速完成支付，需要注意的是待付款状态下可以对库存进行锁定，锁定库存需要配置支付超时时间，超时后将自动取消订单，订单变更关闭状态。

2. 已付款/待发货

用户完成订单支付，订单系统需要记录支付时间，支付流水单号便于对账，订单下放到 WMS



让天下没有难学的技术

系统，仓库进行调拨，配货，分拣，出库等操作。

3. 待收货/已发货

仓储将商品出库后，订单进入物流环节，订单系统需要同步物流信息，便于用户实时知悉物品物流状态

4. 已完成

用户确认收货后，订单交易完成。后续支付侧进行结算，如果订单存在问题进入售后状态

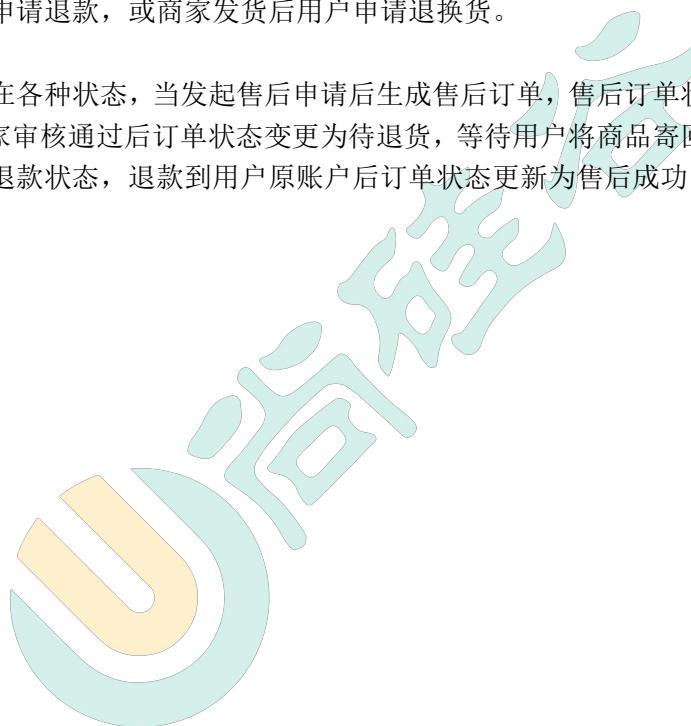
5. 已取消

付款之前取消订单。包括超时未付款或用户商户取消订单都会产生这种订单状态。

6. 售后中

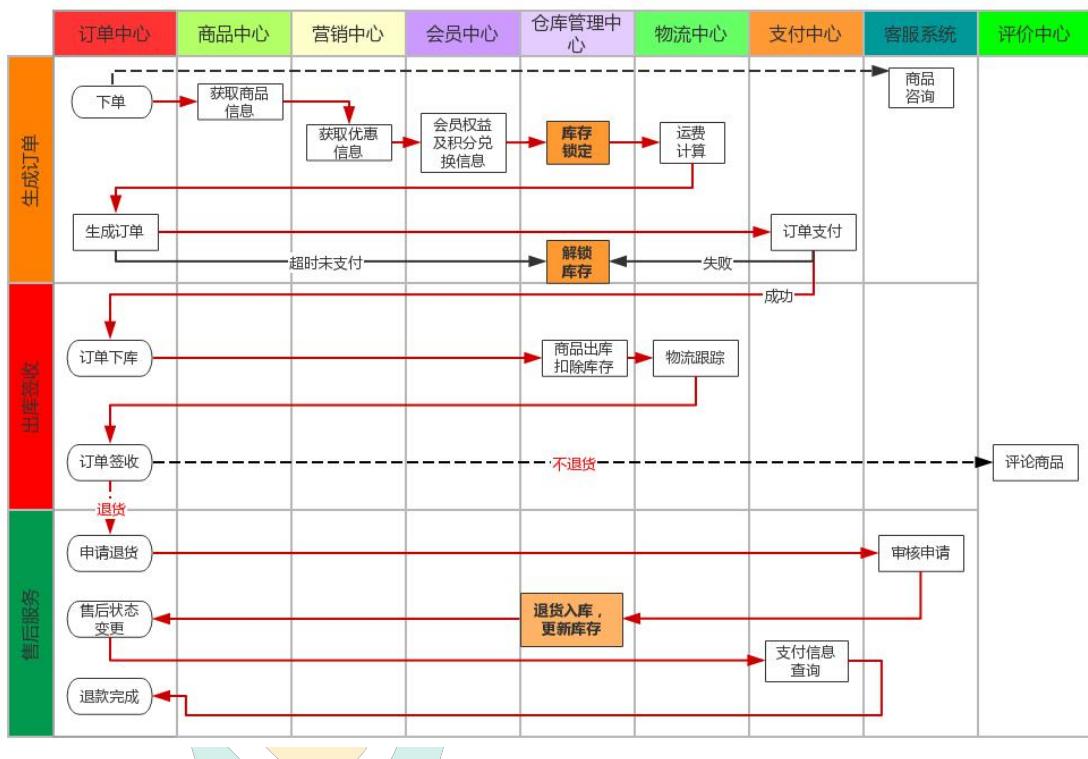
用户在付款后申请退款，或商家发货后用户申请退换货。

售后也同样存在各种状态，当发起售后申请后生成售后订单，售后订单状态为待审核，等待商家审核，商家审核通过后订单状态变更为待退货，等待用户将商品寄回，商家收货后订单状态更新为待退款状态，退款到用户原账户后订单状态更新为售后成功。



2、订单流程

订单流程是指从订单产生到完成整个流转的过程，从而行程了一套标准流程规则。而不同的产品类型或业务类型在系统中的流程会千差万别，比如上面提到的线上实物订单和虚拟订单的流程，线上实物订单与 O2O 订单等，所以需要根据不同的类型进行构建订单流程。不管类型如何订单都包括正向流程和逆向流程，对应的场景就是购买商品和退换货流程，正向流程就是一个正常的网购步骤：订单生成->支付订单->卖家发货->确认收货->交易成功。而每个步骤的背后，订单是如何在多系统之间交互流转的，可概括如下图



1、订单创建与支付

- (1) 、订单创建前需要预览订单，选择收货信息等
- (2) 、订单创建需要锁定库存，库存有才可创建，否则不能创建
- (3) 、订单创建后超时未支付需要解锁库存
- (4) 、支付成功后，需要进行拆单，根据商品打包方式，所在仓库，物流等进行拆单
- (5) 、支付的每笔流水都需要记录，以待查账
- (6) 、订单创建，支付成功等状态都需要给 MQ 发送消息，方便其他系统感知订阅

2、逆向流程

- (1) 、修改订单，用户没有提交订单，可以对订单一些信息进行修改，比如配送信息，优惠信息，及其他一些订单可修改范围的内容，此时只需对数据进行变更即可。
- (2) 、订单取消，**用户主动取消订单和用户超时未支付**，两种情况下订单都会取消订单，而超时情况是系统自动关闭订单，所以在订单支付的响应机制上面要做支付的

限时处理，尤其是在前面说的下单减库存的情形下面，可以保证快速的释放库存。另外需要需要处理的是促销优惠中使用的优惠券，权益等视平台规则，进行相应补回给用户。

- (3) 、退款，在待发货订单状态下取消订单时，分为缺货退款和用户申请退款。如果是全部退款则订单更新为关闭状态，若只是做部分退款则订单仍需进行进行，同时生成一条退款的售后订单，走退款流程。退款金额需原路返回用户的账户。
- (4) 、发货后的退款，发生在仓储货物配送，在配送过程中商品遗失，用户拒收，用户收货后对商品不满意，这样情况下用户发起退款的售后诉求后，需要商户进行退款的审核，双方达成一致后，系统更新退款状态，对订单进行退款操作，金额原路返回用户的账户，同时关闭原订单数据。仅退款情况下暂不考虑仓库系统变化。如果发生双方协调不一致情况下，可以申请平台客服介入。在退款订单商户不处理的情况下，系统需要做限期判断，比如 5 天商户不处理，退款单自动变更同意退款。



3、幂等性处理

参照幂等性文档

4、订单业务

1、搭建环境

订单服务引入页面，nginx 配置动静分离，上传静态资源到 nginx。编写 controller 跳转逻辑

2、订单确认页



可以发现订单结算页，包含以下信息：

1. 收货人信息：有更多地址，即有多个收货地址，其中有一个默认收货地址
2. 支付方式：货到付款、在线支付，不需要后台提供
3. 送货清单：配送方式（不做）及商品列表（根据购物车选中的 skuid 到数据库中查询）
4. 发票：不做
5. 优惠：查询用户领取的优惠券（不做）及可用积分（京豆）

OrderConfirmVo

```

@Data
public class OrderConfirmVO {
    // 收货地址, ums_member_receive_address 表
    private List<MemberReceiveAddressEntity> addresses;

    // 购物清单，根据购物车页面传递过来的 skuids 查询
    private List<OrderItemVO> orderItems;

    // 可用积分，ums_member 表中的 integration 字段
    private Integer bounds;

    // 订单令牌，防止重复提交
    private String orderToken;
}

OrderItemVO (参照 Cart 对象)
public class OrderItemVo {
}

```

```
private Long skuId;
private Boolean check = true;
private String title;
private String image;
private List<String> skuAttr;
private BigDecimal price;
private Integer count;
private BigDecimal totalPrice;
```

3、创建订单

当用户点击提交订单按钮，应该收集页面数据提交到后台并生成订单数据。

1、数据模型

订单确认页，需要提交的数据：

```
@Data
public class OrderSubmitVO {

    //提交上次订单确认页给你的令牌;
    private String orderToken;

    private BigDecimal apyPrice; // 校验总价格时，拿计算价格和这个价格比较

    private Integer payType;//0-在线支付 1-货到付款

    private String delivery_company; // 配送方式

    // 订单清单可以不用提交，继续从购物车中获取

    // 地址信息，提交地址 id，会员 id 不需要提交
    private Long addrId;

    // TODO: 发票相关信息略
    // TODO: 营销信息等
}
```

提交以后，需要响应的数据：

```
@Data
public class OrderSubmitResponseVO {
```

```
private OrderEntity orderEntity;  
private Integer code;  
// 1-不可重复提交或页面已过期 2-库存不足 3-价格校验不合法 等  
}
```

2、防止超卖

数据库 `unsigned int` 做最后的保证。

4、自动关单

订单超时未支付，需要取消订单

5、解锁库存

订单关闭，需要解锁已经占用的库存

库存锁定成功，订单回滚，保证最终一致性，也需要库存自动解锁

4、5 功能参照消息队列流程完成

六、秒杀

1、秒杀业务

秒杀具有瞬间高并发的特点，针对这一特点，必须要做限流 + 异步 + 缓存（页面静态化）+ 独立部署。

限流方式：

1. 前端限流，一些高并发的网站直接在前端页面开始限流，例如：小米的验证码设计
2. nginx 限流，直接负载部分请求到错误的静态页面：令牌算法 漏斗算法
3. 网关限流，限流的过滤器
4. 代码中使用分布式信号量
5. rabbitmq 限流（能者多劳： channel.basicQos(1)），保证发挥所有服务器的性能。

2、秒杀流程

见秒杀流程图

3、限流

参照 Alibaba Sentinel



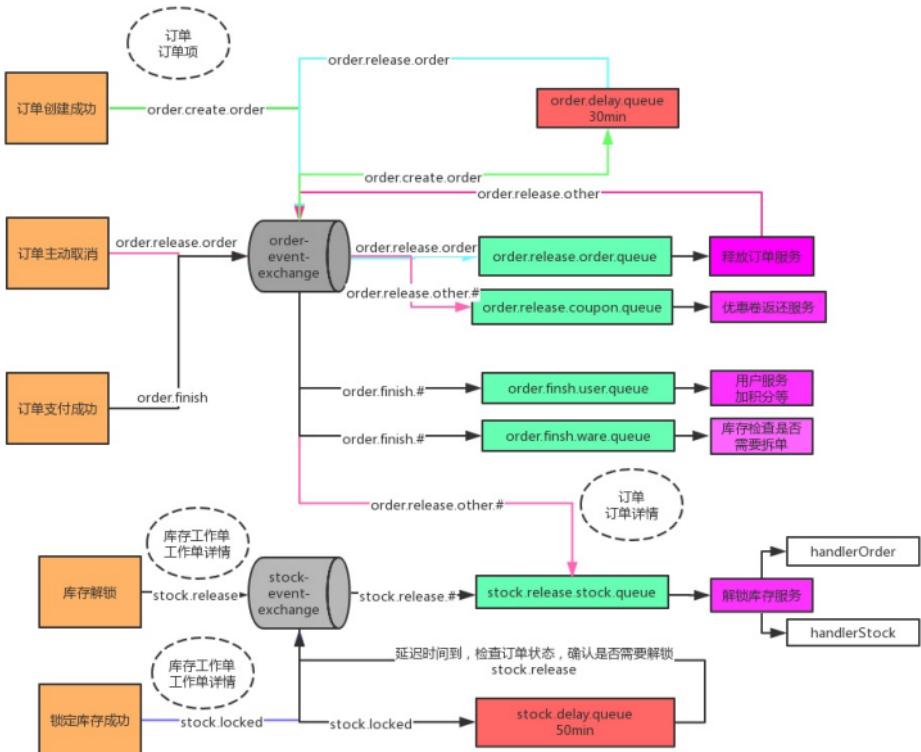
RabbitMQ

Message Queue消息队列



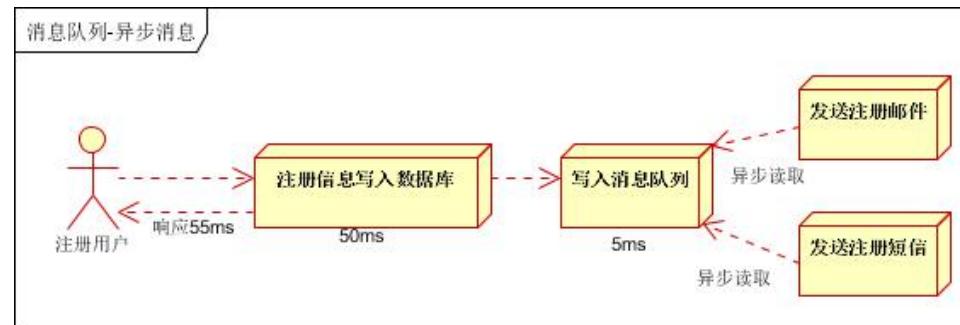
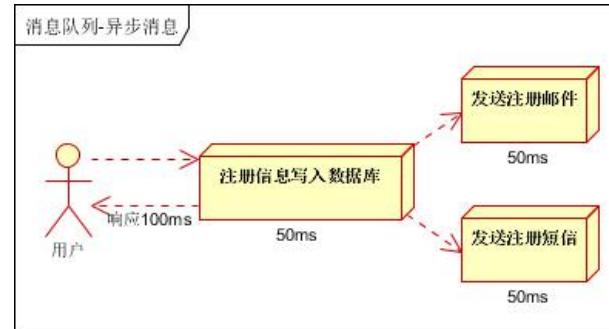
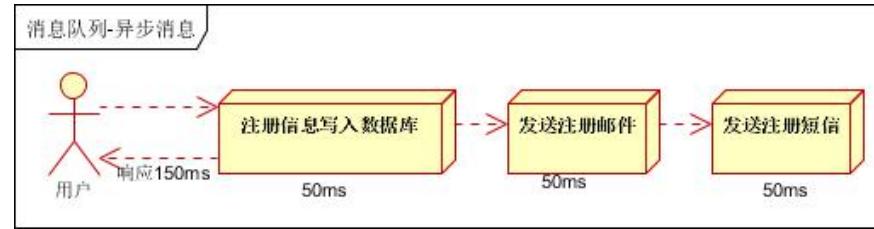
设计建议规范：(基于事件模型的交换机设计)

- 1、交换机命名：业务+exchange；交换机为Topic
- 2、路由键：事件.需要感知的业务(可以不写)
- 3、队列命名：事件+想要监听服务名+queue
- 4、绑定关系：事件.感知的业务(#)



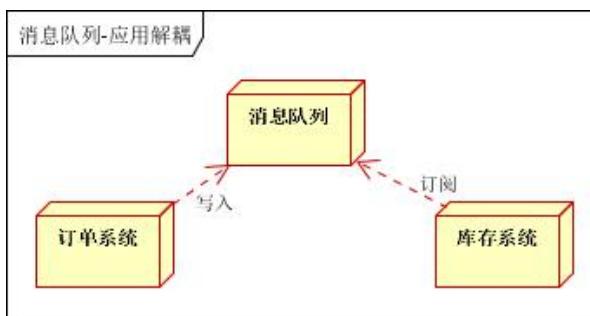
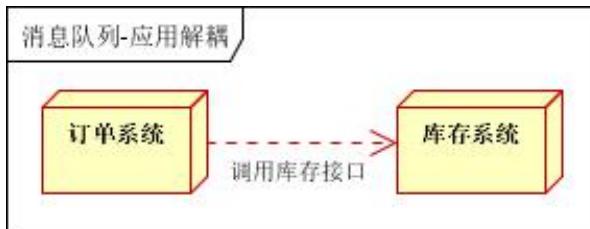


消息中间件

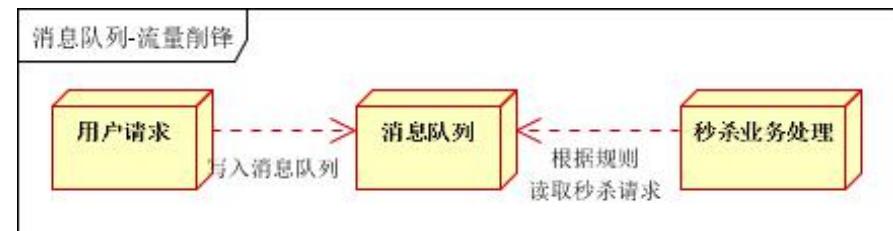


异步处理

应用解耦



流量控制





一、概述

1. 大多应用中，可通过消息服务中间件来提升系统异步通信、扩展解耦能力
2. 消息服务中两个重要概念：

消息代理 (message broker) 和目的地 (destination)

当消息发送者发送消息以后，将由消息代理接管，消息代理保证消息传递到指定目的地。

3. 消息队列主要有两种形式的目的地
 1. 队列 (queue) : 点对点消息通信 (point-to-point)
 2. 主题 (topic) : 发布 (publish) / 订阅 (subscribe) 消息通信

4. 点对点式：

- 消息发送者发送消息，消息代理将其放入一个队列中，消息接收者从队列中获取消息内容，消息读取后被移出队列
- 消息只有唯一的发送者和接受者，但并不是说只能有一个接收者

5. 发布订阅式：

- 发送者（发布者）发送消息到主题，多个接收者（订阅者）监听（订阅）这个主题，那么就会在消息到达时同时收到消息

6. JMS (Java Message Service) JAVA消息服务：

- 基于JVM消息代理的规范。ActiveMQ、HornetMQ是JMS实现

7. AMQP (Advanced Message Queuing Protocol)

- 高级消息队列协议，也是一个消息代理的规范，兼容JMS
- RabbitMQ是AMQP的实现

	JMS (Java Message Service)	AMQP (Advanced Message Queuing Protocol)
定义	Java api	网络线级协议
跨语言	否	是
跨平台	否	是
Model	<p>提供两种消息模型：</p> <ul style="list-style-type: none"> (1) 、Peer-2-Peer (2) 、Pub/sub 	<p>提供了五种消息模型：</p> <ul style="list-style-type: none"> (1) 、direct exchange (2) 、fanout exchange (3) 、topic change (4) 、headers exchange (5) 、system exchange <p>本质来讲，后四种和JMS的pub/sub模型没有太大差别，仅是在路由机制上做了更详细的划分；</p>
支持消息类型	<p>多种消息类型：</p> <p>TextMessage MapMessage BytesMessage StreamMessage ObjectMessage Message (只有消息头和属性)</p>	<p>byte[]</p> <p>当实际应用时，有复杂的消息，可以将消息序列化后发送。</p>
综合评价	<p>JMS 定义了JAVA API层面的标准；在java体系中，多个client均可以通过JMS进行交互，不需要应用修改代码，但是其对跨平台的支持较差；</p>	<p>AMQP定义了wire-level层的协议标准；天然具有跨平台、跨语言特性。</p>

8. Spring支持

- spring-jms提供了对JMS的支持
- spring-rabbit提供了对AMQP的支持
- 需要ConnectionFactory的实现来连接消息代理
- 提供JmsTemplate、RabbitTemplate来发送消息
- @JmsListener (JMS)、@RabbitListener (AMQP) 注解在方法上监听消息代理发布的消息
- @EnableJms、@EnableRabbit开启支持

9. Spring Boot自动配置

- JmsAutoConfiguration
 - RabbitAutoConfiguration
- 10. 市面的MQ产品
 - ActiveMQ、RabbitMQ、RocketMQ、Kafka



RabbitMQ简介：

RabbitMQ是一个由erlang开发的AMQP(Advanced Message Queue Protocol)的开源实现。

核心概念

Message

消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括**routing-key**（路由键）、**priority**（相对于其他消息的优先权）、**delivery-mode**（指出该消息可能需要持久性存储）等。

Publisher

消息的生产者，也是一个向交换器发布消息的客户端应用程序。

Exchange

交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。

Exchange有4种类型：direct(默认)，fanout, topic, 和headers，不同类型的Exchange转发消息的策略有所区别



Queue

消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。

Binding

绑定，用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。

Exchange 和Queue的绑定可以是多对多的关系。

Connection

网络连接，比如一个TCP连接。

Channel

信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内的虚拟连接，AMQP 命令都是通过信道发出的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。



Consumer

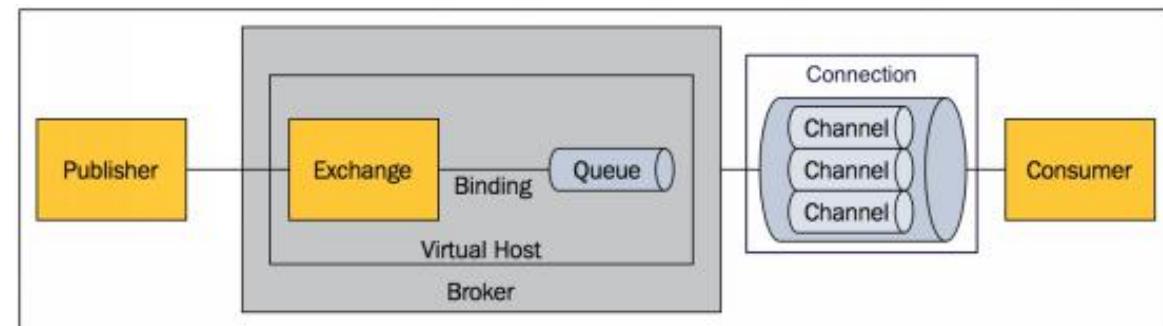
消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

Virtual Host

虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 /。

Broker

表示消息队列服务器实体





三、 Docker安装RabbitMQ



```
docker run -d --name rabbitmq -p 5671:5671 -p 5672:5672 -p 4369:4369 -p  
25672:25672 -p 15671:15671 -p 15672:15672 rabbitmq:management
```

4369, 25672 (Erlang发现&集群端口)

5672, 5671 (AMQP端口)

15672 (web管理后台端口)

61613, 61614 (STOMP协议端口)

1883, 8883 (MQTT协议端口)

<https://www.rabbitmq.com/networking.html>

- 4369: [epmd](#), a peer discovery service used by RabbitMQ nodes and CLI tools
- 5672, 5671: used by AMQP 0-9-1 and 1.0 clients without and with TLS
- 25672: used for inter-node and CLI tools communication (Erlang distribution server port) and is allocated from a dynamic range (limited to a single port by default, computed as AMQP port + 20000). Unless external connections on these ports are really necessary (e.g. the cluster uses [federation](#) or CLI tools are used on machines outside the subnet), these ports should not be publicly exposed. See [networking guide](#) for details.
- 35672-35682: used by CLI tools (Erlang distribution client ports) for communication with nodes and is allocated from a dynamic range (computed as server distribution port + 10000 through server distribution port + 10010). See [networking guide](#) for details.
- 15672: [HTTP API](#) clients, [management UI](#) and [rabbitmqadmin](#) (only if the [management plugin](#) is enabled)
- 61613, 61614: [STOMP clients](#) without and with TLS (only if the [STOMP plugin](#) is enabled)
- 1883, 8883: ([MQTT clients](#) without and with TLS, if the [MQTT plugin](#) is enabled)
- 15674: STOMP-over-WebSockets clients (only if the [Web STOMP plugin](#) is enabled)
- 15675: MQTT-over-WebSockets clients (only if the [Web MQTT plugin](#) is enabled)
- 15692: Prometheus metrics (only if the [Prometheus plugin](#) is enabled)

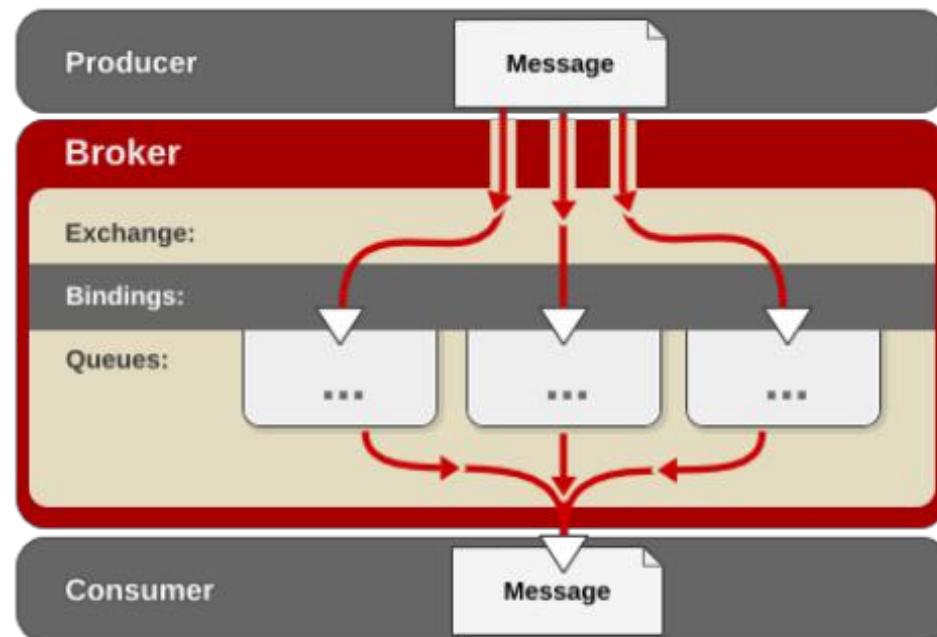


四、RabbitMQ运行机制

AMQP 中的消息路由

- AMQP 中消息的路由过程和 Java 开发者熟悉的 JMS 存在一些差别，AMQP 中增加了 **Exchange** 和 **Binding** 的角色。生产者把消息发布到 Exchange 上，消息最终到达队列并被消费者接收，而 Binding 决定交换器的消息应该发送到那个队列。

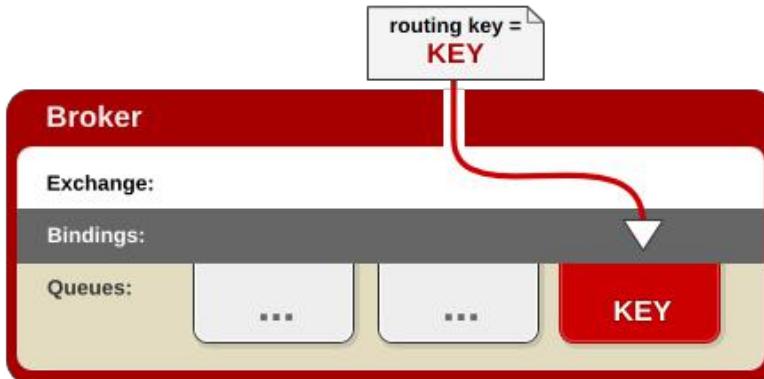
Producer Consumer





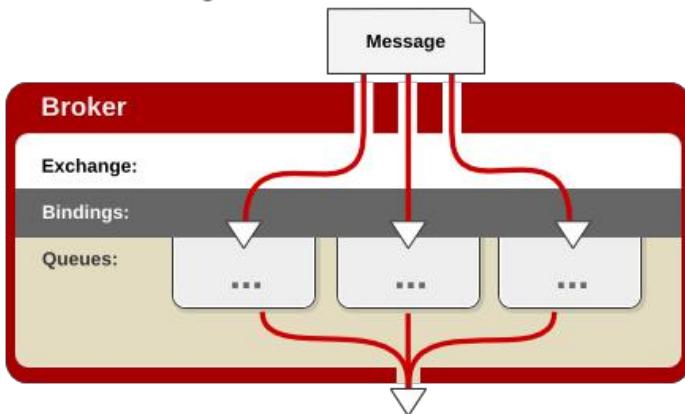
- Exchange 分发消息时根据类型的不同分发策略有区别，目前共四种类型：**direct**、**fanout**、**topic**、**headers**。headers 匹配 AMQP 消息的 header 而不是路由键，headers 交换器和 direct 交换器完全一致，但性能差很多，目前几乎用不到了，所以直接看另外三种类型：

Direct Exchange

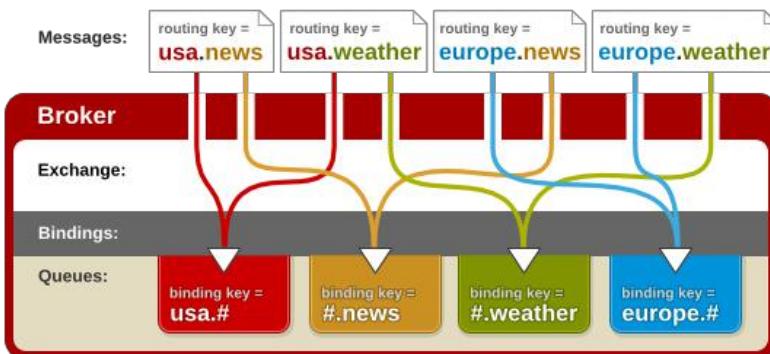


消息中的路由键 (routing key) 如果和 Binding 中的 binding key 一致，交换器就将消息发到对应的队列中。路由键与队列名完全匹配，如果一个队列绑定到交换机要求路由键为 “dog”，则只转发 routing key 标记为 “dog”的消息，不会转发 “dog.puppy”，也不会转发 “dog.guard” 等等。它是完全匹配、单播的模式。

Fanout Exchange

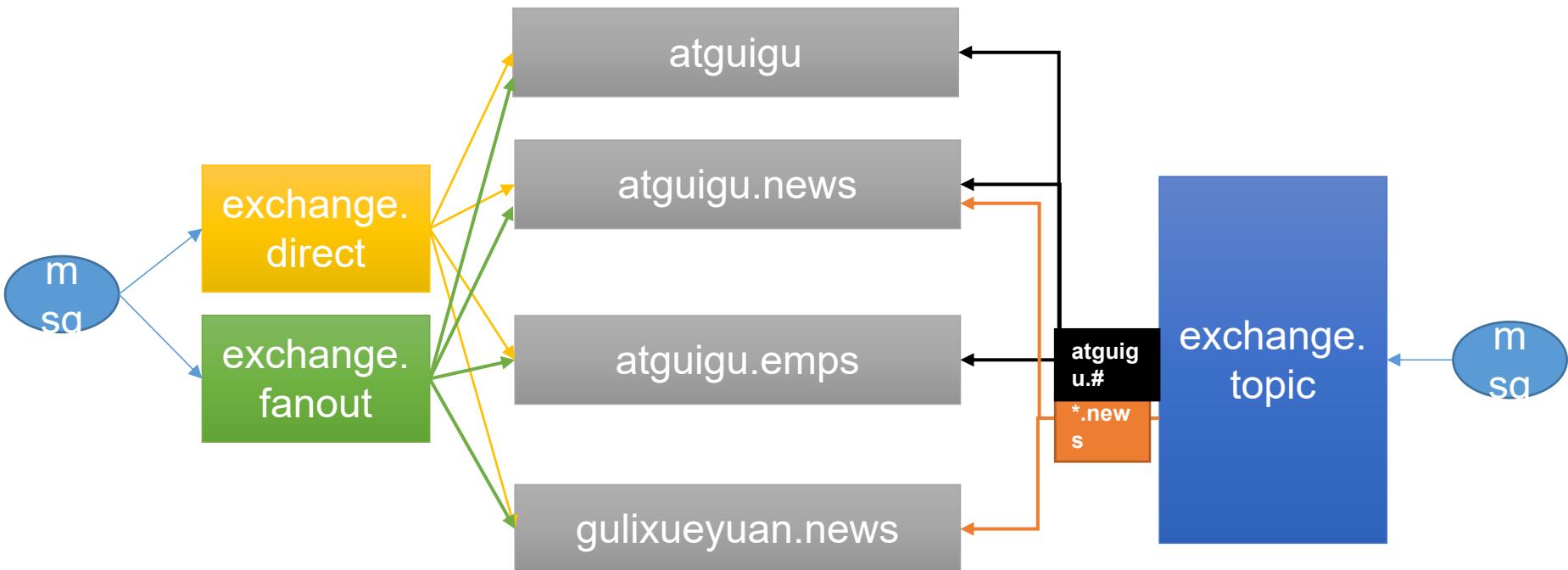


Topic Exchange



每个发到 fanout 类型交换器的消息都会分到所有绑定的队列上去。fanout 交换器不处理路由键，只是简单的将队列绑定到交换器上，每个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。
fanout 类型转发消息是最快的。

topic 交换器通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些**单词之间用点隔开**。它同样也会识别两个通配符：符号“#”和符号“*”。#匹配0个或多个单词，*匹配一个单词。



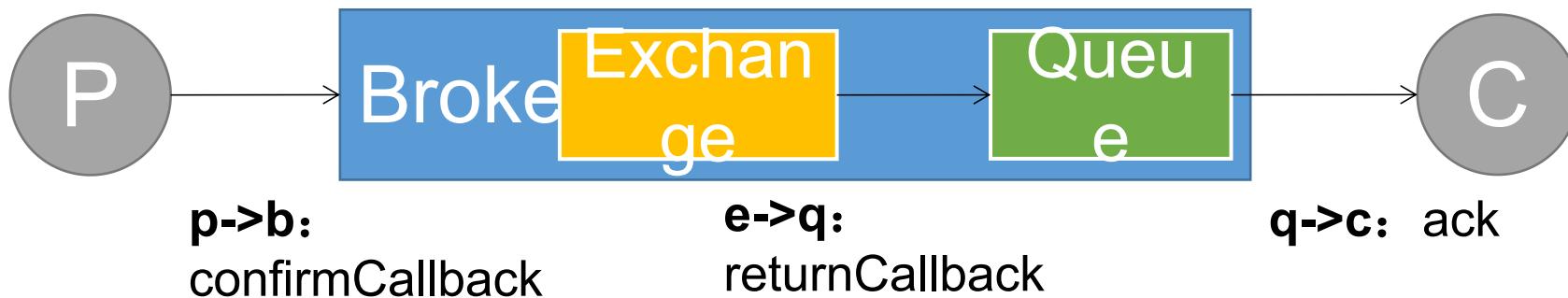


1. 引入 spring-boot-starter-amqp
2. application.yml 配置
3. 测试 RabbitMQ
 1. AmqpAdmin: 管理组件
 2. RabbitTemplate: 消息发送处理组件
 3. @RabbitListener 监听消息的方法可以有三种参数 (不分数量, 顺序)
 - Object content, Message message, Channel channel



六、RabbitMQ消息确认机制-可靠抵达

- 保证消息不丢失，可靠抵达，可以使用事务消息，性能下降250倍，为此引入确认机制
- publisher confirmCallback** 确认模式
- publisher returnCallback** 未投递到 queue 退回模式
- consumer ack**机制





- `spring.rabbitmq.publisher-confirms=true`
 - 在创建 `connectionFactory` 的时候设置 `PublisherConfirms(true)` 选项，开启 `confirmcallback`。
 - `CorrelationData`: 用来表示当前消息唯一性。
 - 消息只要被 `broker` 接收到就会执行 `confirmCallback`，如果是 `cluster` 模式，需要所有 `broker` 接收到才会调用 `confirmCallback`。
 - 被 `broker` 接收到只能表示 `message` 已经到达服务器，并不能保证消息一定会被投递到目标 `queue` 里。所以需要用到接下来的 `returnCallback`。



- `spring.rabbitmq.publisher-returns=true`
- `spring.rabbitmq.template.mandatory=true`
 - confirm 模式只能保证消息到达 broker，不能保证消息准确投递到目标 queue 里。在有些业务场景下，我们需要保证消息一定要投递到目标 queue 里，此时就需要用到 return 退回模式。
 - 这样如果未能投递到目标 queue 里将调用 returnCallback，可以记录下详细到投递数据，定期的巡检或者自动纠错都需要这些数据。



- 消费者获取到消息，成功处理，可以回复Ack给Broker
 - basic.ack用于肯定确认； broker将移除此消息
 - basic.nack用于否定确认； 可以指定broker是否丢弃此消息，可以批量
 - basic.reject用于否定确认； 同上，但不能批量
- 默认自动ack，消息被消费者收到，就会从broker的queue中移除
- queue无消费者，消息依然会被存储，直到消费者消费
- 消费者收到消息，默认会自动ack。但是如果无法确定此消息是否被处理完成，或者成功处理。我们可以开启手动ack模式
 - 消息处理成功， ack()，接受下一个消息，此消息broker就会移除
 - 消息处理失败， nack()/reject()，重新发送给其他人进行处理，或者容错处理后ack
 - 消息一直没有调用ack/nack方法， broker认为此消息正在被处理，不会投递给别人，此时客户端断开，消息不会被broker移除，会投递给别人



场景：

比如未付款订单，超过一定时间后，系统自动取消订单并释放占有物品。

常用解决方案：

spring的 schedule 定时任务轮询数据库

缺点：

消耗系统内存、增加了数据库的压力、存在较大的时间误差

解决：rabbitmq的消息TTL和死信Exchange结合



- 消息的TTL就是消息的存活时间。
- RabbitMQ可以对队列和消息分别设置TTL。
 - 对队列设置就是队列没有消费者连着的保留时间，也可以对每一个单独的消息做单独的设置。超过了这个时间，我们认为这个消息就死了，称之为死信。
 - 如果队列设置了，消息也设置了，那么会取小的。所以一个消息如果被路由到不同的队列中，这个消息死亡的时间有可能不一样（不同的队列设置）。这里单讲单个消息的TTL，因为它才是实现延迟任务的关键。可以通过设置消息的expiration字段或者x-message-ttl属性来设置时间，两者是一样的效果。



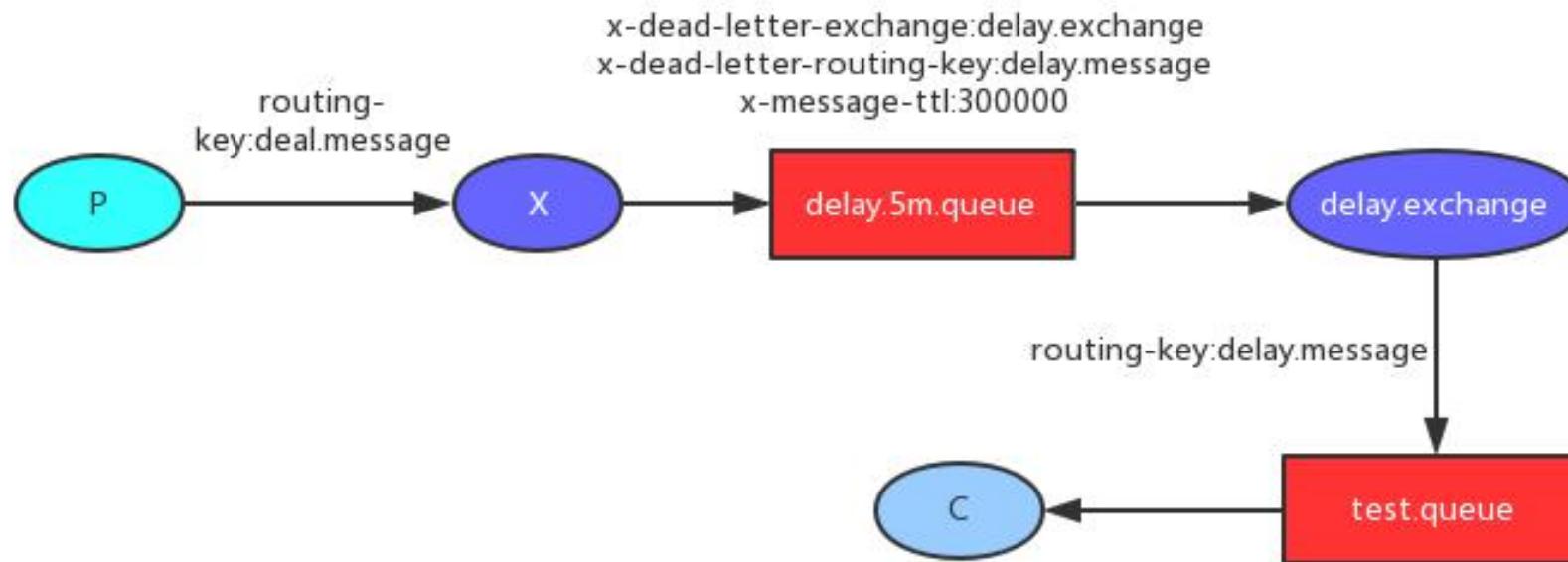
- 一个消息在满足如下条件下，会进死信路由，记住这里是路由而不是队列，一个路由可以对应很多队列。（什么是死信）
 - 一个消息被Consumer拒收了，并且reject方法的参数里requeue是false。也就是说不会被再次放在队列里，被其他消费者使用。`(basic.reject/ basic.nack) requeue=false`
 - 上面的消息的TTL到了，消息过期了。
 - 队列的长度限制满了。排在前面的消息会被丢弃或者扔到死信路由上
- Dead Letter Exchange其实就是一种普通的exchange，和创建其他exchange没有两样。只是在某一个设置Dead Letter Exchange的队列中有消息过期了，会自动触发消息的转发，发送到Dead Letter Exchange中去。
- 我们既可以控制消息在一段时间后变成死信，又可以控制变成死信的消息被路由到某一个指定的交换机，结合二者，其实就可以实现一个延时队列

- 手动ack&异常消息统一放在一个队列处理建议的两种方式
 - catch异常后，**手动发送到指定队列**，然后使用channel给rabbitmq确认消息已消费
 - 给Queue绑定死信队列，使用nack（requeue为false）确认消息消费失败



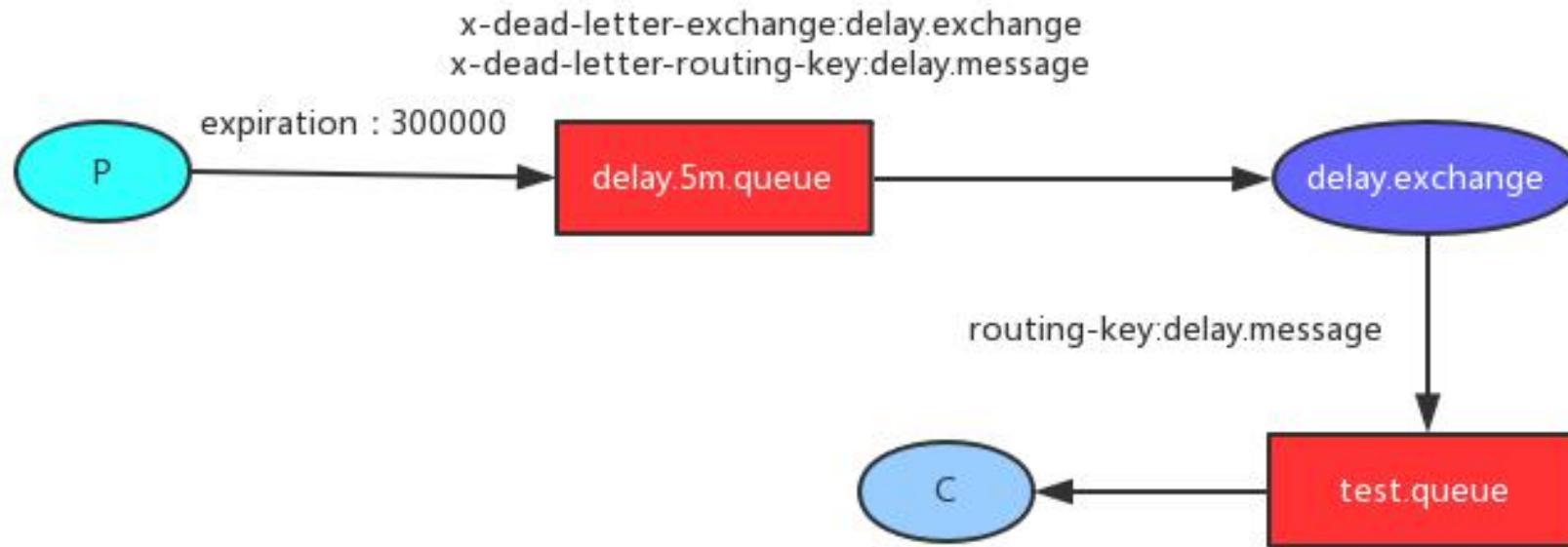


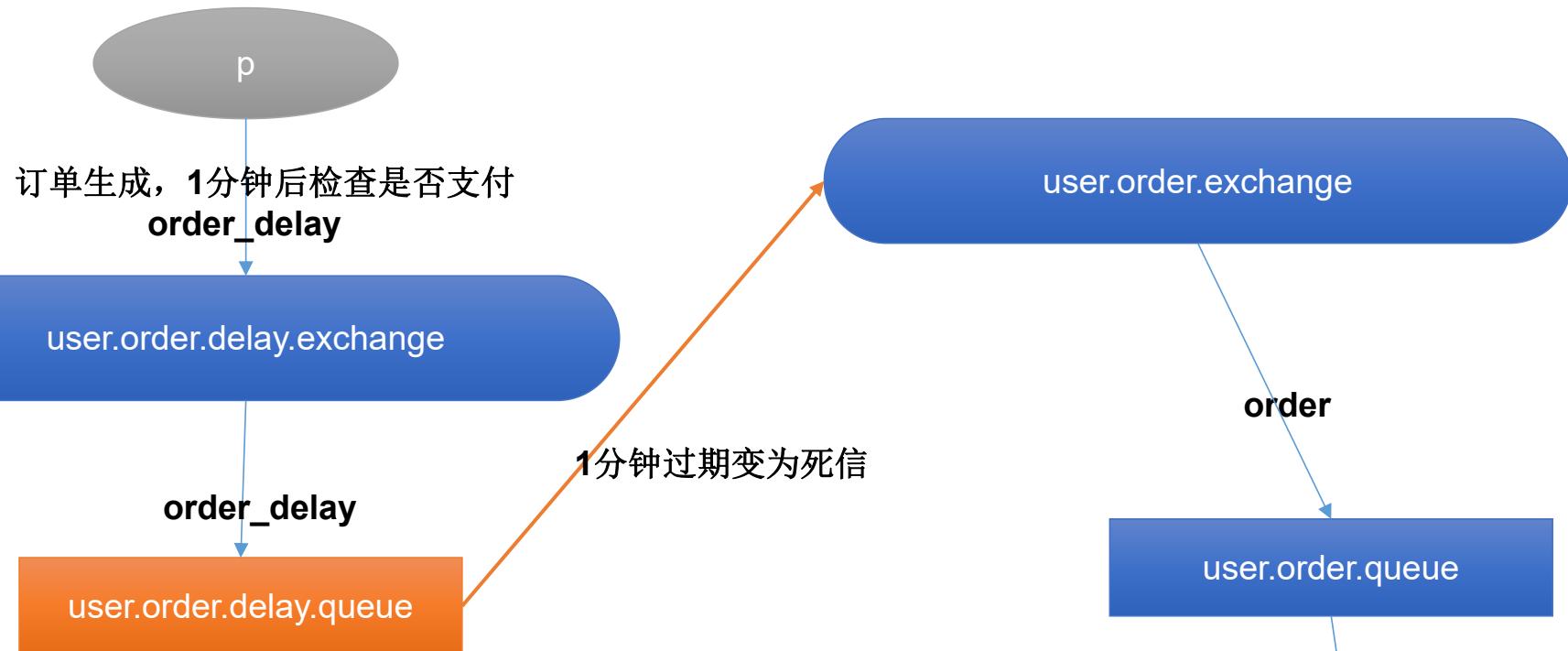
设置队列过期时间实现延时队列



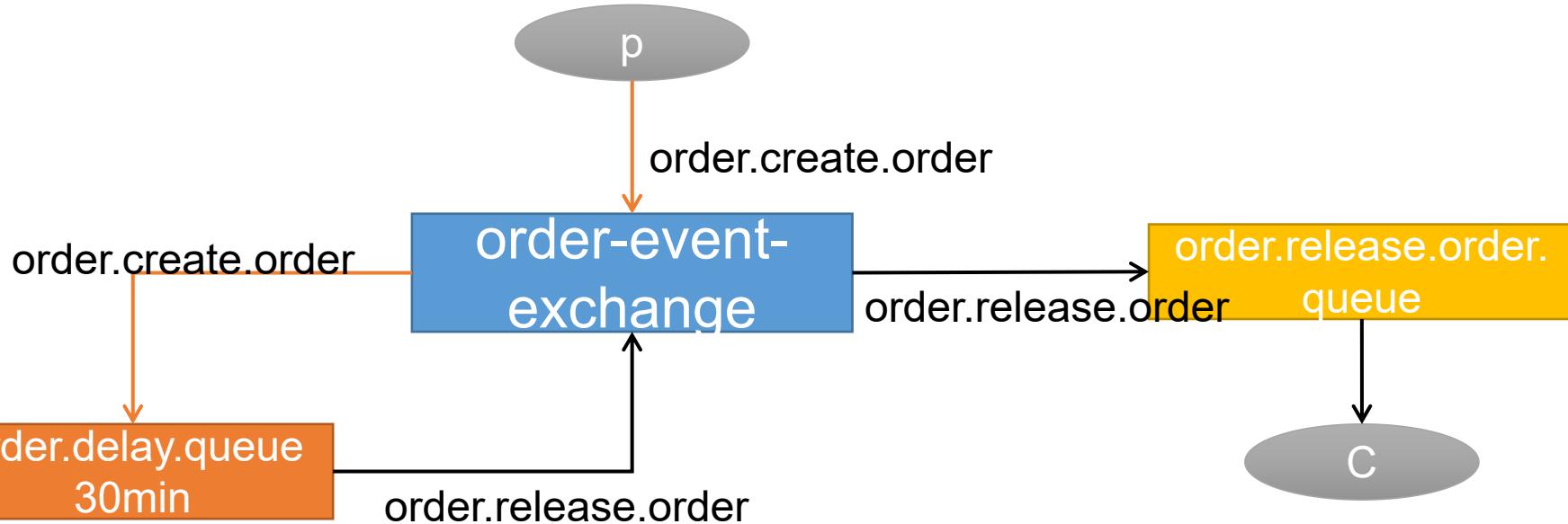


设置消息过期时间实现延时队列





x-dead-letter-exchange: `user.order.exchange`
x-dead-letter-routing-key: `order`
x-message-ttl: 60000



x-dead-letter-exchange: order-event-exchange

x-dead-letter-routing-key: order.release.order

x-message-ttl: 60000



- 1、Queue、Exchange、Binding可以@Bean进去
- 2、监听消息的方法可以有三种参数（不分数量，顺序）
 - Object content, Message message, Channel channel
- 3、channel可以用来拒绝消息，否则自动ack；



- 1、消息丢失
 - 消息发送出去，由于网络问题没有抵达服务器
 - 做好容错方法（try-catch），发送消息可能会网络失败，失败后要有重试机制，可记录到数据库，采用定期扫描重发的方式
 - 做好日志记录，每个消息状态是否都被服务器收到都应该记录
 - 做好定期重发，如果消息没有发送成功，定期去数据库扫描未成功的消息进行重发
 - 消息抵达Broker，Broker要将消息写入磁盘（持久化）才算成功。此时Broker尚未持久化完成，宕机。
 - publisher也必须加入确认回调机制，确认成功的消息，修改数据库消息状态。
 - 自动ACK的状态下。消费者收到消息，但没来得及消息然后宕机
 - 一定开启手动ACK，消费成功才移除，失败或者没来得及处理就noAck并重新入队



• 2、消息重复

- 消息消费成功，事务已经提交，ack时，机器宕机。导致没有ack成功，Broker的消息重新由unack变为ready，并发送给其他消费者
- 消息消费失败，由于重试机制，自动又将消息发送出去
- 成功消费，ack时宕机，消息由unack变为ready，Broker又重新发送
 - 消费者的业务消费接口应该设计为幂等性的。比如扣库存有工作单的状态标志
 - 使用**防重表**（redis/mysql），发送消息每一个都有业务的唯一标识，处理过就不用处理
 - rabbitMQ的每一个消息都有redelivered字段，可以获取**是否是被重新投递过来的**，而不是第一次投递过来的



- 3、消息积压

- 消费者宕机积压
- 消费者消费能力不足积压
- 发送者发送流量太大
 - 上线更多的消费者，进行正常消费
 - 上线专门的队列消费服务，将消息先批量取出来，记录数据库，离线慢慢处理



- ActiveMQ、RabbitMQ、RocketMQ、Kafka

谢谢观看



Gulimall

支付



一、支付宝支付

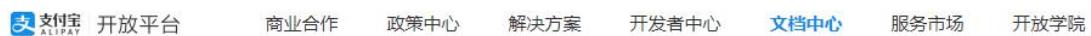
1、进入“蚂蚁金服开放平台”

<https://open.alipay.com/platform/home.htm>

2、下载支付宝官方 demo，进行配置和测试

文档地址

<https://open.alipay.com/platform/home.htm> 支付宝&蚂蚁金服开发者平台

支付宝 开放平台 商业合作 政策中心 解决方案 开发者中心 文档中心 服务市场 开放学院

<https://docs.open.alipay.com/catalog> 开发者文档

<https://docs.open.alipay.com/270/106291/> 全部文档=>电脑网站支付文档；下载 demo

快速入门	API 文档	产品文档	开发工具	开发服务
Step 1 平台概述	支付 API	花呗分期	SDK & DEMO	云监控
Step 2 平台入驻	会员 API	当面付	沙箱环境	安全监测
Step 3 创建应用	店铺 API	App 支付	签名工具	凤蝶 H5
Step 4 配置密钥	营销 API	手机网站支付	小程序开发者工具	小程序云服务
Step 5 上线应用	小程序 API	电脑网站支付	IDEA 插件	语雀云端知识库

3、配置使用沙箱进行测试

- 
- 1、使用 RSA 工具生成签名
 - 2、下载沙箱版钱包
 - 3、运行官方 demo 进行测试

4、什么是公钥、私钥、加密、签名和验签？

1、公钥私钥

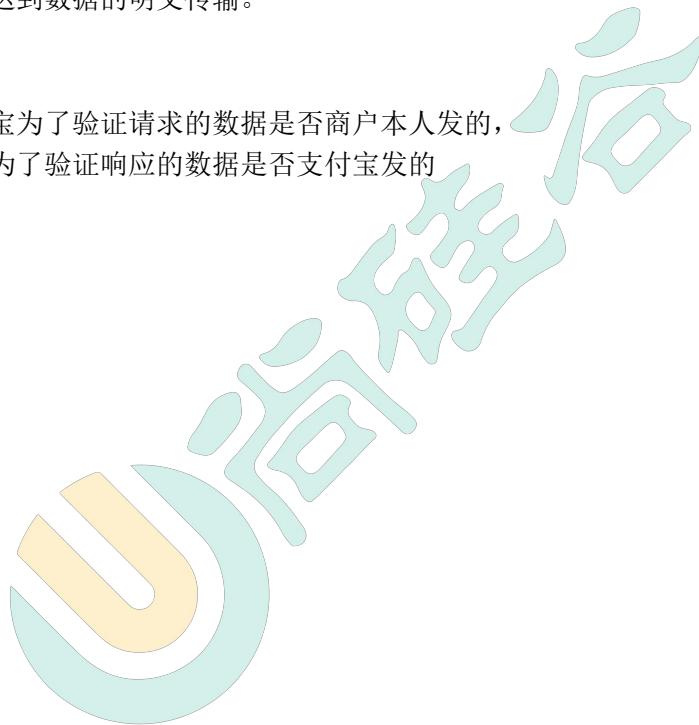
公钥和私钥是一个相对概念

它们的公私性是相对于生成者来说的。

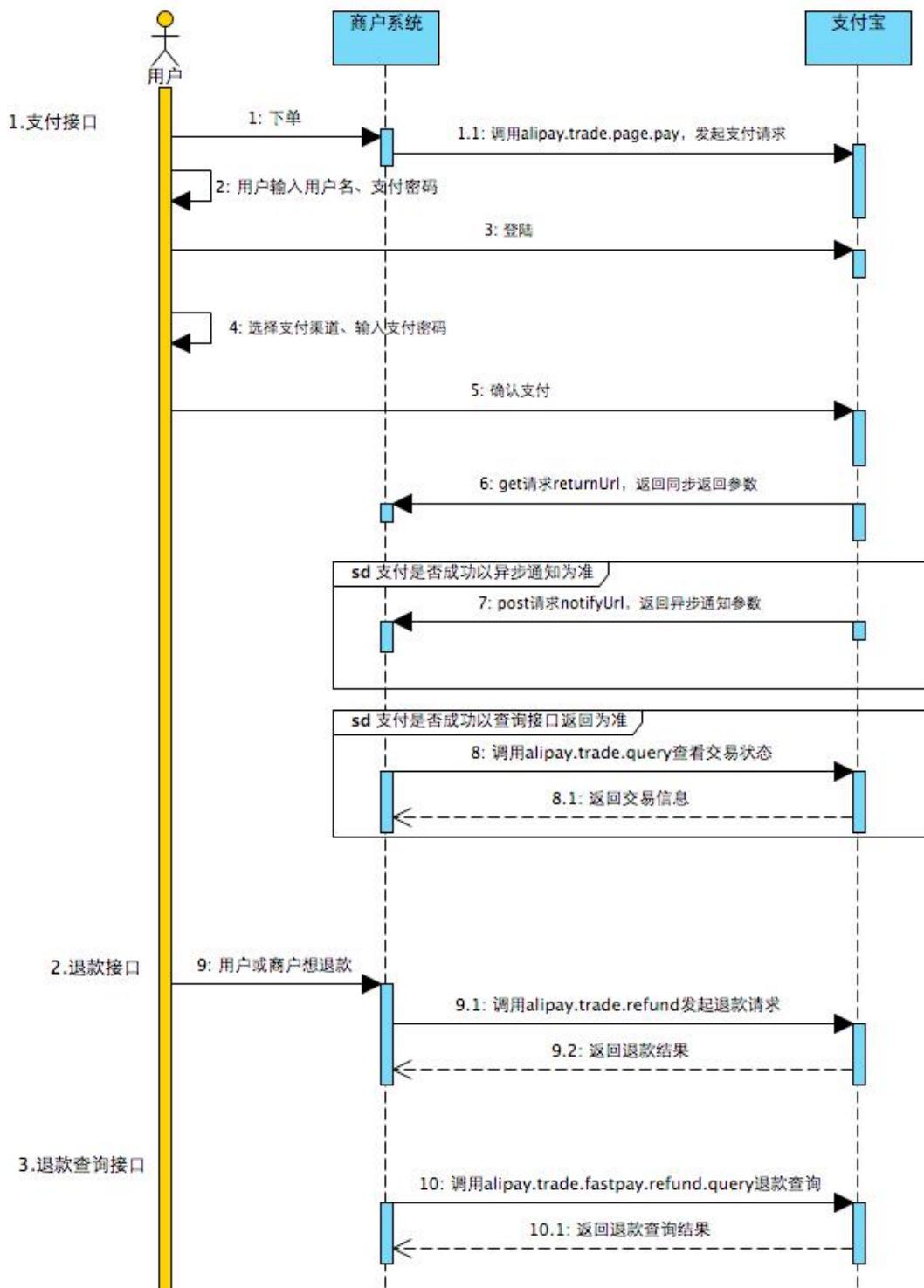
一对密钥生成后，保存在生成者手里的就是私钥，
生成者发布出去大家用的就是公钥

2、加密和数字签名

- 加密是指：
 - 我们使用一对公私钥中的一个密钥来对数据进行加密，而使用另一个密钥来进行解密的技术。
 - 公钥和私钥都可以用来加密，也都可以用来解密。
 - 但这个加解密必须是一对密钥之间的互相加解密，否则不能成功。
 - 加密的目的是：
 - ◆ 为了确保数据传输过程中的不可读性，就是不想让别人看到。
- 签名：
 - 给我们将要发送的数据，做一个唯一签名（类似于指纹）
 - 用来互相验证接收方和发送方的身份；
 - 在验证身份的基础上再验证一下传递的数据是否被篡改过。因此使用数字签名可以用来达到数据的明文传输。
- 验签
 - 支付宝为了验证请求的数据是否商户本人发的，
 - 商户为了验证响应的数据是否支付宝发的



5、支付宝支付流程



二、内网穿透

1、简介

内网穿透功能可以允许我们使用外网的网址来访问主机；

正常的外网需要访问我们项目的流程是：

- 1、买服务器并且有公网固定 IP
- 2、买域名映射到服务器的 IP
- 3、域名需要进行备案和审核

2、使用场景

- 1、开发测试（微信、支付宝）
- 2、智慧互联
- 3、远程控制
- 4、私有云

3、内网穿透的几个常用软件

- 1、natapp: <https://natapp.cn/> 优惠码: 022B93FD (9 折) [仅限第一次使用]
- 2、续断: www.zhexi.tech 优惠码: SBQMEA (95 折) [仅限第一次使用]
- 3、花生壳: <https://www.oray.com/>

Gulimall

定时任务



一、定时任务

1、cron 表达式

语法：秒 分时 日月周 年 (Spring 不支持)

<http://www.quartz-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html>

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	YES	0-59	, - * /
Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of month	YES	1-31	, - * ? / L W
Month	YES	1-12 or JAN-DEC	, - * /
Day of week	YES	1-7 or SUN-SAT	, - * ? / L #
Year	NO	empty, 1970-2099	, - * /

特殊字符：

， : 枚举；

(cron="7,9,23 * * * * ?")：任意时刻的 7,9, 23 秒启动这个任务；

-：范围：

(cron="7-20 * * * * ?")：任意时刻的 7-20 秒之间，每秒启动一次

*：任意；

指定位置的任意时刻都可以

/：步长；

(cron="7/5 * * * * ?")：第 7 秒启动，每 5 秒一次；

(cron="*/5 * * * * ?")：任意秒启动，每 5 秒一次；

?：（出现在日和周几的位置）：为了防止日和周冲突，在周和日上如果要写通配符使用？

(cron="* * * 1 * ?")：每月的 1 号，启动这个任务；

L：（出现在日和周的位置）”，

last：最后一个

(cron="* * * ? * 3L")：每月的最后一个周二

W：

Work Day：工作日

(cron="* * * W * ?")：每个月的工作日触发

(cron="* * * LW * ?")：每个月的最后一个工作日触发

#：第几个

(cron="* * * ? * 5#2")：每个月的第 2 个周 4

2、cron 示例

Expression	**Meaning**
0 0 12 * * ?	Fire at 12pm (noon) every day
0 15 10 ? * *	Fire at 10:15am every day
0 15 10 * * ?	Fire at 10:15am every day
0 15 10 * * ? *	Fire at 10:15am every day
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month
0 15 10 L * ?	Fire at 10:15am on the last day of every month
0 15 10 L-2 * ?	Fire at 10:15am on the 2nd-to-last last day of every month
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.



让天下没有难学的技术

0 11 11 11 11 ?

Fire every November 11th at 11:11am.

3、SpringBoot 整合

@EnableScheduling

@Scheduled



二、分布式定时任务

1、定时任务问题

1)、同时执行导致的重复

由于同样的服务会部署多个节点，多个节点的定时任务代码可能同时启动。将同样的事情做了多次

使用分布式锁。

2)、任务拆分并发执行

使用 ElasticJob

2、扩展-分布式调度

<http://elasticjob.io/docs/elastic-job-cloud/00-overview/>

谷粒商城

WebFlux



一、Reactive Programming（响应式编程）

响应式编程(reactive programming)是一种基于数据流(data stream)和变化传递(propagation of change)的声明式(declarative)的编程范式。

推荐博文：<https://blog.51cto.com/liukang/2090163>

1、变化传递 (propagation of change)

举个简单的例子，某电商网站正在搞促销活动，任何单品都可以参加“满 199 减 40”的活动，而且“满 500 包邮”。

The diagram illustrates the propagation of changes from a shopping cart to payment details. It shows two tables: a shopping cart table and a payment table, with arrows indicating how changes in one table affect the other.

购物车					付款		
商品	单价	数量	商品金额	满199减40	订单总金额	邮费(满500包邮)	最终应付款
饼干	8.50	5	42.50	42.50	521.20	0.00	521.20
干果	39.90	2	79.80	79.80			
玉米油	42.90	1	42.90	42.90			
牛肉干	59.00	1	59.00	59.00			
长粒香米	39.00	2	78.00	78.00			
剃须刀	259.00	1	259.00	219.00			

B	C	D	E	F	G	H	I
购物车				付款			
商品	单价	数量	商品金额	满199减40	订单总金额	邮费(满500包邮)	最终应付款
饼干	8.5	5	=C3*D3	=IF(E3>199,E3-40,E3)	=SUM(F:F)	=IF(I2>500,0,50)	
干果	39.9	2	=C4*D4	=IF(E4>199,E4-40,E4)			
玉米油	42.9	1	=C5*D5	=IF(E5>199,E5-40,E5)			
牛肉干	59	1	=C6*D6	=IF(E6>199,E6-40,E6)			
长粒香米	39	2	=C7*D7	=IF(E7>199,E7-40,E7)			
剃须刀	259	1	=C8*D8	=IF(E8>199,E8-40,E8)			

“商品金额”是通过“单价 x 数量”得到的，“满 199 减 40”会判断该商品金额是否满 199 并根据情况减掉 40，右侧“订单总金额”是“满 199 减 40”这一列的和，“邮费”会根据订单总金额计算，“最终应付款”就是订单总金额加上邮费。

响应式的核心特点之一：变化传递 (propagation of change)。一个单元格变化之后，会像多米诺骨牌一样，导致直接和间接引用它的其他单元格均发生相应变化。



生产者只负责生成并发出数据/事件，消费者来监听并负责定义如何处理数据/事件的变化传

递方式。

2、数据流 (data stream)

数据/事件在响应式编程里会以数据流的形式发出。

小明选购商品的过程，为了既不超预算，又能省邮费，有时加有时减



这一次一次的操作就构成了一串数据流，如果我们能够及时对数据流的每一个事件做出响应，会有效提高系统的响应水平。这是响应式的另一个核心特点：**基于数据流 (data stream)**。

```
public Invoice(Cart cart) {  
    ...  
    this.listenOn(cart.eventStream()); // 1  
    ...  
}
```

cart.eventStream()是要监听的购物车的操作事件数据流，listenOn 方法能够对数据流中到来的元素依次进行处理。

3、声明式 (declarative)

我们再到 listenOn 方法去看一下：

Invoice 模块中，上边的一串公式被组装成如下的伪代码：

```
public void listenOn(DataStream<CartEvent> cartEventStream) {  
    double sum = 0;  
    double total = cartEventStream  
        // 分别计算商品金额  
        .map(cartEvent -> cartEvent.getProduct().getPrice() * cartEvent.getQuantity())  
        // 计算满减后的商品金额  
        .map(v -> (v > 199) ? (v - 40) : v)  
        // 将金额的变化累加到 sum  
        .map(v -> {sum += v; return sum;})  
        // 根据 sum 判断是否免邮，得到最终总付款金额  
        .map(sum -> (sum > 500) ? sum : (sum + 50));  
    ...
```

这是一种“**声明式 (declarative)**”的编程范式。通过四个串起来的 map 调用，我们先声明好了对于数据流“将会”进行什么样的处理，当有数据流过来时，就会按照声明好的处理流程逐个进行处理。



命令式是面向过程的，声明式是面向结构的。

不过命令式和声明式本身并无高低之分，只是声明式比较适合基于流的处理方式。这是响应式的第三个核心特点：**声明式（declarative）**。结合“变化传递”的特点，声明式能够让基于数据流的开发更加友好。

再举个简单的例子方便理解：

```
a = 1;
b = a + 1;
a = 2;
```

这个时候，**b** 是多少呢？在 Java 以及多数语言中，**b** 的结果是 2，第二次对 **a** 的赋值并不会影响 **b** 的值。

假设 Java 引入了一种新的赋值方式 **:=**，表示一种对 **a** 的绑定关系，如

```
a = 1;
b := a + 1;
a = 2;
```

由于 **b** 保存的不是某次计算的值，而是针对 **a** 的一种绑定关系，所以 **b** 能够随时根据 **a** 的值的变化而变化

4、总结

响应式编程的“变化传递”就相当于果汁流水线的管道；在入口放进橙子，出来的就是橙汁；放西瓜，出来的就是西瓜汁，橙子和西瓜、以及机器中的果肉果汁以及残渣等，都是流动的“数据流”；管道的图纸是用“声明式”的语言表示的。

这种编程范式如何让 Web 应用更加“reactive”呢？

我们设想这样一种场景，我们从底层数据库驱动，经过持久层、服务层、MVC 层中的 model，到用户的前端界面的元素，全部都采用声明式的编程范式，从而搭建一条能够传递变化的管道，这样我们只要更新一下数据库中的数据，用户的界面上就相应地发生变化，岂不美哉？尤其重要的是，一处发生变化，我们不需要各种命令式的调用来传递这种变化，而是由搭建好的“流水线”自动传递。

这种场景用在哪呢？比如一个日志监控系统，我们的前端页面将不再需要通过“命令式”的轮询的方式不断向服务器请求数据然后进行更新，而是在建立好通道之后，数据流从系统源不断流向页面，从而展现实时的指标变化曲线；再比如一个社交平台，朋友的动态、点赞和留言不是手动刷出来的，而是当后台数据变化的时候自动体现到界面上的。

二、Reactive Stream（响应式流）

为啥不用 Java Stream 来进行数据流的操作？

- Web 应用具有 I/O 密集的特点，I/O 阻塞会带来比较大的性能损失或资源浪费，我们需要一种异步非阻塞的响应式的库，而 Java Stream 是一种同步 API。
- 假设我们要搭建从数据层到前端的一个变化传递管道，可能会遇到数据层每秒上千次的数据更新，而显然不需要向前端传递每一次更新，这时候就需要一种流量控制能力，就像我们家里的水龙头，可以控制开关流速，而 Java Stream 不具备完善的对数据流的流量控制的能力。

具备“异步非阻塞”特性和“流量控制”能力的数据流，我们称之为响应式流（Reactive Stream）。

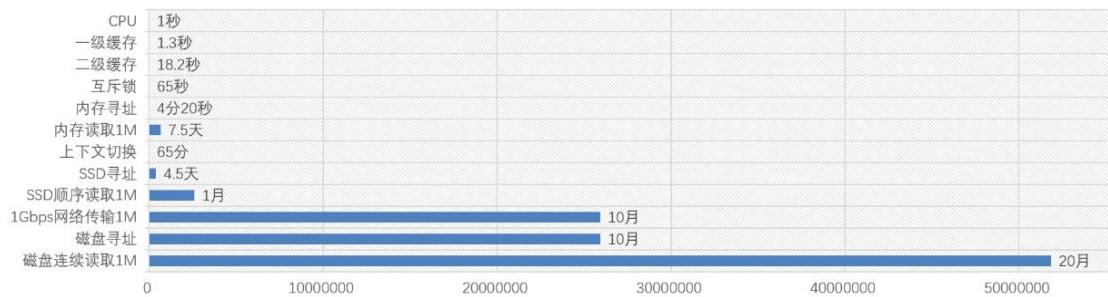
目前有几个实现了响应式流规范的 Java 库，这里简单介绍两个：RxJava 和 Reactor。

Reactor 和 Sprig 都是同一个公司 Pivotal 旗下的项目。也是 Spring5 响应式编程的底层框架在 Java 9 版本中，响应式流的规范被纳入到了 JDK 中，相应的 API 接口是 java.util.concurrent.Flow。

1、阻塞、非阻塞以及同步、异步

- 阻塞和非阻塞反映的是调用者的状态，当调用者调用了服务提供者的方法后，如果一直在等待结果返回，否则无法执行后续的操作，那就是阻塞状态；如果调用之后直接返回，从而可以继续执行后续的操作，那可以理解为非阻塞的。
- 同步和异步反映的是服务提供者的能力，当调用者调用了服务提供者的方法后，如果服务提供者能够立马返回，并在处理完成后通过某种方式通知到调用者，那可以理解为异步的；否则，如果只是在处理完成后才返回，或者需要调用者再去主动查询处理是否完成，就可以理解为是同步的。

互联网时代的大背景下，Web 应用通常要面对高并发、海量数据的挑战，性能从来都是必须要考量的核心因素。阻塞便是性能杀手之一。



对于阻塞造成的性能损失，我们通常有两种思路来解决：

- **并行化**: 使用更多的线程和硬件资源;
 - “多线程并非银弹”，存在一些固有的弊端，但是多线程在高并发方面发挥了重要作用。况且，多线程仍然是目前主流的高并发方案。
 - 高并发环境下，多线程的切换会消耗 CPU 资源
 - 应对高并发环境的多线程开发相对比较难（需要掌握线程同步的原理与工具、ExecutorService、Fork/Join 框架、并发集合和原子类等的使用），并且有些问题难以发现或重现（比如指令重排）；
 - 高并发环境下，更多的线程意味着更多的内存占用（JVM 默认为每个线程分配 1M 的线程栈空间）
- **异步化**: 基于现有的资源来提高执行效率。
 - 异步非阻塞
 - ◆ 回调。如 ajax 的 callback
 - ◆ 异步的 CompletableFuture。

2、流量控制—回压

在响应式流中，数据流的发出者叫做 Publisher，监听者叫做 Subscriber。“发布者”和“订阅者”。



问题来了，假如发布者发出数据的速度和订阅者处理数据的速度不同的时候，怎么办呢？订阅者处理速度快的话，那还说，但是如果处理速度跟不上数据发出的速度



如果没有流量控制，那么订阅者会被发布者快速产生的数据流淹没。就像在一个流水线上，

如果某个工位处理比较慢，而上游下料比较快的话，这个工位的工人师傅就吃不消了，这个时候他需要一种途径来告诉上游下料慢一些。

同样的，订阅者也需要有一种能够向上游反馈流量需求的机制：

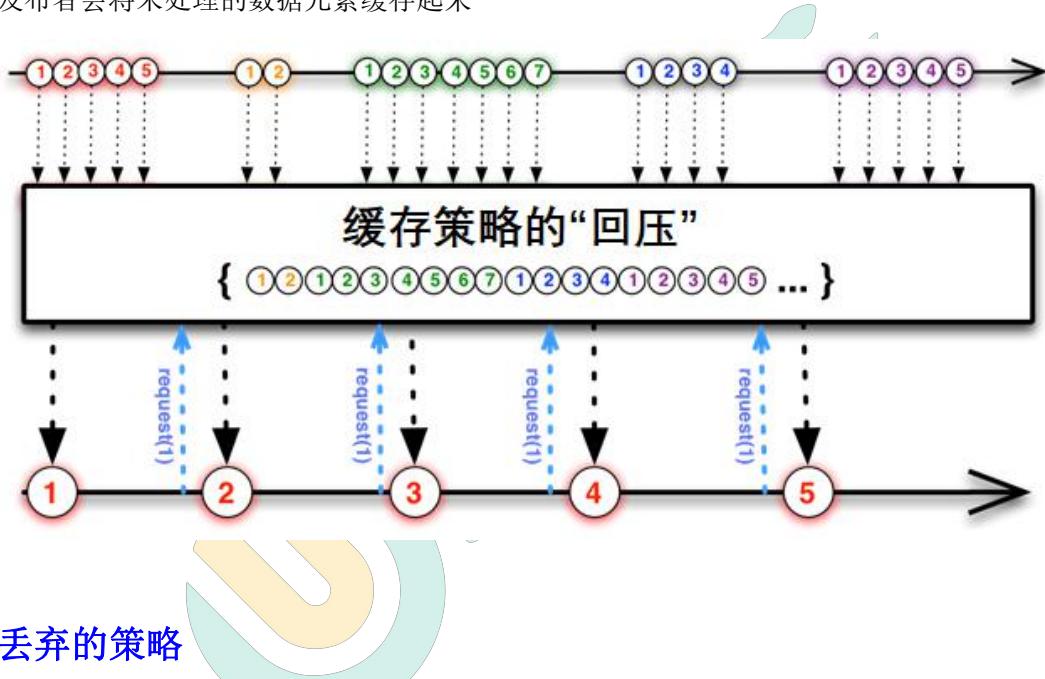


这种能够向上游反馈流量请求的机制就叫做回压（backpressure，也有翻译为“背压”的）。

在具体的使用过程中，回压的处理会涉及不同的策略。

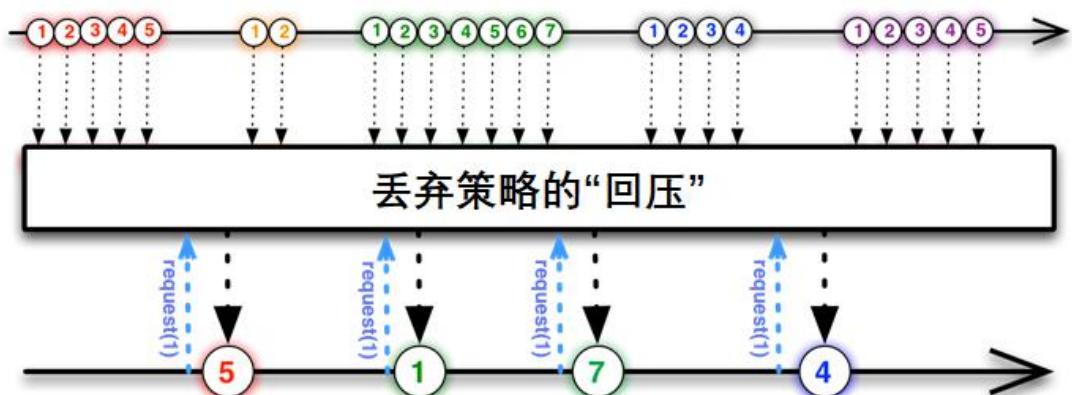
缓存的策略

发布者会将未处理的数据元素缓存起来



丢弃的策略

发布者不需要缓存来不及处理的数据，而是直接丢弃，当订阅者请求数据的时候，会拿到发布者那里最近的一个数据元素



响应式流的两个核心特点：异步非阻塞，以及基于“回压”机制的流量控制。

响应式流 异步非阻塞 背压机制 + 变化传递 + 声明式范式

三、Reactor

推荐阅读：

<https://www.ibm.com/developerworks/cn/java/j-cn-with-reactor-response-encode/index.html>

1、Mono 与 Flux

1)、数据流三种信号

Reactor 两个核心概念 Mono、Flux。

Reactor 中的发布者（Publisher）由 Flux 和 Mono 两个类定义

既然是“数据流”的发布者，Flux 和 Mono 都可以发出三种“数据信号”：

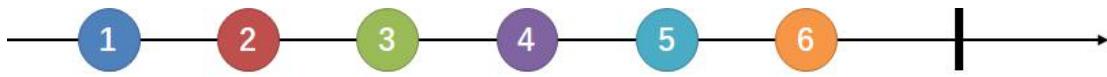
- 元素值
- 错误信号
- 完成信号，

错误信号和完成信号都是终止信号，完成信号用于告知下游订阅者该数据流正常结束，错误信号终止数据流的同时将错误传递给下游订阅者。当消息通知产生时，订阅者中对应的方法 `onNext()`, `onComplete()` 和 `onError()` 会被调用。

Flux 表示的是包含 0 到 N 个元素的异步序列。

Mono 表示的是包含 0 或者 1 个元素的异步序列。

下图所示就是一个 Flux 类型的数据流，黑色箭头是时间轴。它连续发出“1” - “6”共 6 个元素值，以及一个完成信号（图中⑥后边的加粗竖线来表示），完成信号告知订阅者数据流已经结束。



下图所示是一个 Mono 类型的数据流，它发出一个元素值后，又发出一个完成信号。



上两图表示为如下代码

```
Flux.just(1, 2, 3, 4, 5, 6);
```

```
Mono.just(1);
```

Flux 和 Mono 提供了多种创建数据流的方法，just 就是一种比较直接的声明数据流的方式，

其参数就是数据元素。

```
● $ fromArray(T[]) <T> : Flux<T>
● $ fromIterable(Iterable<? extends T>) <T> : Flux<T>
● $ fromStream(Stream<? extends T>) <T> : Flux<T>
```

```
Integer[] array = new Integer[]{1,2,3,4,5,6};
Flux.fromArray(array);
List<Integer> list = Arrays.asList(array);
Flux.fromIterable(list);
Stream<Integer> stream = list.stream();
Flux.fromStream(stream);
```

不过，这三种信号都不是一定要具备的：

- 首先，错误信号和完成信号都是终止信号，二者不可能同时共存；
- 如果没有发出任何一个元素值，而是直接发出完成/错误信号，表示这是一个空数据流；
- 如果没有错误信号和完成信号，那么就是一个无限数据流。

2)、创建 Flux、Mono

1、通过 Flux 类的静态方法，快速创建

- `just()`: 可以指定序列中包含的全部元素。创建出来的 Flux 序列在发布这些元素之后会自动结束。
- `fromArray()`, `fromIterable()` 和 `fromStream()`: 可以从一个数组、Iterable 对象或 Stream 对象中创建 Flux 对象。
- `empty()`: 创建一个不包含任何元素，只发布结束消息的序列。
- `error(Throwable error)`: 创建一个只包含错误消息的序列。
- `never()`: 创建一个不包含任何消息通知的序列。
- `range(int start, int count)`: 创建包含从 `start` 起始的 `count` 个数量的 Integer 对象的序列。
- `interval(Duration period)` 和 `interval(Duration delay, Duration period)`: 创建一个包含了从 0 开始递增的 Long 对象的序列。其中包含的元素按照指定的间隔来发布。除了间隔时间之外，还可以指定起始元素发布之前的延迟时间。
- `intervalMillis(long period)` 和 `intervalMillis(long delay, long period)`: 与 `interval()` 方法的作用相同，只不过该方法通过毫秒数来指定时间间隔和延迟时间。

```
Flux.just("Hello", "World").subscribe(System.out::println);
Flux.fromArray(new Integer[] {1, 2, 3}).subscribe(System.out::println);
Flux.empty().subscribe(System.out::println);
Flux.range(1, 10).subscribe(System.out::println);
Flux.interval(Duration.of(10, ChronoUnit.SECONDS)).subscribe(System.out::println);
Flux.intervalMillis(1000).subscribe(System.out::println);
```

2、 generate() 或 create()

generate() 序列的产生是通过调用所提供的 SynchronousSink 对象的 next(), complete() 和 error(Throwable) 方法来完成的。

generate() 只提供序列中单个消息的产生逻辑(同步通知)，其中的 sink.next() 最多只能调用一次

```
Flux.generate(sink -> {
    sink.next("Hello");
    sink.complete();
}).subscribe(System.out::println);

final Random random = new Random();
Flux.generate(ArrayList::new, (list, sink) -> {
    int value = random.nextInt(100);
    list.add(value);
    sink.next(value);
    if (list.size() == 10) {
        sink.complete();
    }
    return list;
}).subscribe(System.out::println);
```

create() 方法与 generate() 方法的不同之处在于所使用的是 FluxSink 对象。FluxSink 支持同步和异步的消息产生，并且可以在一次调用中产生多个元素。generate 的 next 只能调用一次。

```
Flux.create(sink -> {
    for (int i = 0; i < 10; i++) {
        sink.next(i);
    }
    sink.complete();
}).subscribe(System.out::println);
```

3、 创建 Mono

Mono 的创建方式与 Flux 是很相似的。除了 Flux 所拥有的构造方式之外，还可以支持与 Callable、Runnable、Supplier 等接口集成。

```
// 只有完成信号的空数据流
Flux.just();
Flux.empty();
Mono.empty();
Mono.justOrEmpty(Optional.empty());
```

```
// 只有错误信号的数据流
Flux.error(new Exception("some error"));
Mono.error(new Exception("some error"));
```

空的数据流有什么用？举个例子，当我们从响应式的 DB 中获取结果的时候，就有可能为空：

```
Mono<User> findById(long id);
```

```
Flux<User> findAll();
```

无论是空还是发生异常，都需要通过完成/错误信号告知订阅者，已经查询完毕，但是抱歉没有得到值。

2、subscribe；订阅前什么都不会发生

数据流有了，假设我们想把每个数据元素原封不动地打印出来：

```
Flux.just(1, 2, 3, 4, 5, 6).subscribe(System.out::print); //123456
Mono.just(1).subscribe(System.out::println); //1
```

// 订阅并触发数据流

```
subscribe();
```

// 订阅并指定对正常数据元素如何处理

```
subscribe(Consumer<? super T> consumer);
```

// 订阅并定义对正常数据元素和错误信号的处理

```
subscribe(Consumer<? super T> consumer,
```

```
        Consumer<? super Throwable> errorConsumer);
```

// 订阅并定义对正常数据元素、错误信号和完成信号的处理

```
subscribe(Consumer<? super T> consumer,
```

```
        Consumer<? super Throwable> errorConsumer,
```

```
        Runnable completeConsumer);
```

// 订阅并定义对正常数据元素、错误信号和完成信号的处理，以及订阅发生时的处理逻辑

```
subscribe(Consumer<? super T> consumer,
```

```
        Consumer<? super Throwable> errorConsumer,
```

```
        Runnable completeConsumer,
```

```
        Consumer<? super Subscription> subscriptionConsumer);
```

```
Flux.just(1, 2, 3, 4, 5, 6).subscribe(
```

```
    System.out::println,
```

```
    System.err::println,
```

```
    () -> System.out.println("Completed!"));
```

```
//1
```

```
//2
```

```
//3
```

```
//4
```

```
//5
```

```
//6
```

```
//Completed!
```

```
Mono.error(new Exception("some error")).subscribe(
```

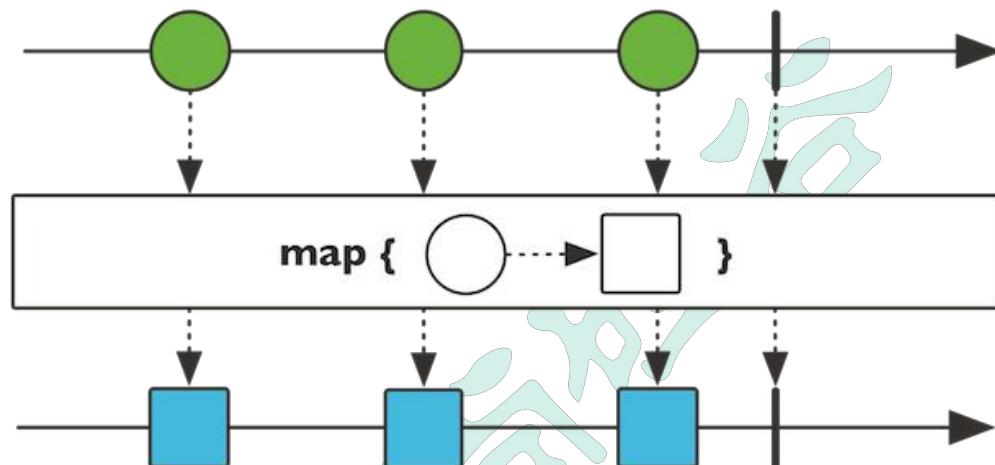
```
    System.out::println,
```

```
        System.err::println,  
        () -> System.out.println("Completed!")  
    );  
//java.lang.Exception: some error
```

这里需要注意的一点是，Flux.just(1, 2, 3, 4, 5, 6)仅仅声明了这个数据流，此时数据元素并未发出，只有 subscribe()方法调用的时候才会触发数据流。所以，订阅前什么都不会发生。

3、操作符

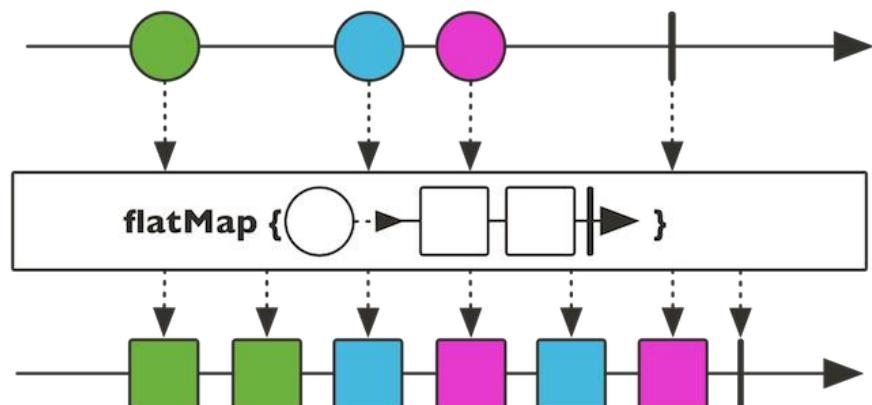
1)、map - 元素映射为新元素



```
Flux.range(1, 4).map(item->{  
    return item*2;  
}).subscribe(System.out::println);  
//2 4 6 8
```

2)、flatMap - 元素映射为流

flatMap 操作可以将每个数据元素转换/映射为一个流，然后将这些流合并为一个大的数据流。

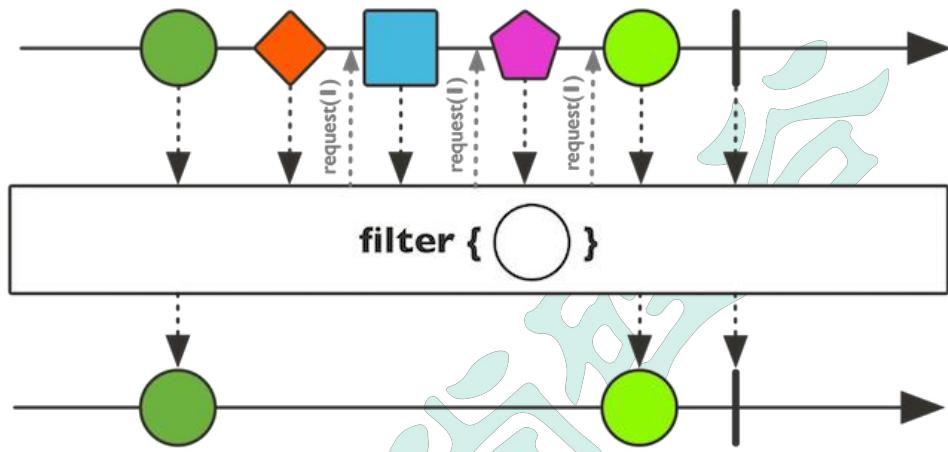


注意到，流的合并是异步的，先来先到，并非是严格按照原始序列的顺序（如图蓝色和红色方块是交叉的）。

```
Flux.just("Hello", "World")
    .flatMap(item -> Flux.fromArray(item.split("\s*")))
// .doOnNext(System.out::println) //doOnNext 方法是“偷窥式”的方法，不会消费数据流
    .subscribe(System.out::println);
```

3)、filter - 过滤

filter 操作可以对数据元素进行筛选。

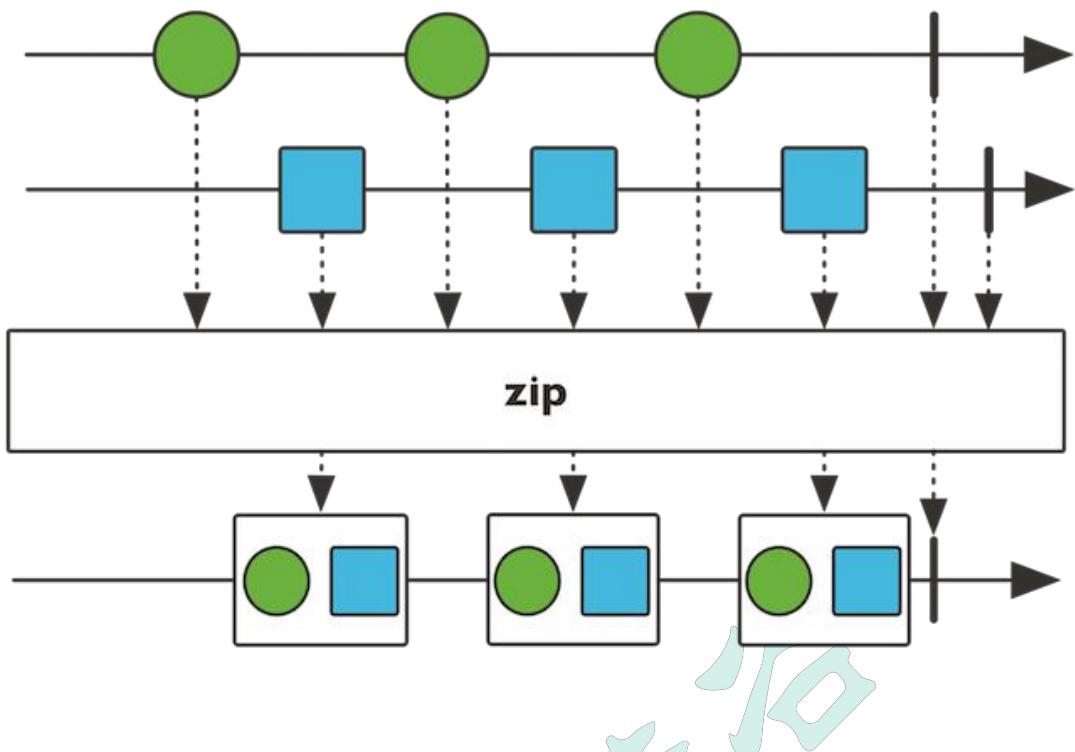


```
Flux.range(1, 10)
    .filter(p -> p % 2 == 0)
    .subscribe(System.out::println); //2,4,6,8,10
```

4)、zip - 一对二合并

将多个流一对一的合并起来。zip 有多个方法变体，我们介绍一个最常见的二合一的。

```
Flux.zip(Flux.range(1, 10), Flux.range(100, 10))
    .concatMap(item -> {
        System.out.println(item);
        Optional<Object> reduce = item.toList().stream().reduce((a, b) ->
Integer.parseInt(a.toString()) + Integer.parseInt(b.toString())));
        return Flux.just(reduce.get());
    })
    .subscribe(System.out::println, System.err::println);
```



5)、更多

Reactor 中提供了非常丰富的操作符，除了以上几个常见的，还有：

- 用于编程方式自定义生成数据流的 `create` 和 `generate` 等及其变体方法；
- 用于“无副作用的 peek”场景的 `doOnNext`、`doOnError`、`doOnComplete`、`doOnSubscribe`、`doOnCancel` 等及其变体方法；
- 用于数据流转换的 `when`、`and/or`、`merge`、`concat`、`collect`、`count`、`repeat` 等及其变体方法；
- 用于过滤/拣选的 `take`、`first`、`last`、`sample`、`skip`、`limitRequest` 等及其变体方法；
- 用于错误处理的 `timeout`、`onErrorReturn`、`onErrorResume`、`doFinally`、`retryWhen` 等及其变体方法；
- 用于分批的 `window`、`buffer`、`group` 等及其变体方法；
- 用于线程调度的 `publishOn` 和 `subscribeOn` 方法。

详细：

<https://htmlpreview.github.io/?https://github.com/get-set/reactor-core/blob/master-zh/src/docs/index.html#which-operator>

4、异常处理

在前面所提及的这些功能基本都属于正常的流处理，然而对于异常的捕获以及采取一些修正

手段也是同样重要的。

利用 Flux/Mono 框架可以很方便的做到这点。

将正常消息和错误消息分别打印

```
Flux.just(1, 2)
    .concatWith(Mono.error(new IllegalStateException()))
    .subscribe(System.out::println, System.err::println);
```

当产生错误时默认返回 0

```
Flux.just(1, 2)
    .concatWith(Mono.error(new IllegalStateException()))
    .onErrorReturn(0)
    .subscribe(System.out::println);
```

自定义异常时的处理

```
Flux.just(1, 2)
    .concatWith(Mono.error(new IllegalArgumentException()))
    .onErrorResume(e -> {
        if (e instanceof IllegalStateException) {
            return Mono.just(0);
        } else if (e instanceof IllegalArgumentException) {
            return Mono.just(-1);
        }
        return Mono.empty();
    })
    .subscribe(System.out::println);
```

当产生错误时重试

```
Flux.just(1, 2)
    .concatWith(Mono.error(new IllegalStateException()))
    .retry(1)
    .subscribe(System.out::println);
```

这里的 `retry(1)` 表示最多重试 1 次，而且重试将从订阅的位置开始重新发送流事件

5、调度器与线程模型

在 Reactor 中，对于多线程并发调度的处理变得异常简单。

在以往的多线程开发场景中，我们通常使用 `Executors` 工具类来创建线程池，通常有如下四种类型：

- `newCachedThreadPool` 创建一个弹性大小缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程；
- `newFixedThreadPool` 创建一个大小固定的线程池，可控制线程最大并发数，超出的线程

会在队列中等待；

- `newScheduledThreadPool` 创建一个大小固定的线程池，支持定时及周期性的任务执行；
- `newSingleThreadExecutor` 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

我们说过，响应式是异步化的，那么就会涉及到多线程的调度。

Reactor 提供了非常方便的调度器(Scheduler)工具方法，可以指定流的产生以及转换(计算)发布所采用的线程调度方式。

这些方式包括：

类别	描述
<code>immediate</code>	采用当前线程
<code>single</code>	单一可复用的线程
<code>elastic</code>	弹性可复用的线程池(IO型)
<code>parallel</code>	并行操作优化的线程池(CPU计算型)
<code>timer</code>	支持任务调度的线程池
<code>fromExecutorService</code>	自定义线程池

```
Flux.create(sink -> {
    sink.next(Thread.currentThread().getName());
    sink.complete();
})
.publishOn(Schedulers.single())
.map(x -> String.format("[%s] %s", Thread.currentThread().getName(), x))
.publishOn(Schedulers.elastic())
.map(x -> String.format("[%s] %s", Thread.currentThread().getName(), x))
.subscribeOn(Schedulers.parallel())
.toStream()
.forEach(System.out::println);
```

使用 `publishOn` 指定了流发布的调度器，`subscribeOn` 则指定的是流订阅的调度器。

首先是 `parallel` 调度器进行流数据的生成，接着使用一个 `single` 单线程调度器进行发布，此时经过第一个 `map` 转换为另一个 `Flux` 流，其中的消息叠加了当前线程的名称。最后进入的是一个 `elastic` 弹性调度器，再次进行一次同样的 `map` 转换。

最终，经过多层转换后的输出如下：

```
[elastic-2] [single-1] parallel-1
```

四、WebFlux

1、Pom

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

2、Controller

```
@RestController
public class HelloFluxController {

    @GetMapping("/hello")
    public Mono<String> hello() {
        return Mono.just("Hello");
    }
}
```

3、Service

```
@Service
public class UserService {

    public List<User> getUsers() throws Exception{
        long random = (long) ((Math.random()*50L) + 100L);
        TimeUnit.MILLISECONDS.sleep(random);
        return Arrays.asList(new User(1L, "zhangsan", 18),new
User(2L,"lisi",19));
    }
}
```

4、压力测试效果

ab 压测, gatling 压测, jmeter 压测

从 java1.4 开始 NIO，引入 Webflux->Reactor->Reactive Streams API，如果是 CPU 密集型，reactive 没用。4 核心 8 线程，一个核心绑定 2 线程，最好，不用进行线程切换。Reactive 是一种观察模式的扩展，Future 是阻塞式的

Supplier 只出（返回）不进（参数）

Consumer 只进（参数）不出（返回）

Function 又进（参数）又出（返回）

BiFunction 二元操作 func(a,b)

为什么 Reactive？传统 Tomcat 400 最大线程，多了就阻塞，线程多切换多。
Netty，Reactor 方式。少量线程几乎不切换，基于事件方式。处理大量并发。

基于异步非阻塞的响应式应用或驱动能够以少量且固定的线程应对高并发的请求或调用，
对于存在阻塞的场景，能够比多线程的并发方案提供更高的性能。

响应式和非阻塞并不是总能让应用跑的更快，况且将代码构建为非阻塞的执行方式本身还会带来少量的成本。但是在类似于 WEB 应用这样的高并发、少计算且 I/O 密集的应用中，响应式和非阻塞往往能够发挥出价值。尤其是微服务应用中，网络 I/O 比较多的情况下，效果会更加惊人。

1. 整体介绍

1) 安装vagrant

2) 安装Centos7

2. docker中安装mysql

3. docker中安装redis

4. 创建maven工程

5. 执行sql脚本

6. clone 人人开源

7. clone renren-generator

clone

修改配置

运行“renren-generator”

8. 微服务注册中心

9. 使用openfeign

10. 配置中心

1) 修改“gulimall-coupon”模块

2) 传统方式

3) nacos config

4) Nacos支持三种配置加载方案

 Namespace方案

 DataID方案

 Group方案

5) 同时加载多个配置集

11. 网关

1、注册“gulimall-gateway”到Nacos

1) 创建“gulimall-gateway”

2) 添加“gulimall-common”依赖和“spring-cloud-starter-gateway”依赖

3) “com.bigdata.gulimall.gulimallgateway.GulimallGatewayApplication”类上加上“@EnableDiscoveryClient”注解

4) 在Nacos中创建“gateway”命名空间，同时在该命名空间中创建“gulimall-gateway.yml”

5) 创建“bootstrap.properties”文件，添加如下配置，指明配置中心地址和所属命名空间

6) 创建“application.properties”文件，指定服务名和注册中心地址

7) 启动“gulimall-gateway”

2、案例

1) 创建“application.yml”

2) 启动“gulimall-gateway”

3) 测试

12. Vue

1、vue声明式渲染

2、双向绑定,模型变化，视图变化。反之亦然

3、事件处理

4、v-bind :单向绑定

- 5、v-model双向绑定
 - 6、v-on为按钮绑定事件
 - 7、v-for遍历循环
 - 过滤器
 - 组件化
 - 生命周期钩子函数
13. element ui
- 14. 递归树形结构获取数据
 - 15. 删除数据
 - 16. 菜单拖动
 - 拖动菜单时需要修改顺序和级别
 - 设置菜单拖动开关
 - 批量删除
 - 17. 品牌管理菜单
 - 添加“显示状态按钮”
 - 添加上传
 - 1) 添加依赖包
 - 2) 上传文件流
 - 其他方式
 - 1) 新建gulimall-third-party
 - 2) 添加依赖，将原来gulimall-common中的“spring-cloud-starter-alicloud-oss”依赖移动到该项目中
 - 3) 在主启动类中开启服务的注册和发现
 - 4) 在nacos中注册
 - 5) 编写配置文件
 - 6) 编写测试类
 - 上传组件
 - 18. JSR303校验
 - 步骤1：使用校验注解
 - 步骤2：在请求方法种，使用校验注解@Valid，开启校验，
 - 步骤3：给校验的Bean后，紧跟一个BindResult，就可以获取到校验的结果。拿到校验的结果，就可以自定义的封装。
 - 步骤4：统一异常处理
 - 19. 分组校验功能（完成多场景的复杂校验）
 - 1、给校验注解，标注上groups，指定什么情况下才需要进行校验
 - 2、业务方法参数上使用@Validated注解
 - 3、默认情况下，在分组校验情况下，没有指定指定分组的校验注解，将不会生效，它只会在不分组的情况下生效。
 - 20. 自定义校验功能
 - 1、编写一个自定义的校验注解
 - 2、编写一个自定义的校验器
 - 3、关联自定义的校验器和自定义的校验注解
 - 4、使用实例
 - 21. 商品SPU和SKU管理
 - 22. 点击子组件，父组件触发事件
 - 23. 规格参数新增与VO

24. 查询分组关联属性和删除关联

25. 查询分组未关联的属性

26. 添加属性和分组的关联关系

27. 发布商品

28. 商品管理

29. 仓库管理

30. 获取spu规格

31. 修改商品规格

小结:

1. 在open fen中会将调用的数据转换为JSON，接收方接收后，将JSON转换为对象，此时调用方和被调用方的处理JSON的对象不一定都是同一个类，只要它们的字段类型吻合即可。

2. 事务究竟要如何加上？

其他

1. 文档参考地址

2. 开机启动docker

3. 查看命令的安装位置

4. vscode中生成代码片段

5. vscode快捷键

6. 关闭eslint的语法检查

7. 安装mybatisx插件

8. mysql的批量删除

9. String.join

10. 在Service中微服务比较多的时候，可以配置将一些微服务放置到compound中，组成一个小组

11. mysql的dateTime和timestamp的区别？

12. SpringBoot中的事务

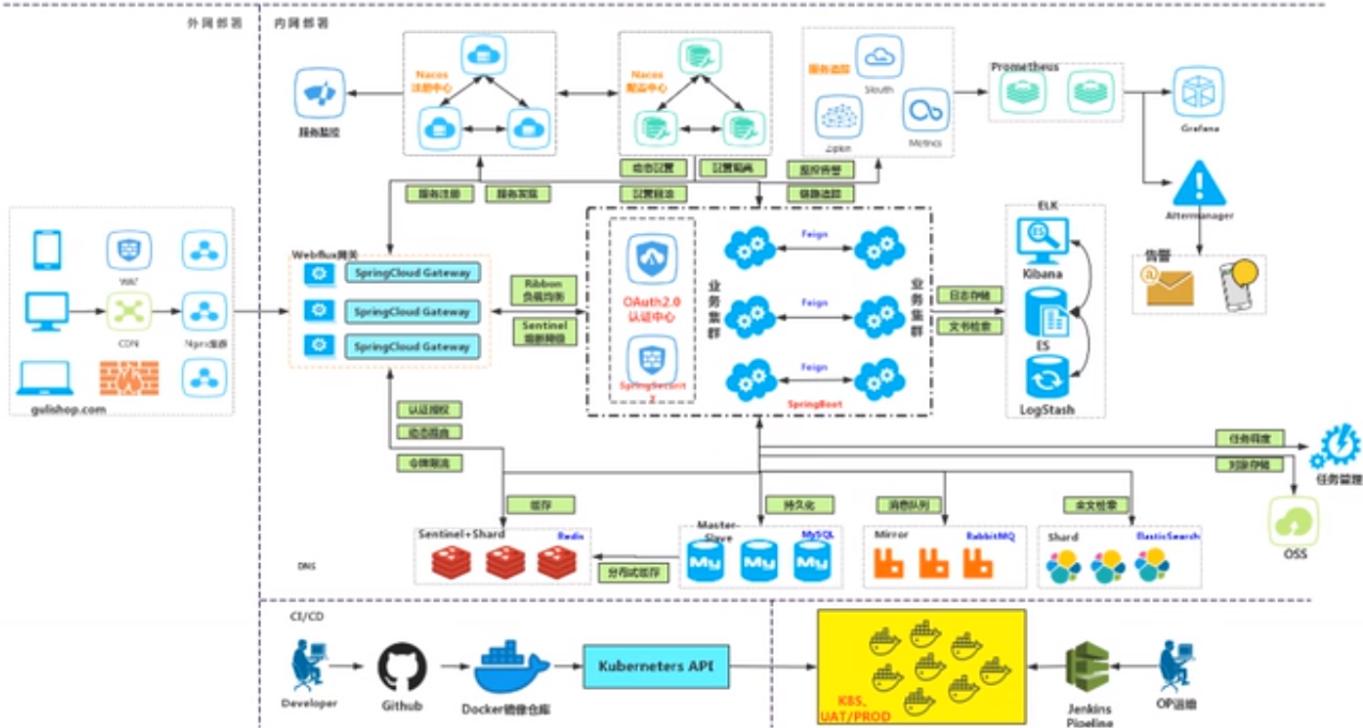
13. IDEA RESTFULL clinet

FAQ

1. TypeError: _vm.previewHandle is not a function

1. 整体介绍

谷粒商城-微服务架构图



1) 安装vagrant

2) 安装Centos7

```
$ vagrant init centos/7
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

执行完上面的命令后，会在用户的家目录下生成Vagrantfile文件。



```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'centos/7' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'centos/7'
    default: URL: https://vagrantcloud.com/centos/7
==> default: Adding box 'centos/7' (v1905.1) for provider: virtualbox
    default: Downloading:
https://vagrantcloud.com/centos/boxes/7/versions/1905.1/providers/virtualbox.box
    default: Download redirected to host: cloud.centos.org
    default: Progress: 0% (Rate: 6717/s, Estimated time remaining: 7:33:42)
```

下载镜像过程比较漫长，也可以采用先用下载工具下载到本地后，然后使用“vagrant box add”添加，再“vagrant up”即可



```
#将下载的镜像添加到virtualBox中
$ vagrant box add centos/7 E:\迅雷下载\CentOS-7-x86_64-Vagrant-1905_01.VirtualBox.box
==> box: Box file was not detected as metadata. Adding it directly...
==> box: Adding box 'centos/7' (v0) for provider:
    box: Unpacking necessary files from: file:///E:/%D1%B8%C0%D7%CF%C2%D4%D8/CentOS-7-
x86_64-Vagrant-1905_01.VirtualBox.box
    box:
==> box: Successfully added box 'centos/7' (v0) for 'virtualbox'!
```

#启动

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'centos/7'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: Administrator_default_1588497928070_24634
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
```

```
default: SSH username: vagrant
default: SSH auth method: private key
default:
default: Vagrant insecure key detected. Vagrant will automatically replace
default: this with a newly generated keypair for better security.
default:
default: Inserting generated public key within guest...
default: Removing insecure key from the guest if it's present...
default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
default: No guest additions were detected on the base box for this VM! Guest
default: additions are required for forwarded ports, shared folders, host only
default: networking, and more. If SSH fails on this machine, please install
default: the guest additions and repackage the box to continue.
default:
default: This is not an error message; everything may continue to work properly,
default: in which case you may ignore this message.
==> default: Configuring and enabling network interfaces...
==> default: Rsyncing folder: /cygdrive/c/Users/Administrator/ => /vagrant
```

vagrant ssh 开启SSH，并登陆到centos7



```
$ vagrant ssh
[vagrant@localhost ~]$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 52:54:00:8a:fe:e6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
        valid_lft 86091sec preferred_lft 86091sec
    inet6 fe80::5054:ff:fe8a:fee6/64 scope link
        valid_lft forever preferred_lft forever
```

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 08:00:27:d1:76:f6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global noprefixroute dynamic eth1
        valid_lft 892sec preferred_lft 892sec
    inet6 fe80::8c94:1942:ba09:2458/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
[vagrant@localhost ~]$
```



```
C:\Users\Administrator>ipconfig
```

Windows IP 配置

以太网适配器 VirtualBox Host-Only Network:

```
连接特定的 DNS 后缀 . . . . . :
本地链接 IPv6 地址 . . . . . : fe80::a00c:1ffa:a39a:c8c2%16
IPv4 地址 . . . . . : 192.168.56.1
子网掩码 . . . . . : 255.255.255.0
默认网关. . . . . :
```

配置网络信息，打开"Vagrantfile"文件：



```
config.vm.network "private_network", ip: "192.168.56.10"
```

修改完成后，重启启动vagrant



```
vagrant reload
```

检查宿主机和virtualBox之间的通信是否正常



```
[vagrant@localhost ~]$ ping 192.168.43.43
PING 192.168.43.43
(192.168.43.43) 56(84) bytes of data.
64 bytes from 192.168.43.43: icmp_seq=1 ttl=127 time=0.533 ms
64 bytes from 192.168.43.43: icmp_seq=2 ttl=127 time=0.659 ms
.
--- 192.168.43.43 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.533/0.596/0.659/0.063 ms
[vagrant@localhost ~]$
[vagrant@localhost ~]$
[vagrant@localhost ~]$ ping www.baidu.com
PING www.a.shifen.com (112.80.248.76) 56(84) bytes of data.
64 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=1 ttl=53 time=56.1 ms
64 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=2 ttl=53 time=58.5 ms
64 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=3 ttl=53 time=53.4 ms
.
```

开启远程登陆，修改“/etc/ssh/sshd_config”



```
PermitRootLogin yes
PasswordAuthentication yes
```

然后重启SSHD



```
systemctl restart sshd
```

使用Xshell或SecureCRT进行远程连接。

```
Last login: Sun May 3 09:41:39 2020
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:8a:fe:e6 brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
            valid_lft 85359sec preferred_lft 85359sec
        inet6 fe80::5054:ff:fe8a:fee6/64 scope link
            valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d1:76:f6 brd ff:ff:ff:ff:ff:ff
        inet 192.168.56.102/24 brd 192.168.56.255 scope global noprefixroute dynamic eth1
            valid_lft 630sec preferred_lft 630sec
        inet6 fe80::8c94:1942:ba09:2458/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
[root@localhost ~]#
```

2. docker中安装mysql



```
[root@hadoop-104 module]# docker pull mysql:5.7
5.7: Pulling from library/mysql
123275d6e508: Already exists
27cddf5c7140: Pull complete
c17d442e14c9: Pull complete
2eb72fffed068: Pull complete
d4aa125eb616: Pull complete
52560afb169c: Pull complete
68190f37a1d2: Pull complete
3fd1dc6e2990: Pull complete
85a79b83df29: Pull complete
35e0b437fe88: Pull complete
992f6a10268c: Pull complete
Digest: sha256:82b72085b2fcff073a6616b84c7c3bcbb36e2d13af838cec11a9ed1d0b183f5e
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

查看镜像

```
[root@hadoop-104 module]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
mysql 5.7 f5829c0eee9e 2 hours ago 455MB
[root@hadoop-104 module]#
```

启动mysql

```
sudo docker run -p 3306:3306 --name mysql \
-v /mydata/mysql/log:/var/log/mysql \
-v /mydata/mysql/data:/var/lib/mysql \
-v /mydata/mysql/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=root \
-d mysql:5.7
```

修改配置

```
[root@hadoop-104 conf]# pwd
/mydata/mysql/conf

[root@hadoop-104 conf]# cat my.cnf
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
[mysqld]
init_connect='SET collation_connection = utf8_unicode_ci'
init_connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_unicode_ci
skip-character-set-client-handshake
skip-name-resolve
[root@hadoop-104 conf]#
[root@hadoop-104 conf]# docker restart mysql
mysql
[root@hadoop-104 conf]#
```

进入容器查看配置：



```
[root@hadoop-104 conf]# docker exec -it mysql /bin/bash
root@b3a74e031bd7:/# whereis mysql
mysql: /usr/bin/mysql /usr/lib/mysql /etc/mysql /usr/share/mysql

root@b3a74e031bd7:/# ls /etc/mysql
my.cnf
root@b3a74e031bd7:/# cat /etc/mysql/my.cnf
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
[mysqld]
init_connect='SET collation_connection = utf8_unicode_ci'
init_connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_unicode_ci
skip-character-set-client-handshake
skip-name-resolve
root@b3a74e031bd7:/#
```

设置启动docker时，即运行mysql



```
[root@hadoop-104 ~]# docker update mysql --restart=always
mysql
[root@hadoop-104 ~]#
```

3. docker中安装redis

下载docker

```
[root@hadoop-104 ~]# docker pull redis
Using default tag: latest
latest: Pulling from library/redis
123275d6e508: Already exists
f2edbd6a658e: Pull complete
66960bede47c: Pull complete
79dc0b596c90: Pull complete
de36df38e0b6: Pull complete
602cd484ff92: Pull complete
Digest: sha256:1d0b903e3770c2c3c79961b73a53e963f4fd4b2674c2c4911472e8a054cb5728
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

启动docker

```
[root@hadoop-104 ~]# mkdir -p /mydata/redis/conf
[root@hadoop-104 ~]# touch /mydata/redis/conf/redis.conf
[root@hadoop-104 ~]# echo "appendonly yes" >> /mydata/redis/conf/redis.conf
[root@hadoop-104 ~]# docker run -p 6379:6379 --name redis -v /mydata/redis/data:/data \
> -v /mydata/redis/conf/redis.conf:/etc/redis/redis.conf \
> -d redis redis-server /etc/redis/redis.conf
ce7ae709711986e3f90c9278b284fe6f51f1c1102ba05f3692f0e934ceca1565
[root@hadoop-104 ~]#
```

连接到docker的redis

```
[root@hadoop-104 ~]# docker exec -it redis redis-cli
127.0.0.1:6379> set key1 v1
OK
127.0.0.1:6379> get key1
"v1"
127.0.0.1:6379>
```

设置redis容器在docker启动的时候启动



```
[root@hadoop-104 ~]# docker update redis --restart=always  
redis  
[root@hadoop-104 ~]#
```

4. 创建maven工程

5. 执行sql脚本

gulimall_oms.sql
gulimall_pms.sql
gulimall_sms.sql
gulimall_ums.sql
gulimall_wms.sql
pms_catalog.sql
sys_menus.sql

6. clone 人人开源

<https://gitee.com/renrenio>



人人开源

欢迎加入我们~

QQ群: 324780204 <https://www.renren.io>

关注

仓库 7

Issues 70

Pull Requests

动态

成员 3

热门

renren-fast

renren-fast是一个轻量级的Spring Boot2.1快速开发平台，其设计目标是开发迅速、学习简单、轻量级、易扩展；...

Java

@ 1668 ⭐ 5394 ⚡ 2762

SpringBoot2.0-Learning

SpringBoot2.0

Java

@ 1 ⭐ 8 ⚡ 1

renren-generator

人人开源项目的代码生成器，可在线生成entity、xml、o、service、vue、sql代码，减少70%以上的开发任务

Java

@ 624 ⭐ 2158 ⚡ 148

renren-security

采用SpringBoot2.0、MyBatis、Shiro框架，开发的一套权限系统，极低门槛，拿来即用。设计之初，就非常注重...

Java

@ 2253 ⭐ 6738 ⚡ 3552

renren-fast-vue

renren-fast-vue基于vue、element-ui构建开发，实现renren-fast后台管理前端功能，提供一套更优的前端解决方...

JavaScript

@ 200 ⭐ 888 ⚡ 805

renren-fast-adminlte

renren-fast-adminlte基于adminlte构建开发，实现renfast后台管理前端功能。

HTML

@ 28 ⭐ 97 ⚡ ?

克隆到本地：



```
git clone https://gitee.com/renrenio/renren-fast-vue.git
```

```
git clone https://gitee.com/renrenio/renren-fast.git
```

将拷贝下来的“renren-fast”删除“.git”后，拷贝到“gulimall”工程根目录下，然后将它作为gulimall的一个module

创建“gulimall_admin”的数据库，然后执行“renren-fast/db/mysql.sql”中的SQL脚本

修改“application-dev.yml”文件，默认为dev环境，修改连接mysql的url和用户名密码

```
spring:  
  datasource:  
    type: com.alibaba.druid.pool.DruidDataSource  
    druid:  
      driver-class-name: com.mysql.cj.jdbc.Driver  
      url: jdbc:mysql://192.168.137.14:3306/gulimall_admin?  
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai  
      username: root  
      password: root
```

启动“gulimall_admin”，然后访问“<http://localhost:8080/renren-fast/>”

← → ⌂ ⓘ localhost:8080/renren-fast/

```
{"msg":"invalid token","code":401}
```

安装node.js，并且安装仓库



```
npm config set registry http://registry.npm.taobao.org/
```

```
PS D:\tmp\renren-fast-vue> npm config set registry http://registry.npm.taobao.org/
PS D:\tmp\renren-fast-vue> npm install
npm WARN ajv-keywords@1.5.1 requires a peer of ajv@>=4.10.0 but none is installed. You
must install peer dependencies yourself.
npm WARN sass-loader@6.0.6 requires a peer of node-sass@^4.0.0 but none is installed. You
must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9:
wanted {"os": "darwin", "arch": "any"} (current: {"os": "win32", "arch": "x64"})

up to date in 17.227s
PS D:\tmp\renren-fast-vue>
```

```
PS D:\tmp\renren-fast-vue> npm run dev

> renren-fast-vue@1.2.2 dev D:\tmp\renren-fast-vue
> webpack-dev-server --inline --progress --config build/webpack.dev.conf.js

 10% building modules 5/10 modules 5 active ...-0!D:\tmp\renren-fast-
vue\src\main.js(node:19864) Warning: Accessing non-existent property 'cat' of module
exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:19864) Warning: Accessing non-existent property 'cd' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'chmod' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'cp' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'dirs' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'pushd' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'popd' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'echo' of module exports inside
circular dependency
(node:19864) Warning: Accessing non-existent property 'tempdir' of module exports inside
circular dependency
```

```
(node:19864) Warning: Accessing non-existent property 'pwd' of module exports inside circular dependency
```

常见问题1：“Module build failed: Error: Cannot find module 'node-sass”

运行过程中，出现“Module build failed: Error: Cannot find module 'node-sass'报错问题”，解决方法

用npm install -g cnpm --registry=<https://registry.npm.taobao.org>，从淘宝镜像那下载，然后cnpm下载成功。

最后输入cnpm install node-sass --save。npm run dev终于能跑起来了！！！

版权声明：本文为CSDN博主「夕阳下美了剪影」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

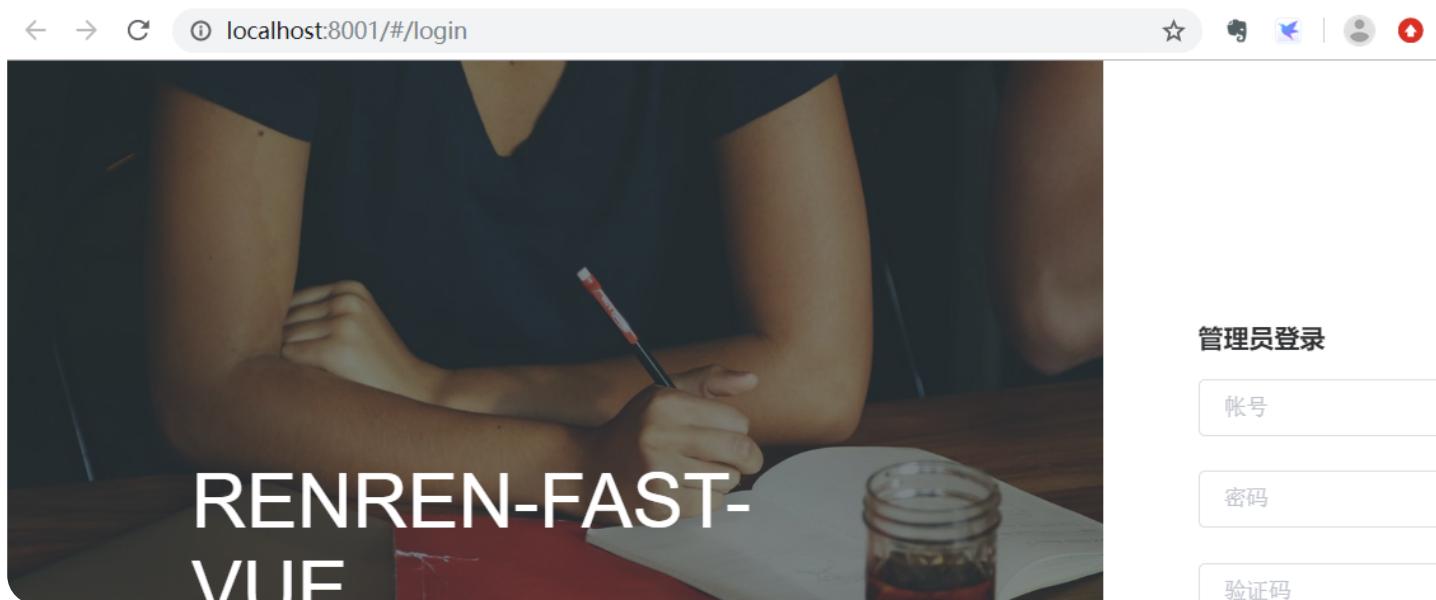
原文地址：https://blog.csdn.net/qq_38401285/article/details/86483278

常见问题2：cnpm - 解决 " cnpm : 无法加载文件 C:\Users\93457\AppData\Roaming\npm\cnpm.ps1，因为在此系统上禁止运行脚本。有关详细信息。。。 "

<https://www.cnblogs.com/500m/p/11634969.html>

所有问题的根源都在“node_modules”，npm install之前，应该将这个文件夹删除，然后再进行安装和运行。

再次运行npm run dev恢复正常：



7. clone renren-generator

clone

<https://gitee.com/renrenio/renren-generator.git>

然后将该项目放置到“gulimall”的跟路径下，然后添加该Module，并且提交到github上

修改配置

renren-generator/src/main/resources/generator.properties



#代码生成器，配置信息

```
mainPath=com.bigdata
#包名
package=com.bigdata.gulimall
moduleName=product
#作者
author=cosmoswong
#Email
email=cosmoswong@sina.com
#表前缀(类名不会包含表前缀)
tablePrefix=pms_
```

运行“renren-generator”

访问：<<http://localhost:80/>

The screenshot shows the homepage of the Renren Generator. The top navigation bar includes links for '捐赠作者' (Sponsor), '编程教程' (Programming Tutorials), and '项目文档' (Project Documentation). The main content area features a '基本信息' (Basic Information) section with a '获取帮助' (Get Help) link that points to a list of links for Git, official community, and project tracking. Below this is a '官方QQ群' (Official QQ Group) section.

点击“renren-fast”，能够看到它将“renren-fast”的所有表都列举了出来：

The screenshot shows the 'renren-fast' project page within the Renren Generator interface. It displays a table of database tables with columns: 表名 (Table Name), Engine, 表备注 (Table Notes), and 创建时间 (Create Time). The table lists 10 tables related to product management, such as pms_category, pms_sku_attr_value, and pms_spu_info_desc. A red box highlights the entire table area.

序号	表名	Engine	表备注	创建时间
1	pms_category	InnoDB	商品三级分类	2020-04-23 02:39:09
2	pms_sku_sale_attr_value	InnoDB	sku销售属性&值	2020-04-23 02:34:18
3	pms_sku_info	InnoDB	sku信息	2020-04-23 02:34:18
4	pms_sku_images	InnoDB	sku图片	2020-04-23 02:34:18
5	pms_product_attr_value	InnoDB	spu属性值	2020-04-23 02:34:18
6	pms_comment_replay	InnoDB	商品评价回复关系	2020-04-23 02:34:18
7	pms_category_brand_relation	InnoDB	品牌分类关联	2020-04-23 02:34:18
8	pms_spu_info_desc	InnoDB	spu信息介绍	2020-04-23 02:34:18
9	pms_brand	InnoDB	品牌	2020-04-23 02:34:18
10	pms_spu_info	InnoDB	spu信息	2020-04-23 02:34:18

选择所有的表，然后点击“生成代码”，将下载的“renren.zip”，解压后取出main文件夹，放置到“gulimall-product”项目的main目录中。

下面的几个module，也采用同样的方式来操作。

但是对于“undo_log”，存在一个问题

```
CREATE TABLE `undo_log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `branch_id` bigint(20) NOT NULL,
  `xid` varchar(100) NOT NULL,
  `context` varchar(128) NOT NULL,
  `rollback_info` longblob NOT NULL,
  `log_status` int(11) NOT NULL,
  `log_created` datetime NOT NULL,
  `log_modified` datetime NOT NULL,
  `ext` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

它的数据类型是“longblob”类型，逆向工程后，对应的数据类型未知：

```
41     */
42     private unknowType rollbackInfo;
43     /**
```

这个问题该要怎么解决？

8. 微服务注册中心

要注意nacos集群所在的server，一定要关闭防火墙，否则容易出现各种问题。

搭建nacos集群，然后分别启动各个微服务，将它们注册到Nacos中。

```
application:
  name: gulimall-coupon
cloud:
  nacos:
    discovery:
      server-addr: 192.168.137.14
```

查看注册情况：

<http://192.168.137.14:8848/nacos/#/serviceManagement?dataId=&group=&appName=&namespace=>

Nacos 1.2.1

public

服务列表 | public

服务管理

服务名	分组名称	集群数目	实例数	健康实例数	触发
gulimall_coupon	DEFAULT_GROUP	1	1	1	false

9. 使用openfeign

1)、引入open-feign

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

2)、编写一个接口，告诉SpringCloud这个接口需要调用远程服务

修改“com.bigdata.gulimall.coupon.controller.CouponController”，添加以下controller方法：

```
    @RequestMapping("/member/list")
    public R memberCoupons(){
        CouponEntity couponEntity = new CouponEntity();
        couponEntity.setCouponName("discount 20%");
        return R.ok().put("coupons", Arrays.asList(couponEntity));
    }
```

新建“com.bigdata.gulimall.member.feign.CouponFeignService”接口

```
@FeignClient("gulimall_coupon")
public interface CouponFeignService {
    @RequestMapping("/coupon/coupon/member/list")
    public R memberCoupons();
}
```

修改“com.bigdata.gulimall.member.GulimallMemberApplication”类，添加上“@EnableFeignClients”：

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients(basePackages = "com.bigdata.gulimall.member.feign")
public class GulimallMemberApplication {

    public static void main(String[] args) {
        SpringApplication.run(GulimallMemberApplication.class, args);
    }
}
```

声明接口的每一个方法都是调用哪个远程服务的那个请求

3)、开启远程调用功能

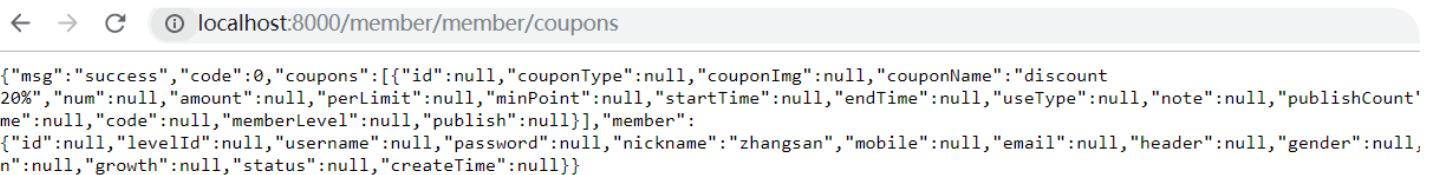
com.bigdata.gulimall.member.controller.MemberController



```
@RequestMapping( "/coupons" )
public R test(){
    MemberEntity memberEntity=new MemberEntity();
    memberEntity.setNickname( "zhangsan" );
    R memberCoupons = couponFeignService.memberCoupons();

    return
memberCoupons.put( "member" ,memberEntity).put( "coupons" ,memberCoupons.get( "coupons" ) );
}
```

(4)、访问<http://localhost:8000/member/member/coupons>



停止“gulimall-coupon”服务，能够看到注册中心显示该服务的健康值为0：

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值
gulimall-member	DEFAULT_GROUP	1	1	1	false
gulimall-coupon	DEFAULT_GROUP	1	1	0	true

再次访问：<http://localhost:8000/member/member/coupons>

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Apr 24 12:13:01 CST 2020

There was an unexpected error (type=Internal Server Error, status=500).

com.netflix.client.ClientException: Load balancer does not have available server for client: gulimall-coupon

启动“gulimall-coupon”服务，再次访问，又恢复了正常。

10. 配置中心

1) 修改“gulimall-coupon”模块

添加pom依赖：



```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

创建bootstrap.properties文件，该配置文件会优先于“application.yml”加载。



```
spring.application.name=gulimall-coupon
spring.cloud.nacos.config.server-addr=192.168.137.14:8848
```

2) 传统方式

为了详细说明config的使用方法，先来看原始的方式

创建“application.properties”配置文件，添加如下配置内容：

```
coupon.user.name="zhangsan"
coupon.user.age=30
```

修改“com.baomidou.gulimall.coupon.controller.CouponController”文件，添加如下内容：

```
@Value("${coupon.user.name}")
private String name;
@Value("${coupon.user.age}")
private Integer age;

@RequestMapping("/test")
public R getConfigInfo(){
    return R.ok().put("name",name).put("age",age);
}
```

启动“gulimall-coupon”服务：

访问：<http://localhost:7000/coupon/coupon/test>

← → ⌂ ① localhost:7000/coupon/coupon/test

```
{"msg":"success","code":0,"name":"\"zhangsan\"","age":30}
```

这样做存在的一个问题，如果频繁的修改application.properties，在需要频繁重新打包部署。下面我们将采用Nacos的配置中心来解决这个问题。

3) nacos config

1、在Nacos注册中心中，点击“配置列表”，添加配置规则：

The screenshot shows the Nacos 1.2.1 configuration interface. On the left sidebar, the '配置列表' (Configuration List) option is selected and highlighted with a red box. In the main area, a configuration entry is being created with the following details:

- Data ID:** gulimall-coupon (highlighted with a red box)
- Group:** DEFAULT_GROUP
- 更多高级选项** (More Advanced Options) button
- 描述:** (Description) input field
- 配置格式:** Properties (highlighted with a red box)
- 配置内容:** (Configuration Content)
1. coupon.user.name="lisi"
2. coupon.user.age=25 (highlighted with a red box)
- 发布** (Publish) button (highlighted with a red box)

DataID: gulimall-coupon

配置格式：properties

文件的命名规则为：\${spring.application.name}-\${spring.profiles.active}.\${spring.cloud.nacos.config.file-extension}

`\${spring.application.name}`：为微服务名

`\${spring.profiles.active}`：指明是哪种环境下的配置，如dev、test或info

`\${spring.cloud.nacos.config.file-extension}`：配置文件的扩展名，可以为properties、yml等

2、查看配置：

The screenshot shows the Nacos 1.2.1 configuration management interface. The '配置管理' (Configuration Management) tab is selected. The search bar shows 'public'. The results table displays the following configuration entry:

Data Id	Group	归属应用	操作
gulimall-coupon.properties	DEFAULT_GROUP		详情 示例代码 编辑 删除 更多

At the bottom of the interface, there are buttons for '删除' (Delete), '导出选中的配置' (Export Selected Configuration), and '克隆' (Clone). There are also pagination controls: '每页显示: 10' (Items per page: 10), '上一页' (Previous Page), '下一页' (Next Page), and a page number '1'.

3、修改“com.bigdata.gulimall.coupon.controller.CouponController”类，添加“@RefreshScope”注解

```
@RestController  
@RequestMapping("coupon/coupon")  
@RefreshScope  
public class CouponController {
```

这样都会动态的从配置中心读取配置。

4、访问：<http://localhost:7000/coupon/coupon/test>

能够看到读取到了nacos 中的最新的配置信息，并且在指明了相同的配置信息时，配置中心中设置的值优先于本地配置。

4) Nacos支持三种配置加载方案

Nacos支持“Namespace+group+data ID”的配置解决方案。

详情见：<https://github.com/alibaba/spring-cloud-alibaba/blob/master/spring-cloud-alibaba-docs/src/main/asciidoc-zh/nacos-config.adoc>

Namespace方案

通过命名空间实现环境区分

下面是配置实例：

1、创建命名空间：

“命名空间”—>“创建命名空间”：

命名空间名称	命名空间ID	配置数	操作
public(保留空间)		1	详情 删除 编辑
dev	a2c83f0b-e0a8-40fb-9b26-1e9d61be7d6d	0	详情 删除 编辑
test	30fab783-c133-4c27-9454-06e9301ef4a1	0	详情 删除 编辑
prop	e3a9b6df-16cb-445f-ae26-bc7d89ce8b98	0	详情 删除 编辑

创建三个命名空间，分别为dev, test和prop

2、回到配置列表中，能够看到所创建的三个命名空间

Data ID	Group	归属应用	操作
gulimall-coupon.properties	DEFAULT_GROUP		详情 示例代码 编辑

下面我们需要在dev命名空间下，创建“gulimall-coupon.properties”配置规则：

* Data ID:

* Group:

更多高级选项

描述:

配置格式: TEXT JSON XML YAML Properties

* 配置内容:

3、访问：<http://localhost:7000/coupon/coupon/test>

localhost:7000/coupon/coupon/test

```
{"msg": "success", "code": 0, "name": "\"lisi\"", "age": 25}
```

并没有使用我们在dev命名空间下所配置的规则，而是使用的是public命名空间下所配置的规则，这是怎么回事呢？

查看“gulimall-coupon”服务的启动日志：

```
2020-04-24 16:37:24.158  WARN 32792 --- [           main]
c.a.c.n.c.NacosPropertySourceBuilder      : Ignore the empty nacos configuration and get
it based on dataId[gulimall-coupon] & group[DEFAULT_GROUP]
2020-04-24 16:37:24.163  INFO 32792 --- [           main]
c.a.nacos.client.config.utils.JVMUtil    : isMultiInstance:false
2020-04-24 16:37:24.169  INFO 32792 --- [           main]
b.c.PropertySourceBootstrapConfiguration : Located property source:
[BootstrapPropertySource {name='bootstrapProperties-gulimall-
coupon.properties',DEFAULT_GROUP'}, BootstrapPropertySource {name='bootstrapProperties-
gulimall-coupon',DEFAULT_GROUP'}]
```

“**gulimall-coupon.properties**”，默认就是public命名空间中的内容中所配置的规则。

4、指定命名空间

如果想要使得我们自定义的命名空间生效，需要在“bootstrap.properties”文件中，指定使用哪个命名空间：

```
spring.cloud.nacos.config.namespace=a2c83f0b-e0a8-40fb-9b26-1e9d61be7d6d
```

这个命名空间ID来源于我们在第一步所创建的命名空间

NACOS 1.2.1

命名空间

[新建命名空间](#)

命名空间名称	命名空间ID	配置数	操作
public(保留空间)		1	详情 删除 编辑
dev	a2c83f0b-e0a8-40fb-9b26-1e9d61be7d6d	0	详情 删除 编辑
test	30fab783-c133-4c27-9454-06e9301ef4a1	0	详情 删除 编辑
prop	e3a9b6df-16cb-445f-ae26-bc7d89ce8b98	0	详情 删除 编辑

5、重启“gulimall-coupon”，再次访问：<http://localhost:7000/coupon/coupon/test>

localhost:7000/coupon/coupon/test

```
{"msg": "success", "code": 0, "name": "\"wangwu\"", "age": 19}
```

但是这种命名空间的粒度还是不够细化，对此我们可以为项目的每个微服务module创建一个命名空间。

6、为所有微服务创建命名空间

命名空间名称	命名空间ID	配置数	操作
public(保留空间)		1	详情 删除 编辑
dev	a2c83f0b-e0a8-40fb-9b26-1e9d61be7d6d	1	详情 删除 编辑
test	30fab783-c133-4c27-9454-06e9301ef4a1	0	详情 删除 编辑
prop	e3a9b6df-16cb-445f-ae26-bc7d89ce8b98	0	详情 删除 编辑
coupon	7905c915-64ad-4066-8ea9-ef63918e5f79	0	详情 删除 编辑
product	3c50ffaa-010b-4b59-9372-902e35059232	0	详情 删除 编辑
ware	2bd4d2fd-a2de-4a5c-8dfa-68d009d19b31	0	详情 删除 编辑
order	795521fa-77ef-411e-a8d8-0889fdf6964	0	详情 删除 编辑
member	a2342ef4-a63b-45db-91ba-1eabfeac378d	0	详情 删除 编辑

7、回到配置列表选项卡，克隆public的配置规则到coupon命名空间下

NACOS 1.2.1

配置管理 | public 查询结果：共查询到 1 条满足要求的配置。

Data ID: 模糊查询请输入Data ID Group: 模糊查询请输入Group

Data Id	Group
gulimall-coupon.properties	

删除 导出选中的配置 克隆

克隆配置

源空间: public | 配置数量: 1 | 选中的条目

* 目标空间: coupon | 7905c915-64ad-4066-8ea9-ef63918e5f79

相同配置: 终止导入

开始克隆

修改 Data Id 和 Group (可选操作)

Data Id	Group
gulimall-coupon.properties	DEFAULT_GROUP

切换到coupon命名空间下，查看所克隆的规则：

配置管理 | coupon 7905c915-64ad-4066-8ea9-ef63918e5f79 查询结果：共查询到 1 条满足要求的配置。

Data ID: 模糊查询请输入Data ID Group: 模糊查询请输入Group

Data Id	Group	归属应用:	操作
gulimall-coupon.properties	DEFAULT_GROUP		详情 示例代码 编辑 删除 更多

8、修改“gulimall-coupon”下的bootstrap.properties文件，添加如下配置信息

spring.cloud.nacos.config.namespace=7905c915-64ad-4066-8ea9-ef63918e5f79

这里指明的是，读取时使用coupon命名空间下的配置。

9、重启“gulimall-coupon”，访问：<http://localhost:7000/coupon/coupon/test>

localhost:7000/coupon/coupon/test

```
{"msg": "success", "code": 0, "name": "\\"lisi\\\"", "age": 25}
```

DataID方案

通过指定spring.profile.active和配置文件的DataID，来使不同环境下读取不同的配置，读取配置时，使用的是默认命名空间public，默认分组（default_group）下的DataID。

默认情况，Namespace=public，Group=DEFAULT GROUP，默认Cluster是DEFAULT

Group方案

通过Group实现环境区分

实例：通过使用不同的组，来读取不同的配置，还是以上面的gulimall-coupon微服务为例

1、新建“gulimall-coupon.properties”，将它置于“tmp”组下

The screenshot shows a configuration management interface. At the top, there are fields for 'Data ID' (set to 'gulimall-coupon.properties') and 'Group' (set to 'tmp'). Below these are 'Advanced Options' and a 'Description' field containing '临时组'. Under 'Format', 'Properties' is selected. In the 'Content' section, two lines of configuration are shown: '1 coupon.user.name="zhaoliu"' and '2 coupon.user.age=59'. The 'Properties' tab is highlighted.

2、修改“bootstrap.properties”配置，添加如下的配置

The screenshot shows a configuration management interface with three colored dots (red, yellow, green) at the top. Below them, the configuration 'spring.cloud.nacos.config.group=tmp' is listed in blue text.

3、重启“gulimall-coupon”，访问：<http://localhost:7000/coupon/coupon/test>

The screenshot shows a browser window with the URL 'localhost:7000/coupon/coupon/test'. The page displays a JSON response: {"msg": "success", "code": 0, "name": "\"zhaoliu\"", "age": 59}.

5) 同时加载多个配置集

当微服务数量很庞大时，将所有配置都书写到一个配置文件中，显然不是太合适。对此我们可以将配置按照功能的不同，拆分为不同的配置文件。

如下面的配置文件：

```
server:
  port: 7000

spring:
  datasource:
    #MySQL配置
    driverClassName: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.137.14:3306/gulimall_sms?
    useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: root

  application:
    name: gulimall-coupon
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.137.14:8848

mybatis-plus:
  global-config:
    db-config:
      id-type: auto
  mapper-locations: classpath:/mapper/**/*.xml
```

我们可以将，

数据源有关的配置写到一个配置文件中：

```
spring:
  datasource:
    #MySQL配置
    driverClassName: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.137.14:3306/gulimall_sms?
    useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: root
```

和框架有关的写到另外一个配置文件中：

```
mybatis-plus:
  global-config:
    db-config:
      id-type: auto
  mapper-locations: classpath:/mapper/**/*.xml
```

也可以将上面的这些配置交给nacos来进行管理。

实例：将“gulimall-coupon”的“application.yml”文件拆分为多个配置，并放置到nacos配置中心

1、创建“datasource.yml”，用于存储和数据源有关的配置

```
spring:
  datasource:
    #MySQL配置
    driverClassName: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.137.14:3306/gulimall_sms?
    useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: root
```

在coupon命名空间中，创建“datasource.yml”配置

* Data ID: datasource.yml
* Group: dev

更多高级选项

描述: 数据源相关配置

配置格式: TEXT JSON XML **YAML** HTML Properties

* 配置内容: :

```
1 | spring:
2 |   datasource:
3 |     #MySQL配置
4 |     driverClassName: com.mysql.cj.jdbc.Driver
5 |     url: jdbc:mysql://192.168.137.14:3306/gulimall_sms?useUnicode=true&characterEncoding=UTF-8&useSSL=false
6 |     username: root
7 |     password: root
```

2、将和mybatis相关的配置，放置到“mybatis.yml”中

```
mybatis-plus:
  global-config:
    db-config:
      id-type: auto
  mapper-locations: classpath:/mapper/**/*.xml
```

* Data ID: mybatis.yml
* Group: dev

更多高级选项

描述: mybatis相关配置

配置格式: TEXT JSON XML **YAML** HTML Properties

* 配置内容: :

```
1 | mybatis-plus:
2 |   global-config:
3 |     db-config:
4 |       id-type: auto
5 |     mapper-locations: classpath:/mapper/**/*.xml
```

3、创建“other.yml”配置，保存其他的配置信息

```
server:  
  port: 7000  
  
spring:  
  application:  
    name: gulimall-coupon  
  cloud:  
    nacos:  
      discovery:  
        server-addr: 192.168.137.14:8848
```

* Data ID: other.yml

* Group: dev

[更多高级选项](#)

描述: 其他配置

配置格式: TEXT JSON XML YAML HTML Properties

* 配置内容: ② :

```
1 server:  
2   port: 7000  
3  
4 spring:  
5   application:  
6     name: gulimall-coupon  
7   cloud:  
8     nacos:  
9       discovery:  
10      server-addr: 192.168.137.14:8848
```

现在“mybatis.yml”、“datasource.yml”和“other.yml”共同构成了微服务的配置。

4、修改“gulimall-coupon”的“bootstrap.properties”文件，加载“mybatis.yml”、“datasource.yml”和“other.yml”配置



```
spring.cloud.nacos.config.extension-configs[0].data-id=mybatis.yml
spring.cloud.nacos.config.extension-configs[0].group=dev
spring.cloud.nacos.config.extension-configs[0].refresh=true

spring.cloud.nacos.config.extension-configs[1].data-id=datasource.yml
spring.cloud.nacos.config.extension-configs[1].group=dev
spring.cloud.nacos.config.extension-configs[1].refresh=true

spring.cloud.nacos.config.extension-configs[2].data-id=other.yml
spring.cloud.nacos.config.extension-configs[2].group=dev
spring.cloud.nacos.config.extension-configs[2].refresh=true
```

"spring.cloud.nacos.config.ext-config"已经被废弃，建议使用"spring.cloud.nacos.config.extension-configs"

5、注释“application.yml”文件中的所有配置

6、重启“gulimall-coupon”服务，然后访问：<http://localhost:7000/coupon/coupon/test>

← → ⌂ ⓘ localhost:7000/coupon/coupon/test

```
{"msg": "success", "code": 0, "name": "\"lisi\"", "age": 25}
```

7、访问：<http://localhost:7000/coupon/coupon/list>，查看是否能够正常的访问数据库

← → ⌂ ⓘ localhost:7000/coupon/coupon/list

```
{"msg": "success", "code": 0, "page": {"totalCount": 0, "pageSize": 10, "totalPage": 0, "currPage": 1, "list": []}}
```

小结：

- 1)、微服务任何配置信息，任何配置文件都可以放在配置中心；
- 2)、只需要在bootstrap.properties中，说明加载配置中心的哪些配置文件即可；
- 3)、@Value, @ConfigurationProperties。都可以用来获取配置中心中所配置的信息；
- 4)、配置中心有的优先使用配置中心中的，没有则使用本地的配置。

11. 网关

1、注册“gulimall-gateway”到Nacos

1) 创建“gulimall-gateway”

SpringCloud gateway

2) 添加“gulimall-common”依赖和“spring-cloud-starter-gateway”依赖

```
<dependency>
    <groupId>com.bigdata.gulimall</groupId>
    <artifactId>gulimall-common</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

3)

“**com.bigdata.gulimall.gulimallgateway.GulimallGatewayApplication**”类上加上“**@EnableDiscoveryClient**”注解

4) 在Nacos中创建“gateway”命名空间，同时在该命名空间中创建“gulimall-gateway.yml”

* Data ID: gulimall-gateway.yml

* Group: DEFAULT_GROUP

[更多高级选项](#)

描述: gateway

Beta发布: 默认不要勾选。

配置格式: TEXT JSON XML YAML HTML Properties

配置内容 [?](#):

```
1 spring:
2   application:
3     name: gulimall-gateway
```

5) 创建“bootstrap.properties”文件，添加如下配置，指明配置中心地址和所属命名空间

● ○ ●

```
spring.application.name=gulimall-gateway
spring.cloud.nacos.config.server-addr=192.168.137.14:8848
spring.cloud.nacos.config.namespace=1c82552e-1af0-4ced-9a48-26f19c2d315f
```

6) 创建“application.properties”文件，指定服务名和注册中心地址

● ○ ●

```
spring.application.name=gulimall-gateway
spring.cloud.nacos.discovery.server-addr=192.168.137.14:8848
server.port=88
```

7) 启动“gulimall-gateway”

启动报错：

```
● ● ●

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded
datasource could be configured.

Reason: Failed to determine a suitable driver class
```

解决方法：在“com.baomidou.gulimall.gulimallgateway.GulimallGatewayApplication”中排除和数据源相关的配置

```
● ● ●

@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
```

重新启动

访问：<http://192.168.137.14:8848/nacos/#>, 查看到该服务已经注册到了Nacos中

The screenshot shows the Nacos 1.2.1 web interface. The top navigation bar includes links for public, dev, test, prop, coupon, product, ware, order, member, and gateway. The left sidebar has sections for Configuration Management, Service Management, Service List, Subscribers List, Cluster Management, and Permission Control. The Service Management section is active, showing a search bar for service name and group name, and a toggle for hidden empty services. The Service List table displays two entries: 'gulimall-gateway' and 'gulimall-coupon'. The 'gulimall-gateway' row is highlighted with a red border. The table columns are: 服务名 (Service Name), 分组名称 (Group Name), 集群数目 (Cluster Count), 实例数 (Instance Count), and 健康实例数 (Healthy Instance Count). Both services have a cluster count of 1, 1 instance, and 1 healthy instance.

服务名	分组名称	集群数目	实例数	健康实例数
gulimall-gateway	DEFAULT_GROUP	1	1	1
gulimall-coupon	DEFAULT_GROUP	1	1	1

2、案例

现在想要实现针对于“<http://localhost:88/hello?url=baidu>”，转发到“<https://www.baidu.com>”，针对于“<http://localhost:88/hello?url=qq>”的请求，转发到“<https://www.qq.com/>”

1) 创建“application.yml”

```
spring:
  cloud:
    gateway:
      routes:
        - id: baidu_route
          uri: https://www.baidu.com
          predicates:
            - Query=url, baidu
        - id: qq_route
          uri: https://www.qq.com/
          predicates:
            - Query=url, qq
```

2) 启动“gulimall-gateway”

3) 测试

访问： <http://localhost:88/hello?url=baidu>

访问： <http://localhost:88/hello?url=qq>

12. Vue

安装vue



```
# 最新稳定版
$ npm install vue
```

1、vue声明式渲染



```
let vm = new Vue({
  el: "#app", //绑定元素
  data: { //封装数据
    name: "张三",
    num: 1
  },
  methods:{ //封装方法
    cancle(){
      this.num -- ;
    },
    hello(){
      return "1"
    }
  }
});
```

2、双向绑定,模型变化，视图变化。反之亦然

双向绑定使用v-model



```
<input type="text" v-model="num">
```



```
<h1> {{name}} ,非常帅，有{{num}}个人为他点赞{{hello()}}</h1>
```

9 点赞 取消

lisi ,非常帅，有9个人为他点赞1

Console tab in DevTools showing the following messages:

- 4 messages
- 3 user messages
- 1 error
 - No warnings
- 3 info
- No verbose

The error message is expanded:

```
x Expression  
not available
```

The expanded error message shows the assignment and its value:> vm.name
<- "张三"
>
> vm.name="lisi"
<- "lisi"
>

3、事件处理

v-xx: 指令

- 1、创建vue实例，关联页面的模板，将自己的数据（data）渲染到关联的模板，响应式的
- 2、指令来简化对dom的一些操作。
- 3、声明方法来做更复杂的操作。methods里面可以封装方法。

v-on是按钮的单击事件：

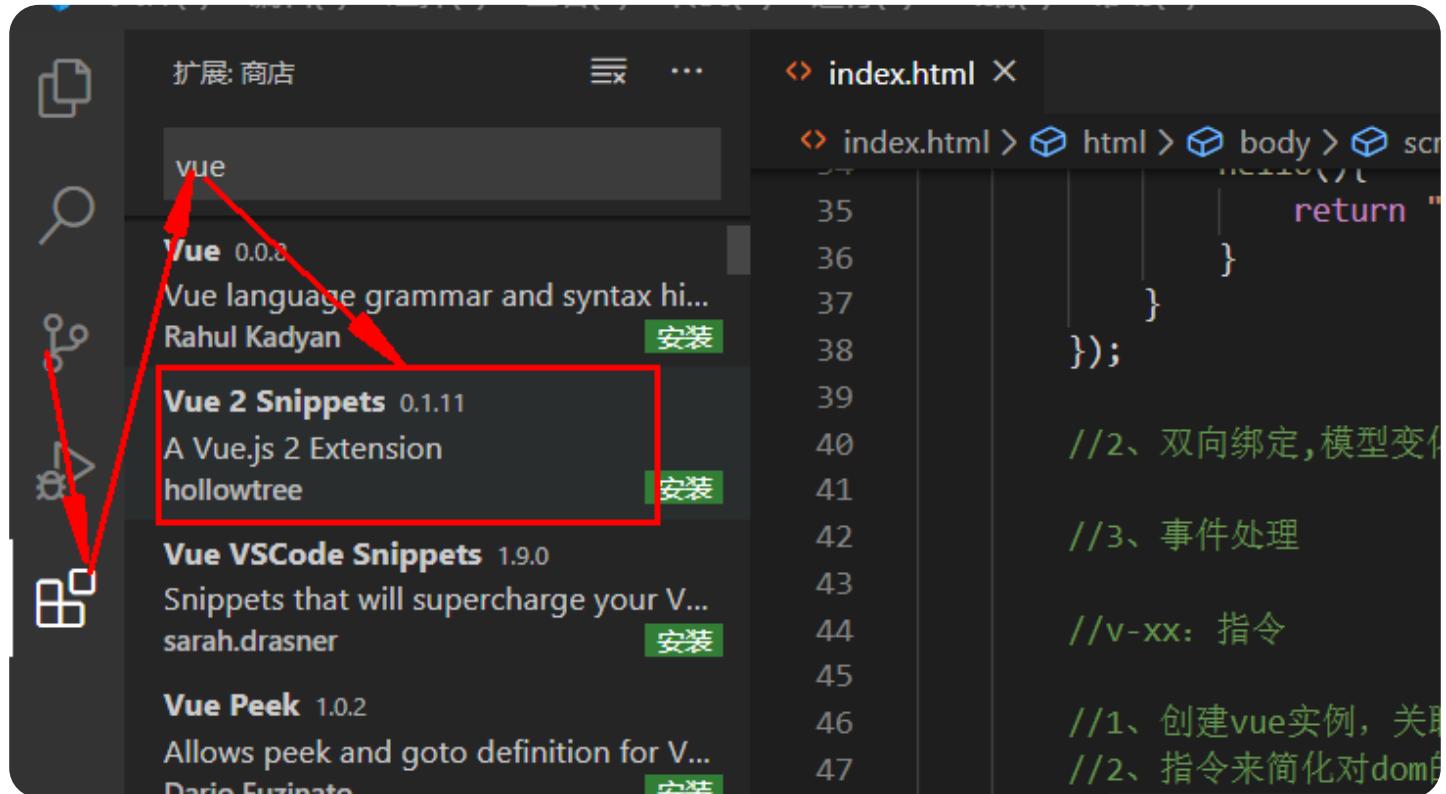


```
<button v-on:click="num++">点赞</button>
```

在VUE中el,data和vue的作用:

- el: 用来绑定数据;
- data: 用来封装数据;
- methods: 用来封装方法, 并且能够封装多个方法, 如何上面封装了cancel和hello方法。

安装“Vue 2 Snippets”, 用来做代码提示



为了方便的在浏览器上调试VUE程序, 需要安装“[vue-devtools](#)”, 编译后安装到chrome中即可。

详细的使用方法见: [Vue调试神器vue-devtools安装](#)

“v-html”不会对于HTML标签进行转义, 而是直接在浏览器上显示data所设置的内容;而“ v-text”会对html标签进行转义



```
<span v-html="msg"></span>
<br/>
<span v-text="msg"></span>
</div>

<script src="../node_modules/vue/dist/vue.js"></script>

<script>
new Vue({
  el: "#app",
  data: {
    msg: "<h1>Hello</h1>",
    link: "http://www.baidu.com"
  },
  methods: {
    hello(){
      return "World"
    }
  }
})
</script>
```

运行结果：

← → C ⓘ 127.0.0.1:5500/1、指令/1、v-text、v-html.html

<h1>Hello</h1> 2 World

Hello

<h1>Hello</h1>

{} :称为差值表达式，它必须要写在Html表达式，可以完成数学运算和方法调用

4、v-bind :单向绑定

给html标签的属性绑定



```
<!-- 给html标签的属性绑定 -->
<div id="app">

    <a v-bind:href="link">gogogo</a>

    <!-- class,style {class名: 加上? }-->
    <span v-bind:class="{active:isActive, 'text-danger':hasError}"
          :style="{color: color1,fontSize: size}">你好</span>

</div>

<script src="../node_modules/vue/dist/vue.js"></script>

<script>
    let vm = new Vue({
        el: "#app",
        data: {
            link: "http://www.baidu.com",
            isActive:true,
            hasError:true,
            color1:'red',
            size:'36px'
        }
    })
</script>
```

上面所完成的任务就是给a标签绑定一个超链接。并且当“isActive”和“hasError”都是true的时候，将属性动态的绑定到，则绑定该“active”和 “text-danger”class。这样可以动态的调整属性的存在。

而且如果想要实现修改vm的“color1”和“size”， span元素的style也能够随之变化，则可以写作v-bind:style，也可以省略v-bind。

5、v-model双向绑定



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

  <!-- 表单项，自定义组件 -->
  <div id="app">

    精通的语言：
    <input type="checkbox" v-model="language" value="Java"> Java<br/>
    <input type="checkbox" v-model="language" value="PHP"> PHP<br/>
    <input type="checkbox" v-model="language" value="Python"> Python<br/>
    选中了 {{language.join(",")}}
  </div>

  <script src="../node_modules/vue/dist/vue.js"></script>

  <script>
    let vm = new Vue({
      el: "#app",
      data: {
        language: []
      }
    })
  </script>

</body>
</html>
```

上面完成的功能就是通过“v-model”为输入框绑定多个值，能够实现选中的值，在data的language也在不断的發生着变化，

精通的语言: java

PHP

Python

选中了 Python

The screenshot shows the Vue DevTools interface. The top bar has tabs for 查看器 (Inspector), 控制台 (Console), 调试器 (Debugger), 网络 (Network), 样式编辑器 (Style Editor), 性能 (Performance), 内存 (Memory), 存储 (Storage), 无障碍环境 (Accessibility), and Vue. The Vue tab is active. Below the tabs, it says "Ready. Detected Vue 2.6.11." and has links for Components, Vuex, and Events. A search bar says "Filter components". The main area shows "<Root> = \$vm0" in a green bar. To the right, there's a tree view under "<Root>". The "data" node is expanded, showing "language: Array[1]" with "0: 'Python'". A red box highlights this "language" array.

如果在控制台上指定`vm.language=["Java","PHP"]`, 则`data`值也会跟着变化。

精通的语言: java

PHP

Python

选中了 Java,PHP

The screenshot shows the Vue DevTools interface. The top bar has tabs for 查看器 (Inspector), 调试器 (Debugger), 网络 (Network), 样式编辑器 (Style Editor), 性能 (Performance), 内存 (Memory), 存储 (Storage), 无障碍环境 (Accessibility), 控制台 (Console), and Vue. The Vue tab is active. Below the tabs, it says "Ready. Detected Vue 2.6.11." and has links for Components, Vuex, Events, and Routing. A search bar says "Filter components". The main area shows "<Root> = \$vm0" in a green bar. To the right, there's a tree view under "<Root>". The "data" node is expanded, showing "language: Array[2]" with "0: 'Java'" and "1: 'PHP'". A red box highlights this "language" array.

过滤输出

① You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at <https://vuejs.org/guide/deployment.html>

vue-devtools Detected Vue v2.6.11

» vm.language=[“java”, “php”]

← ▶ Array [“java”, “php”]

vue-devtools Open Vue devtools to see component details

» vm.language=[“java”, “PHP”]

← ▶ Array [“java”, “PHP”]

» vm.language=[“Java”, “PHP”]

← ▶ Array [“Java”, “PHP”]

⚠️ 贴吧警告: 贴吧您不了解的东西时请多心\(\)/。这可能会导致攻击者窃取您的身份信息或控制您的计算机。如果仍想贴吧, 请在下方输入“allow posting”(不必按回车键)以允许贴吧。

通过“v-model”实现了页面发生了变化，则数据也发生变化，数据发生变化，则页面也发生变化，这样就实现了双向绑定。

数组的连接操作: 选中了 `{{language.join(",")}}`

6、v-on为按钮绑定事件



```
<!--事件中直接写js片段-->
<button v-on:click="num++">点赞</button>
<!--事件指定一个回调函数，必须是Vue实例中定义的函数-->
<button @click="cancle">取消</button>
```

上面是为两个按钮绑定了单击事件，其中一个对于num进行自增，另外一个自减。

v-on:click也可以写作@click

事件的冒泡：



```
<!-- 事件修饰符 -->
<div style="border: 1px solid red;padding: 20px;" v-on:click="hello">
    大div
    <div style="border: 1px solid blue;padding: 20px;" @click="hello">
        小div <br />
        <a href="http://www.baidu.com" @click.prevent="hello">去百度</a>
    </div>
</div>
```

上面的这两个嵌套div中，如果点击了内层的div，则外层的div也会被触发；这种问题可以事件修饰符来完成：



```
<!-- 事件修饰符 -->
<div style="border: 1px solid red;padding: 20px;" v-on:click.once="hello">
    大div
    <div style="border: 1px solid blue;padding: 20px;" @click.stop="hello">
        小div <br />
        <a href="http://www.baidu.com" @click.prevent.stop="hello">去百度</a>
        <!--这里禁止了超链接的点击跳转操作，并且只会触发当前对象的操作-->
    </div>
</div>
```

关于事件修饰符：

在事件处理程序中调用 `event.preventDefault()` 或 `event.stopPropagation()` 是非常常见的需求。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法只有纯粹的数据逻辑，而不是去处理 DOM 事件细节。

为了解决这个问题，`Vue.js` 为 `v-on` 提供了事件修饰符。修饰符是由点开头的指令后缀来表示的。

- `stop`：阻止事件冒泡到父元素
- `prevent`：阻止默认事件发生
- `capture`：使用事件捕获模式
- `self`：只有元素自身触发事件才执行。（冒泡或捕获的都不执行）
- `once`：只执行一次

按键修饰符：

3、按键修饰符

在监听键盘事件时，我们经常需要检查常见的键值。`Vue` 允许为 `v-on` 在监听键盘事件时添加按键修饰符：

```
<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->
<input v-on:keyup.13="submit">
```

记住所有的 `keyCode` 比较困难，所以 `Vue` 为最常用的按键提供了别名：

```
<!-- 同上 -->
<input v-on:keyup.enter="submit">
<!-- 缩写语法 -->
<input @keyup.enter="submit">
```

全部的按键别名：

- `enter`
- `tab`
- `delete` (捕获“删除”和“退格”键)
- `esc`

- `space`
- `up`
- `down`
- `left`
- `right`

7、v-for遍历循环



```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>

  <div id="app">
    <ul>
      <li v-for="(user,index) in users" :key="user.name" v-if="user.gender == '女'">
        <!-- 1、显示user信息: v-for="item in items" -->
        当前索引: {{index}} ==> {{user.name}} ==> {{user.gender}} ==>{{user.age}}
<br>
        <!-- 2、获取数组下标: v-for="(item,index) in items" -->
        <!-- 3、遍历对象:
            v-for="value in object"
            v-for="(value,key) in object"
            v-for="(value,key,index) in object"
          -->
        对象信息:
        <span v-for="(v,k,i) in user">{{k}}=={{v}}=={{i}};</span>
        <!-- 4、遍历的时候都加上:key来区分不同数据, 提高vue渲染效率 -->
      </li>
    </ul>
    <ul>
      <li v-for="(num,index) in nums" :key="index"></li>
    </ul>
  </div>
  <script src="../node_modules/vue/dist/vue.js"></script>
  <script>
```

```

let app = new Vue({
  el: "#app",
  data: {
    users: [{ name: '柳岩', gender: '女', age: 21 },
             { name: '张三', gender: '男', age: 18 },
             { name: '范冰冰', gender: '女', age: 24 },
             { name: '刘亦菲', gender: '女', age: 18 },
             { name: '古力娜扎', gender: '女', age: 25 }],
    nums: [1,2,3,4,4]
  },
})
</script>
</body>

</html>

```

4、遍历的时候都加上:key来区分不同数据，提高vue渲染效率

过滤器

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
  <!-- 过滤器常用来处理文本格式化的操作。过滤器可以用在两个地方：双花括号插值和 v-bind 表达式 -->
  <div id="app">
    <ul>
      <li v-for="user in userList">
        {{user.id}} ==> {{user.name}} ==> {{user.gender == 1?"男":"女"}} ==>
        {{user.gender | genderFilter}} ==> {{user.gender | gFilter}}
        <!-- 这里的"|"表示的管道，将user.gender的值交给genderFilter -->
      </li>
    </ul>
  </div>
</body>

```

```
</div>
<script src="../node_modules/vue/dist/vue.js"></script>

<script>
// 全局过滤器
Vue.filter("gFilter", function (val) {
    if (val == 1) {
        return "男~~~";
    } else {
        return "女~~~";
    }
})

let vm = new Vue({
    el: "#app",
    data: {
        userList: [
            { id: 1, name: 'jacky', gender: 1 },
            { id: 2, name: 'peter', gender: 0 }
        ]
    },
    filters: {
        /////////////////////////////////////////////////////////////////// filters 定义局部过滤器，只可以在当前vue实例中使用
        genderFilter(val) {
            if (val == 1) {
                return "男";
            } else {
                return "女";
            }
        }
    }
})
</script>
</body>

</html>
```

组件化



```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>

  <div id="app">
    <button v-on:click="count++>我被点击了 {{count}} 次</button>

    <counter></counter>
    <counter></counter>
    <counter></counter>
    <counter></counter>
    <counter></counter>
    <!-- 使用所定义的组件button-counter -->
    <button-counter></button-counter>
  </div>
  <script src="../node_modules/vue/dist/vue.js"></script>

<script>
  //1、全局声明注册一个组件
  Vue.component("counter", {
    template: `<button v-on:click="count++>我被点击了 {{count}} 次</button>`,
    data() {
      return {
        count: 1
      }
    }
  });

  //2、局部声明一个组件
  const buttonCounter = {
    template: `<button v-on:click="count++>我被点击了 {{count}} 次~~~</button>`,
    data() {
```

```

        return {
            count: 1
        }
    );
}

new Vue({
    el: "#app",
    data: {
        count: 1
    },
    components: {
        //声明所定义的局部组件
        'button-counter': buttonCounter
    }
})
</script>
</body>

</html>

```

- 组件其实也是一个 `Vue` 实例，因此它在定义时也会接收： `data`、`methods`、生命周期函数等
- 不同的是组件不会与页面的元素绑定，否则就无法复用了，因此没有 `el` 属性。
- 但是组件渲染需要 `html` 模板，所以增加了 `template` 属性，值就是 `HTML` 模板
- 全局组件定义完毕，任何 `Vue` 实例都可以直接在 `HTML` 中通过组件名称来使用组件了
- `data` 必须是一个函数，不再是一个对象。

生命周期钩子函数



```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">

```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>

<body>
  <div id="app">
    <span id="num">{{num}}</span>
    <button @click="num++">赞! </button>
    <h2>{{name}}, 有{{num}}个人点赞</h2>
  </div>

  <script src="../node_modules/vue/dist/vue.js"></script>

  <script>
    let app = new Vue({
      el: "#app",
      data: {
        name: "张三",
        num: 100
      },
      methods: {
        show() {
          return this.name;
        },
        add() {
          this.num++;
        }
      },
      beforeCreate() {
        console.log("=====beforeCreate=====");
        console.log("数据模型未加载: " + this.name, this.num);
        console.log("方法未加载: " + this.show());
        console.log("html模板未加载: " + document.getElementById("num"));
      },
      created: function () {
        console.log("=====created=====");
        console.log("数据模型已加载: " + this.name, this.num);
        console.log("方法已加载: " + this.show());
        console.log("html模板已加载: " + document.getElementById("num"));
        console.log("html模板未渲染: " + document.getElementById("num").innerText);
      },
      beforeMount() {
        console.log("=====beforeMount=====");
        console.log("html模板未渲染: " + document.getElementById("num").innerText);
      },
    })
  </script>
```

```
mounted() {
    console.log("=====mounted=====");
    console.log("html模板已渲染: " + document.getElementById("num").innerText);
},
beforeUpdate() {
    console.log("=====beforeUpdate=====");
    console.log("数据模型已更新: " + this.num);
    console.log("html模板未更新: " + document.getElementById("num").innerText);
},
updated() {
    console.log("=====updated=====");
    console.log("数据模型已更新: " + this.num);
    console.log("html模板已更新: " + document.getElementById("num").innerText);
}
});
</script>
</body>

</html>
```

13. element ui

官网: <https://element.eleme.cn/#/zh-CN/component/installation>

安装



```
npm i element-ui -S
```

在 main.js 中写入以下内容:

```
import ElementUI from 'element-ui'  
import 'element-ui/lib/theme-chalk/index.css';  
  
Vue.use(ElementUI);
```

14. 递归树形结构获取数据

在注册中心中“product”命名空间中，创建“gulimall-product.yml”配置文件：



The screenshot shows a configuration management interface with the following details:

- Namespace:** product
- Search Results:** 共查询到 1 条满足要求的配置。
- Table Headers:** Data ID, Group, 归属应用:, 操作
- Table Data:** A single row for "gulimall-product.yml" with Group: DEFAULT_GROUP.
- Buttons:** 删除 (Delete), 导出选中的配置 (Export Selected Configuration), 克隆 (Clone), 详情 (Details), 示例代码 (Example Code), 编辑 (Edit), 删 (Delete), 更多 (More)
- Pagination:** 每页显示: 10

将“application.yml”内容拷贝到该配置文件中

```
server:  
  port: 10000  
  
spring:  
  datasource:  
    #MySQL配置  
    driverClassName: com.mysql.cj.jdbc.Driver  
    url: jdbc:mysql://192.168.137.14:3306/gulimall_pms?  
      useUnicode=true&characterEncoding=UTF-8&useSSL=false  
    username: root  
    password: root  
  application:
```

```
name: gulimall-product
cloud:
nacos:
discovery:
server-addr: 192.168.137.14:8848

mybatis-plus:
global-config:
db-config:
id-type: auto
mapper-locations: classpath:/mapper/**/*.xml
```

在本地创建“bootstrap.properties”文件，指明配置中心的位置和使用到的配置文件：

```
spring.application.name=gulimall-product
spring.cloud.nacos.config.server-addr=192.168.137.14:8848
spring.cloud.nacos.config.namespace=3c50ffaa-010b-4b59-9372-902e35059232
spring.cloud.nacos.config.extension-configs[0].data-id=gulimall-product.yml
spring.cloud.nacos.config.extension-configs[0].group=DEFAULT_GROUP
spring.cloud.nacos.config.extension-configs[0].refresh=true
```

然后启动gulimall-product，查看到该服务已经出现在了nacos的注册中心中了

修改“com.baomidou.gulimall.product.service.CategoryService”类，添加如下代码：

```
/**
 * 列表
 */
@RequestMapping("/list/tree")
public List<CategoryEntity> list(){
    List<CategoryEntity> categoryEntities = categoryService.listWithTree();

    return categoryEntities;
}
```

测试：<http://localhost:10000/product/category/list/tree>

localhost:10000/product/category/list

JSON 原始数据 头
保存 复制 美化输出

```
{"msg":"success","code":0,"CategoryEntitys":[{"catId":1,"name":"图书、音像、电子书刊","parentCid":0,"catLevel":1,"showStatus":1,"sort":0,"icon":null,"productUnit":null,"productCount":0},{"catId":3,"catLevel":1,"name":"数码","parentCid":0,"catLevel":1,"showStatus":1,"sort":0,"icon":null,"productUnit":null,"productCount":0}, {"catId":4,"name":"数码","parentCid":0,"catLevel":1,"showStatus":1,"sort":0,"icon":null,"productUnit":null,"productCount":0}, {"catId":6,"name":"厨具","parentCid":0,"catLevel":1,"showStatus":1,"sort":0,"icon":null,"productUnit":null,"productCount":0}, {"catId":7,"name":"厨具","parentCid":0,"catLevel":1,"showStatus":1,"sort":0,"icon":null,"productUnit":null,"productCount":0}], "parentCid":0, "catLevel":1, "showStatus":1, "sort":0, "icon":null, "productUnit":null, "productCount":0, "catId":0}
```

如何区别是哪种分类级别?

答: 可以通过分类的parent_cid来进行判断, 如果是一级分类, 其值为0.

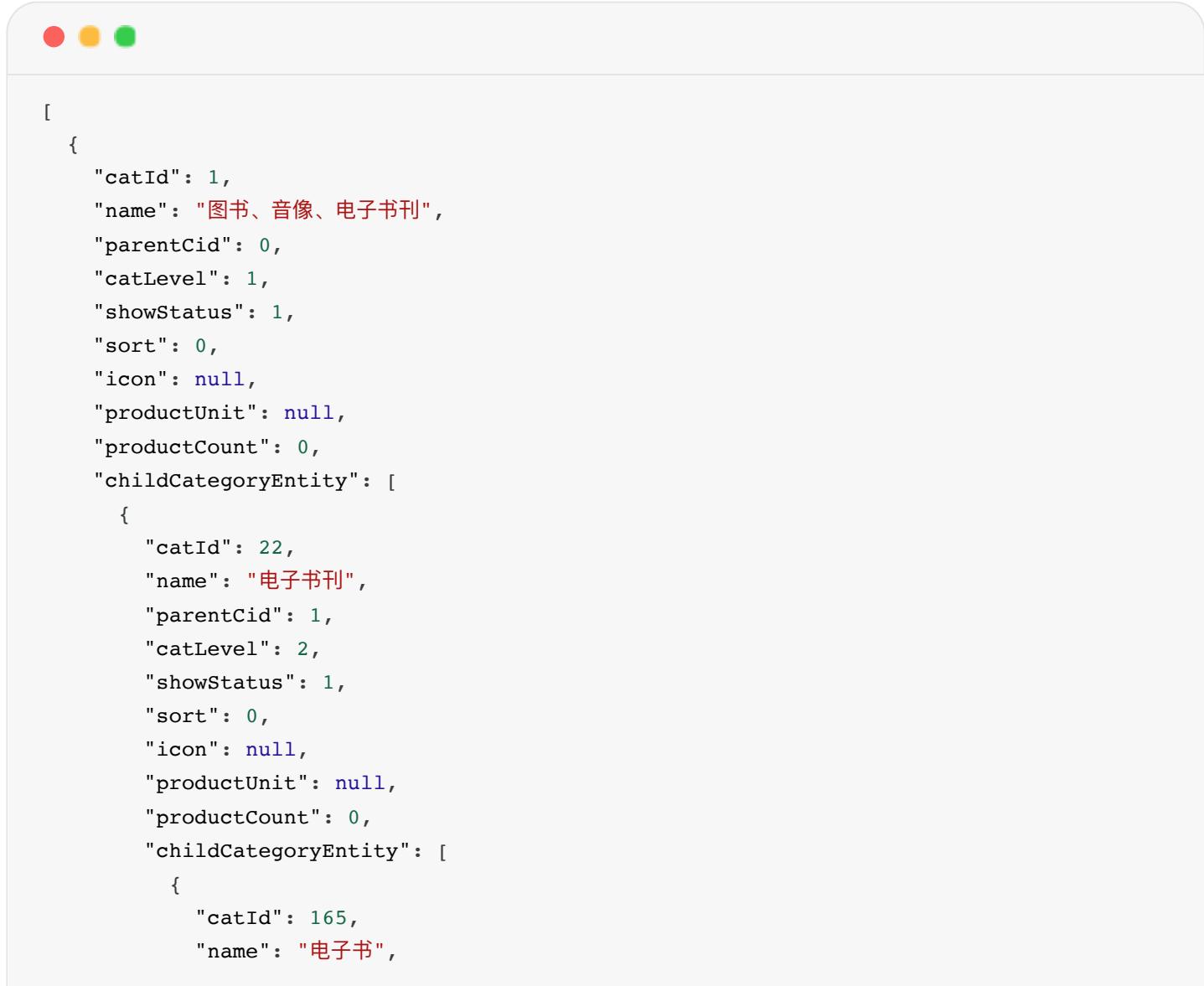
```
/*
 * 列表
 */
@RequestMapping("/list/tree")
public List<CategoryEntity> list(){
    List<CategoryEntity> categoryEntities = categoryService.listWithTree();
    //找到所有的一级分类
    List<CategoryEntity> level1Menus = categoryEntities.stream()
        .filter(item -> item.getParentCid() == 0)
        .map(menu->{
            menu.setChildCategoryEntity(getChildrens(menu,categoryEntities));
            return menu;
        })
        .sorted((menu1, menu2) -> {
            return (menu1.getSort() == null ? 0:menu1.getSort())-
                (menu2.getSort() == null?0:menu2.getSort());
        })
        .collect(Collectors.toList());
    return level1Menus;
}

public List<CategoryEntity> getChildrens(CategoryEntity root,List<CategoryEntity> all){
    List<CategoryEntity> childrens = all.stream().filter(item -> {
```

```
        return item.getParentCid() == root.getCatId();
    }).map(item -> {
        item.setChildCategoryEntity(getChildrens(item, all));
        return item;
}).sorted((menu1, menu2) -> {
    return (menu1.getSort() ==null ? 0:menu1.getSort())- (menu2.getSort() ==null?
0:menu2.getSort());
}).collect(Collectors.toList());

return childrens;
}
```

下面是得到的部分JSON数据



The screenshot shows a mobile application interface with a light gray header bar featuring three colored dots (red, yellow, green). Below the header is a white list view containing a single item. The item is represented by a JSON object:

```
[  
  {  
    "catId": 1,  
    "name": "图书、音像、电子书刊",  
    "parentCid": 0,  
    "catLevel": 1,  
    "showStatus": 1,  
    "sort": 0,  
    "icon": null,  
    "productUnit": null,  
    "productCount": 0,  
    "childCategoryEntity": [  
      {  
        "catId": 22,  
        "name": "电子书刊",  
        "parentCid": 1,  
        "catLevel": 2,  
        "showStatus": 1,  
        "sort": 0,  
        "icon": null,  
        "productUnit": null,  
        "productCount": 0,  
        "childCategoryEntity": [  
          {  
            "catId": 165,  
            "name": "电子书",  
          }  
        ]  
      }  
    ]  
  }  
]
```

```
"parentCid": 22,
"catLevel": 3,
"showStatus": 1,
"sort": 0,
"icon": null,
"productUnit": null,
"productCount": 0,
"childCategoryEntity": []
},
{
"catId": 166,
"name": "网络原创",
"parentCid": 22,
"catLevel": 3,
"showStatus": 1,
"sort": 0,
"icon": null,
"productUnit": null,
"productCount": 0,
"childCategoryEntity": []
},
{
"catId": 167,
"name": "数字杂志",
"parentCid": 22,
"catLevel": 3,
"showStatus": 1,
"sort": 0,
"icon": null,
"productUnit": null,
"productCount": 0,
"childCategoryEntity": []
},
{
"catId": 168,
"name": "多媒体图书",
"parentCid": 22,
"catLevel": 3,
"showStatus": 1,
"sort": 0,
"icon": null,
"productUnit": null,
"productCount": 0,
"childCategoryEntity": []
}
]
```

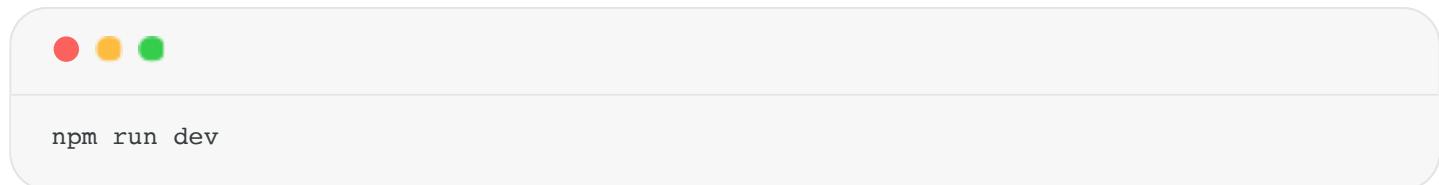
```
},
{
  "catId": 23,
  "name": "音像",
  "parentCid": 1,
  "catLevel": 2,
  "showStatus": 1,
  "sort": 0,
  "icon": null,
  "productUnit": null,
  "productCount": 0,
  "childCategoryEntity": [
    {
      "catId": 169,
      "name": "音乐",
      "parentCid": 23,
      "catLevel": 3,
      "showStatus": 1,
      "sort": 0,
      "icon": null,
      "productUnit": null,
      "productCount": 0,
      "childCategoryEntity": []
    },
    {
      "catId": 170,
      "name": "影视",
      "parentCid": 23,
      "catLevel": 3,
      "showStatus": 1,
      "sort": 0,
      "icon": null,
      "productUnit": null,
      "productCount": 0,
      "childCategoryEntity": []
    },
    {
      "catId": 171,
      "name": "教育音像",
      "parentCid": 23,
      "catLevel": 3,
      "showStatus": 1,
      "sort": 0,
      "icon": null,
      "productUnit": null,
      "productCount": 0,
      "childCategoryEntity": []
    }
  ]
}
```

```
        "childCategoryEntity": []
    }
]
},
{

```

启动后端项目renren-fast

启动前端项目renren-fast-vue:



访问: <http://localhost:8001/#/login>

创建一级菜单:



创建完成后，在后台的管理系统中会创建一条记录:

The screenshot shows two main parts. On the left is a tree view of database tables under 'gulimall_admin'. A red box highlights the 'sys_menu' table. On the right is a table titled 'sys_menu' with columns: menu_id, parent_id, name, url, perms, type, icon, and order_num. A red box highlights the last row, which has a menu_id of 31, a parent_id of 0, a name of '商品系统', and a type of 'editor'.

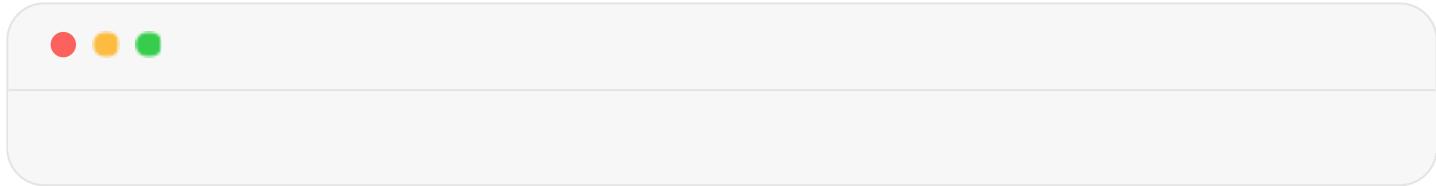
menu_id	parent_id	name	url	perms	type	icon	order_num
5	1	SQL监控	http://localhost:8080/renr (Null)	1 sql	1		4
6	1	定时任务	job/schedule (Null)	1 job	1		5
7	6	查看	(Null)	sys:schedule:list,sys:sched	2	(Null)	0
8	6	新增	(Null)	sys:schedule:save	2	(Null)	0
9	6	修改	(Null)	sys:schedule:update	2	(Null)	0
10	6	删除	(Null)	sys:schedule:delete	2	(Null)	0
11	6	暂停	(Null)	sys:schedule:pause	2	(Null)	0
12	6	恢复	(Null)	sys:schedule:resume	2	(Null)	0
13	6	立即执行	(Null)	sys:scheduler:run	2	(Null)	0
14	6	日志列表	(Null)	sys:schedule:log	2	(Null)	0
15	2	查看	(Null)	sys:user:list,sys:user:info	2	(Null)	0
16	2	新增	(Null)	sys:user:save,sys:role:sele	2	(Null)	0
17	2	修改	(Null)	sys:user:update,sys:role:sele	2	(Null)	0
18	2	删除	(Null)	sys:user:delete	2	(Null)	0
19	3	查看	(Null)	sys:role:list,sys:role:info	2	(Null)	0
20	3	新增	(Null)	sys:role:save,sys:menu:list	2	(Null)	0
21	3	修改	(Null)	sys:role:update,sys:menu:	2	(Null)	0
22	3	删除	(Null)	sys:role:delete	2	(Null)	0
23	4	查看	(Null)	sys:menu:list,sys:menu:info	2	(Null)	0
24	4	新增	(Null)	sys:menu:save,sys:menu:	2	(Null)	0
25	4	修改	(Null)	sys:menu:update,sys:menu:	2	(Null)	0
26	4	删除	(Null)	sys:menu:delete	2	(Null)	0
27	1	参数管理	sys/config	sys:config:list,sys:config:in	1 config		6
29	1	系统日志	sys/log	sys:log:list	1 log		7
30	1	文件上传	oss/oss	sys:oss:all	1 oss		6
31	0	商品系统			0 editor		0

然后创建子菜单：



创建renren-fast-vue\src\views\modules\product目录，子所以是这样来创建，是因为product/category，对应于product-category

在该目录下，新建“category.vue”文件：



刷新页面出现404异常，查看请求发现，请求的是“<http://localhost:8080/renren-fast/product/category/list/tree>”

The screenshot shows the Network tab of the developer tools with the following details:

- 请求网址:** <http://localhost:8080/renren-fast/product/category/list/tree>
- 远程地址:** 127.0.0.1:8080
- 状态码:** 404
- 版本:** HTTP/1.1
- Referer 政策:** no-referrer-when-downgrade
- 响应头 (401 字节):**
 - Access-Control-Allow-Credentials: true
 - Access-Control-Allow-Origin: <http://localhost:8001>
 - Connection: keep-alive
 - Content-Type: application/json
 - Date: Sat, 25 Apr 2020 09:35:08 GMT
 - Keep-Alive: timeout=5

这个请求是不正确的，正确的请求是：<http://localhost:10000/product/category/list/tree>,

修正这个问题：

替换“static\config\index.js”文件中的“window.SITE_CONFIG['baseUrl']”

替换前：

```
window.SITE_CONFIG[ 'baseUrl' ] = 'http://localhost:8080/renren-fast';
```

替换后：

```
window.SITE_CONFIG[ 'baseUrl' ] = 'http://localhost:88/api';
```

<http://localhost:88>，这个地址是我们网关微服务的接口。

这里我们需要通过网关来完成路径的映射，因此将renren-fast注册到nacos注册中心中，并添加配置中心

```
application:
  name: renren-fast
cloud:
nacos:
  discovery:
    server-addr: 192.168.137.14:8848

config:
  name: renren-fast
  server-addr: 192.168.137.8848
  namespace: ee409c3f-3206-4a3b-ba65-7376922a886d
```

配置网关路由，前台的所有请求都是经由“<http://localhost:88/api>”来转发的，在“gulimall-gateway”中添加路由规则：

```
- id: admin_route
  uri: lb://renren-fast
  predicates:
    - Path=/api/**
```

但是这样做也引入了另外的一个问题，再次访问：<http://localhost:8001/#/login>，发现验证码不再显示：

分析原因：

1. 现在的验证码请求路径为，<http://localhost:88/api/captcha.jpg?uuid=69c79f02-d15b-478a-8465-a07fd09001e6>
2. 原始的验证码请求路径：<http://localhost:8001/renren-fast/captcha.jpg?uuid=69c79f02-d15b-478a-8465-a07fd09001e6>

在admin_route的路由规则下，在访问路径中包含了“api”，因此它会将它转发到renren-fast，网关在转发的时候，会使用网关的前缀信息，为了能够正常的取得验证码，我们需要对请求路径进行重写

关于请求路径重写：

6.16. The RewritePath - GatewayFilter Factory

The `RewritePath` `GatewayFilter` factory takes a path `regexp` parameter and a `replacement` parameter. This uses Java regular expressions for a flexible way to rewrite the request path. The following listing configures a `RewritePath` `GatewayFilter`:

Example 41. application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: rewritepath_route
          uri: https://example.org
          predicates:
            - Path=/foo/**
          filters:
            - RewritePath=/red(?<segment>/?.*) , ${segment}
```

For a request path of `/red/blue`, this sets the path to `/blue` before making the downstream request. Note that the `$` should be replaced with `\$` because of the YAML specification.

修改“admin_route”路由规则：

```
- id: admin_route
  uri: lb://renren-fast
  predicates:
    - Path=/api/**
  filters:
    - RewritePath=/api/(?<segment>/?.*) , /renren-fast/${segment}
```

再次访问：<http://localhost:8001/#/login>, 验证码能够正常的加载了。

但是很不幸新的问题又产生了，访问被拒绝了

问题描述：已拦截跨源请求：同源策略禁止读取位于 <http://localhost:88/api/sys/login> 的远程资源。 (原因：CORS 头缺少 'Access-Control-Allow-Origin') 。

问题分析：这是一种跨域问题。访问的域名和端口和原来的请求不同，请求就会被限制

跨域

- 跨域：指的是浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是浏览器对javascript 加的安全限制。
- 同源策略：是指协议，域名，端口都要相同，其中有一个不同都会产生跨域；

URL	说明	是否允许通信
http://www.a.com/a.js http://www.a.com/b.js	同一域名下	允许
http://www.a.com/lab/a.js http://www.a.com/script/b.js	同一域名下不同文件夹	允许
http://www.a.com:8000/a.js http://www.a.com/b.js	同一域名，不同端口	不允许
http://www.a.com/a.js https://www.a.com/b.js	同一域名，不同协议	不允许
http://www.a.com/a.js http://70.32.92.74/b.js	域名和IP对应	不允许
http://www.a.com/a.js http://script.a.com/b.js	主域相同，子域不同	不允许
http://www.a.com/a.js http://a.com/b.js	同一域名，不同二级域名（同上）	不允许（cookie这种情况下也不允许访问）
http://www.cnblogs.com/a.js http://www.a.com/b.js	不同域名	不允许

跨域流程：

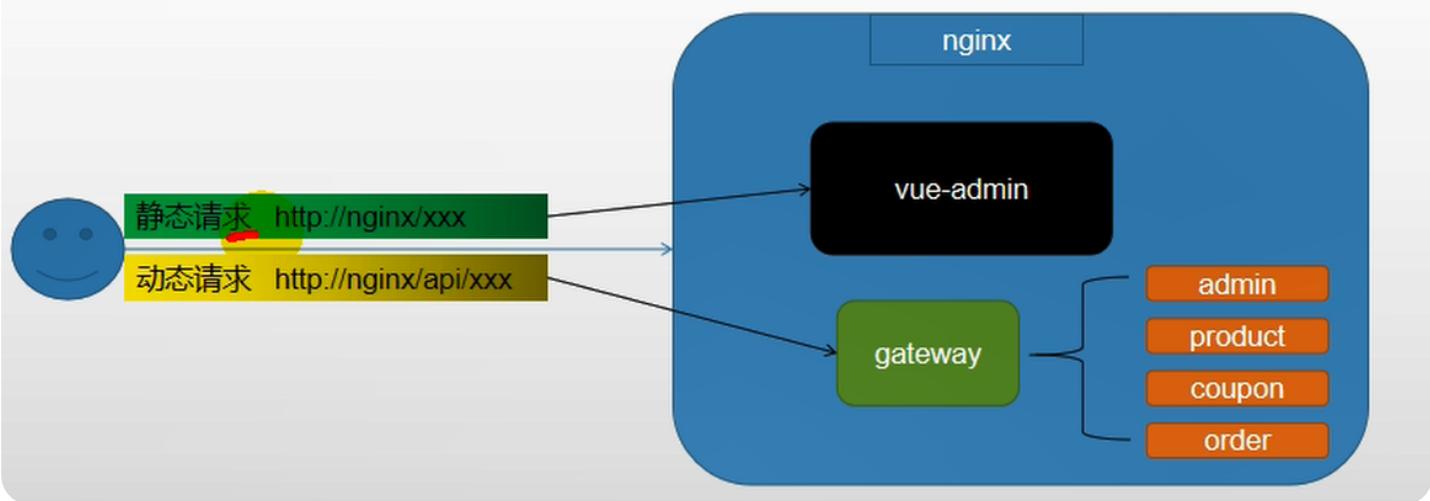
跨域流程

非简单请求 (PUT、DELETE) 等，需要先发送预检请求



https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS

解决跨域-（一）使用nginx部署为同一域



解决跨域-（二）配置当次请求允许跨域

- 1、添加响应头

- Access-Control-Allow-Origin: 支持哪些来源的请求跨域
- Access-Control-Allow-Methods: 支持哪些方法跨域
- Access-Control-Allow-Credentials: 跨域请求默认不包含cookie，设置为true可以包含cookie
- Access-Control-Expose-Headers: 跨域请求暴露的字段
 - CORS请求时，XMLHttpRequest对象的getResponseHeader()方法只能拿到6个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想拿到其他字段，就必须在Access-Control-Expose-Headers里面指定。
- Access-Control-Max-Age: 表明该响应的有效时间为多少秒。在有效时间内，浏览器无须为同一请求再次发起预检请求。请注意，浏览器自身维护了一个最大有效时间，如果该首部字段的值超过了最大有效时间，将不会生效。

解决方法：在网关中定义“GulimallCorsConfiguration”类，该类用来做过滤，允许所有的请求跨域。

```
@Configuration
public class GulimallCorsConfiguration {

    @Bean
    public CorsWebFilter corsWebFilter(){
        UrlBasedCorsConfigurationSource source=new UrlBasedCorsConfigurationSource();
        CorsConfiguration corsConfiguration = new CorsConfiguration();
        corsConfiguration.addAllowedHeader("*");
        corsConfiguration.addAllowedMethod("*");
        corsConfiguration.addAllowedOrigin("*");
        corsConfiguration.setAllowCredentials(true);

        source.registerCorsConfiguration("/**",corsConfiguration);
        return new CorsWebFilter(source);
    }
}
```

再次访问：<http://localhost:8001/#/login>

请求网址: http://localhost:88/api/sys/login
 请求方法: OPTIONS
 远程地址: 127.0.0.1:88
 状态码: 200 OK ⓘ
 版本: HTTP/1.1
 Referrer 政策: no-referrer-when-downgrade

响应头 (294 字节)

- ⑦ Access-Control-Allow-Credentials: true
- ⑦ Access-Control-Allow-Headers: content-type, token
- ⑦ Access-Control-Allow-Methods: POST
- ⑦ Access-Control-Allow-Origin: http://localhost:8001

15 个请求 | 已传输 150.12 KB / 4.32 KB | 完成: 6.85 秒 | DOMContentLoaded: 1.43 秒 | load: 2.40 秒

过滤输出

```
./src/router/index.js
./src/main.js
multi (webpack)-dev-server/client?http://localhost:8001 webpack/hot/dev-server babel-nolyfill ./src/main.js
```

已拦截跨源请求: 同源策略禁止读取位于 http://localhost:88/api/sys/Login 的远程资源。(原因: 不允许有多个 'Access-Control-Allow-Origin' CORS 头) [详细了解]

<http://localhost:8001/renre>已拦截跨源请求: 同源策略禁止读取位于 <http://localhost:88/api/sys/login> 的远程资源。
 (原因: 不允许有多个 'Access-Control-Allow-Origin' CORS 头) n-fast/captcha.jpg?uuid=69c79f02-d15b-478a-8465-a07fd09001e6

出现了多个请求，并且也存在多个跨源请求。

为了解决这个问题，需要修改renren-fast项目，注释掉“io.renren.config.CorsConfig”类。然后再次进行访问。

在显示分类信息的时候，出现了404异常，请求的<http://localhost:88/api/product/category/list/tree>不存在

请求网址: http://localhost:88/api/product/category/list/tree
 请求方法: GET
 远程地址: 127.0.0.1:88
 状态码: 404 Not Found ⓘ
 版本: HTTP/1.1
 Referrer 政策: no-referrer-when-downgrade

消息头

参数 响应 耗时 堆栈跟踪

18 个请求 | 已传输 98.31 KB / 5.34 KB | 完成: 2.23 秒 | DOMContentLoaded: 1.45 秒 | load: 1.65 秒

过滤输出

错误 警告 日志 信息 调试 CSS JS

这是因为网关上所做的路径映射不正确，映射后的路径为<http://localhost:8001/renren-fast/product/category/list/tree>

但是只有通过<http://localhost:10000/product/category/list/tree>路径才能够正常访问，所以会报404异常。

解决方法就是定义一个product路由规则，进行路径重写：

```
- id: product_route
  uri: lb://gulimall-product
  predicates:
    - Path=/api/product/**
  filters:
    - RewritePath=/api/(?<segment>/?.*)/$\{segment}
```

在路由规则的顺序上，将精确的路由规则放置到模糊的路由规则的前面，否则的话，精确的路由规则将不会被匹配到，类似于异常体系中try catch子句中异常的处理顺序。

15. 删除数据

添加delete和append标识，并且增加复选框

```
● ○ ●

<el-tree
  :data="menus"
  show-checkbox //显示复选框
  :props="defaultProps"
  :expand-on-click-node="false" //设置节点点击时不展开
  node-key="catId"
>
  <span class="custom-tree-node" slot-scope="{ node, data }">
    <span>{{ node.label }}</span>
    <span>
      <el-button v-if="node.level <= 2" type="text" size="mini" @click="() =>
append(data)">Append</el-button>
      <el-button
        v-if="node.childNodes.length == 0"
        type="text"
        size="mini"
        @click="() => remove(node, data)"
        >Delete</el-button>
    </span>
  </span>
</el-tree>
```

测试删除数据，打开postman输入“<http://localhost:88/api/product/category/delete>”，请求方式设置为POST，为了比对效果，可以在删除之前查询数据库的pms_category表：

父类ID	父类名称	类目ID	类目名称	排序	状态	创建时间	更新时间
1000	毛衣链	107	3	1	0 (Null)	(Null)	0

由于delete请求接收的是一个数组，所以这里使用JSON方式，传入了一个数组：

The screenshot shows the Postman interface. The method is set to POST, and the URL is http://localhost:88/api/product/category/delete. The 'Body' tab is selected, and the content type is set to raw JSON. The JSON payload is a single-element array: [1000].

再次查询数据库能够看到cat_id为1000的数据已经被删除了。

修改“com.bigdata.gulimall.product.controller.CategoryController”类，添加如下代码：

```
@RequestMapping("/delete")
public R delete(@RequestBody Long[] catIds){
    //删除之前需要判断待删除的菜单那是否被别的地方所引用。
//    categoryService.removeByIds(Arrays.asList(catIds));

    categoryService.removeMenuByIds(Arrays.asList(catIds));
    return R.ok();
}
```

com.bigdata.gulimall.product.service.impl.CategoryServiceImpl

```
@Override
public void removeMenuByIds(List<Long> asList) {
    //TODO 检查当前的菜单是否被别的地方所引用
    categoryDao.deleteBatchIds(asList);
}
```

然而多数时候，我们并不希望删除数据，而是标记它被删除了，这就是逻辑删除；

可以设置show_status为0，标记它已经被删除。

cat_id	name	parent_cid	cat_level	show_status	sort	icon	product_unit	product
1414	小型车（二手）	164	3	1	0	(NULL)	(NULL)	
1415	紧凑型车（二手）	164	3	1	0	(NULL)	(NULL)	
1416	中型车（二手）	164	3	1	0	(NULL)	(NULL)	
1417	中大型车（二手）	164	3	1	0	(NULL)	(NULL)	
1418	豪华车（二手）	164	3	1	0	(NULL)	(NULL)	
1419	MPV（二手）	164	3	1	0	(NULL)	(NULL)	
1420	SUV（二手）	164	3	1	0	(NULL)	(NULL)	
1421	跑车（二手）	164	3	1	0	(NULL)	(NULL)	
1422	皮卡（二手）	164	3	1	0	(NULL)	(NULL)	
1423	面包车（二手）	164	3	1	0	(NULL)	(NULL)	
1431	dsd323	1	2	1	(NULL)	(NULL)	(NULL)	
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	

mybatis-plus的逻辑删除：

The screenshot shows the official MyBatis-Plus website at mp.baomidou.com/guide/logic-delete.html. The sidebar on the left has a red box around the 'Logic Delete' link under the 'Plugin Extension' section. The main content area is titled '逻辑删除' (Logic Delete) and includes configuration instructions for Spring Boot. A red box highlights the configuration code in the 'SpringBoot 配置方式:' section:

```
mybatis-plus:
  global-config:
    db-config:
      logic-delete-value: 1 # 逻辑已删除值(默认为 1)
      logic-not-delete-value: 0 # 逻辑未删除值(默认为 0)
```

- application.yml 加入配置(如果你的默认值和mp默认的一样,该配置可无):

配置全局的逻辑删除规则，在“src/main/resources/application.yml”文件中添加如下内容：

```
mybatis-plus:
  global-config:
    db-config:
      id-type: auto
      logic-delete-value: 1
      logic-not-delete-value: 0
```

修改“com.baomidou.mybatisplus.entity.CategoryEntity”类，添加上@TableLogic，表明使用逻辑删除：

```
/**  
 * 是否显示[0-不显示, 1显示]  
 */  
@TableLogic(value = "1", delval = "0")  
private Integer showStatus;
```

然后在POSTMan中测试一下是否能够满足需要。另外在“src/main/resources/application.yml”文件中，设置日志级别，打印出SQL语句：

```
logging:  
  level:  
    com.bigdata.gulimall.product: debug
```

打印的日志：

```
==> Preparing: UPDATE pms_category SET show_status=0 WHERE cat_id IN ( ? ) AND  
show_status=1  
==> Parameters: 1431(Long)  
<==     Updates: 1  
get changedGroupKeys: []
```

16. 菜单拖动

同一个菜单内拖动	正常
拖动到父菜单的前面或后面	正常
拖动到父菜单同级的另外一个菜单中	正常

关注的焦点在于，拖动到目标节点中，使得目标节点的catlevel+deep小于3即可。拖动到目标节点前后的条件是，使得

拖动菜单时需要修改顺序和级别

需要考虑两种类型节点的catLevel

一种关系是：如果是同一个节点下的子节点的前后移动，则不需要修改其catLevel

如果是拖动到另外一个节点内或父节点中，则要考虑修改其catLevel

如果拖动到与父节点平级的节点关系中，则要将该拖动的节点的catLevel，设置为兄弟节点的Level，

先考虑parentCid还是先考虑catLevel?

两种关系在耦合

另外还有一种是前后拖动的情况

哪个范围最大？

肯定是拖动类型关系最大，

如果是前后拖动，则拖动后需要看待拖动节点的层级和设置待拖动节点的parentId，

如果待拖动节点和目标节点的层级相同，则认为是同级拖动，只需要修改节点的先后顺序即可；

否则认为是跨级拖动，则需要修改层级和重新设置parentID

如果

以拖动类型来分，并不合适，比较合适的是跨级拖动和同级拖动

如何判断是跨级拖动还是同级拖动，根据拖动的层级来看，如果是同级别的拖动，只需要修改先后顺序即可，但是这样也会存在一个问题，就是当拖动到另外一个分组下的同级目录中，显然也需要修改parentID，究竟什么样的模型最好呢？

另外也可以判断在跨级移动时，跨级后的parentID是否相同，如果不相同，则认为是在不同目录下的跨级移动需要修改parentID。

顺序、catLevel和parentID

同级移动：

- (1) 首先判断待移动节点和目标节点的catLevel是否相同，
- (2) 相同则认为是同级移动，

如果此时移动后目标节点的parentID和待移动节点的相同，但是移动类型是前后移动，只需要调整顺序即可，此时移动类型是inner，则需要修改catLevel和parentID和顺序

如果此时移动后目标节点的parentID和待移动节点的不相同，但是移动类型是前后移动，则需要调整顺序和parentID，此时移动类型是inner，则需要修改catLevel和parentID和顺序

通过这两步的操作能看到一些共性，如果抽取移动类型作为大的分类，则在这种分类下，

如果是前后移动，则分为下面几种情况：

同级别下的前后移动：界定标准为catLevel相同，但是又可以分为parentID相同和parentID不同，parent相同时，只需要修改顺序即可；parentID不同时，需要修改parentID和顺序

不同级别下的前后移动：界定标准为catLevel不同，此时无论如何都要修改parentID，顺序和catLevel

如果是inner类型移动，则分为一下几种情况。

此时不论是同级inner，还是跨级inner，都需要修改parentID，顺序和catLevel

哪种情况需要更新子节点呢？

那就要看要拖拽的节点是否含有子节点，如果有子节点，则需要更新子节点的catLevel，不需要更新它之间的顺序和parentID，只需要更新catLevel即可。这种更新子节点的Level应该归类，目前的目标是只要有子节点就更新它的catLevel，

(2) 如果待移动节点和目标节点的catLevel不同，则认为是跨级移动。如果是移动到父节点中，则需要设置catLevel, parentID和顺序。此时需要分两种情况来考虑，如果是移动到父节点中，则需要设置catLevel, parentID和顺序，如果是移动到兄弟节点中，则需要设置

包含移动到父节点同级目录，兄弟节点中。

设置菜单拖动开关



```
<el-switch v-model="draggable" active-text="开启拖拽" inactive-text="关闭拖拽"></el-switch>
```

但是现在存在的一个问题是在每次拖拽的时候，都会发送请求，更新数据库这样频繁的与数据库交互，现在想要实现一个拖拽过程中不更新数据库，拖拽完成后，统一提交拖拽后的数据。

现在还存在一个问题，如果是将一个菜单连续的拖拽，最终还放到了原来的位置，但是updateNode中却出现了很多节点更新信息，这样显然也是一个问题。

批量删除



```
<el-button type="danger" plain size="small" @click="batchDelete">批量删除</el-button>
```



```
//批量删除
batchDelete() {
  let checkNodes = this.$refs.menuTree.getCheckedNodes();
```

```
// console.log("被选中的节点: ", checkNodes);

let catIds = [];
for (let i = 0; i < checkNodes.length; i++) {
    catIds.push(checkNodes[i].catId);
}

this.$confirm(`确定要删除?`, "提示", {
    confirmButtonText: "确定",
    cancelButtonText: "取消",
    type: "warning"
})
.then(() => {
    this.$http({
        url: this.$http.adornUrl("/product/category/delete"),
        method: "post",
        data: this.$http.adornData(catIds, false)
    }).then(({ data }) => {
        this.$message({
            message: "菜单批量删除成功",
            type: "success"
        });

        //重新刷新页面
        this.getMeus();
    });
});

})
.catch(() => {
    //取消删除
});
},

```

17. 品牌管理菜单

新增

操作成功

X

类型 目录 菜单 按钮

* 菜单名称

* 上级菜单

菜单路由

授权标识

排序号

菜单图标

取消

确定

(2) 将“逆向工程得到的resources\src\views\modules\product文件拷贝到gulimall\renren-fast-vue\src\views\modules\product目录下，也就是下面的两个文件

brand.vue brand-add-or-update.vue

但是显示的页面没有新增和删除功能，这是因为权限控制的原因，

参数名

<input type="checkbox"/>	品牌id	品牌名	品牌logo地址	介绍	显示状态[0-不显示; 1-显示]	检索首字母	排序	操作
<input type="checkbox"/>	1	huawei		huawei honor8				修改 删除
<input type="checkbox"/>	2			huawei honor8				修改 删除

共 0 条 < **1** > 前往 页

```

<el-button v-if="isAuth('product:brand:save')" type="primary" @click="addOrUpdateHandle()>新增</el-button>
<el-button v-if="isAuth('product:brand:delete')" type="danger" @click="deleteHandle()" :disabled="dataListSelections.length <= 0">批量删除</el-button>

```

查看“isAuth”的定义位置：

7 文件中有 28 个结果 - 在编辑器中打开

- JS main.js src (3) import { isAuth } from ...
- JS index.js src\utils (1) export funct...

```

13 /**
14 * 是否有权限
15 * @param {*} key
16 */
17 export function isAuth (key) {
18   return JSON.parse(sessionStorage.getItem('permissions') || '[]').indexOf(key) !== -1 || false
19 }
20 */
21 /**
22 */

```

它是在“index.js”中定义，现在将它设置为返回值为true，即可显示添加和删除功能。

再次刷新页面能够看到，按钮已经出现了：

参数名

<input type="checkbox"/>	品牌id	品牌名	品牌logo地址	介绍	显示状态[0-不显示; 1-显示]	检索首字母	排序	操作
<input type="checkbox"/>	1	huawei		huawei honor8				修改 删除
<input type="checkbox"/>	2			huawei honor8				修改 删除

共 0 条 < **1** > 前往 页

添加“显示状态按钮”

brand.vue

```
<template slot-scope="scope">
  <el-switch
    v-model="scope.row.showStatus"
    active-color="#13ce66"
    inactive-color="#ff4949"
    @change="updateBrandStatus(scope.row)"
    :active-value = "1"
    :inactive-value = "0"
  ></el-switch>
</template>
```

brand-add-or-update.vue

```
<el-form-item label="显示状态" prop="showStatus">
  <el-switch v-model="dataForm.showStatus" active-color="#13ce66" inactive-
  color="#ff4949"></el-switch>
</el-form-item>
```

```
//更新开关的状态
updateBrandStatus(data) {
  console.log("最新状态", data);
  let {brandId, showStatus} = data;
  this.$http({
    url: this.$http.adornUrl("/product/brand/update"),
    method: "post",
    data: this.$http.adornData({brandId, showStatus}, false)
  }).then(({ data }) => {
    this.$message({
      message: "状态更新成功",
      type: "success"
    });
  })
}
```

```
});  
},
```

添加上传

和传统的单体应用不同，这里我们选择将数据上传到分布式文件服务器上。

这里我们选择将图片放置到阿里云上，使用对象存储。

阿里云上使用对象存储方式：



创建Bucket

创建 Bucket

⑦ 创建存储空间 >

注意: Bucket 创建成功后, 您所选择的 存储类型、区域 不支持变更。

Bucket 名称

gulimall-images

15/63 ✓

区域

华东2 (上海)



相同区域内的产品内网可以互通; 订购后不支持更换区域, 请谨慎选择。

您在该区域下没有可用的 存储包、流量包。建议您购买资源包享受更多优惠, 点击 购买。

Endpoint

oss-cn-shanghai.aliyuncs.com

存储类型

标准存储

低频访问

归档存储

低频访问: 数据长期存储、较少访问, 存储单价低于标准类型。

如何选择适合您的存储类型?

同城冗余存储 Hot

启用

关闭

OSS 将您的数据以冗余的方式存储在同一区域 (Region) 的 3 个可用区 (Zone) 中。提供机房级容灾能力。更多详情请参见 同城冗余存储。

同城冗余存储能提高您的数据可用性, 同时会采用相对较高的计费标准。请查看 价格详情。同城冗余存储属性开启后, 将不支持关闭。

版本控制 Hot

开通

不开通

开启版本控制特性后, 针对数据的覆盖和删除操作将会以历史版本的形式保存下来。了解 版本控制。

读写权限

私有

公共读

公共读写

公共读: 对文件写操作需要进行身份验证; 可以对文件进行匿名读。

服务器端加密

无

AES256

KMS

您尚未开通 KMS 服务

将文件上传至 OSS 后, 自动对其进行硬盘加密存储, KMS 加密方式需要进行权限设置, 当前 KMS 仅支持 OSS 默认托管的 CMK, 如需试用 KMS 单独创建的 CMK 加密 (BYOK), 请和我们联系, 了解 更多服务器端加密指南。

实时日志查询

开通

不开通

确定

取消

上传文件：

The screenshot shows the Alibaba Cloud Object Storage console. On the left, there's a sidebar with '对象存储' (Object Storage) at the top, followed by '概览' (Overview), 'Bucket 列表' (Bucket List) which is highlighted with a red box, '我的访问路径' (My Access Path), and 'gulimall-images'. The main area has a title 'gulimall-images' and a toolbar with '读写权限' (Read-Write Permissions), '上传文件' (Upload File) which is also highlighted with a red box, '新建目录' (Create Directory), '碎片管理' (Fragment Management), '授权' (Authorization), '批量操作' (Batch Operations), and '刷新' (Refresh). Below the toolbar, there are tabs for '文件管理' (File Management) and '权限管理' (Permission Management). A search bar labeled '文件名' (File Name) is present.

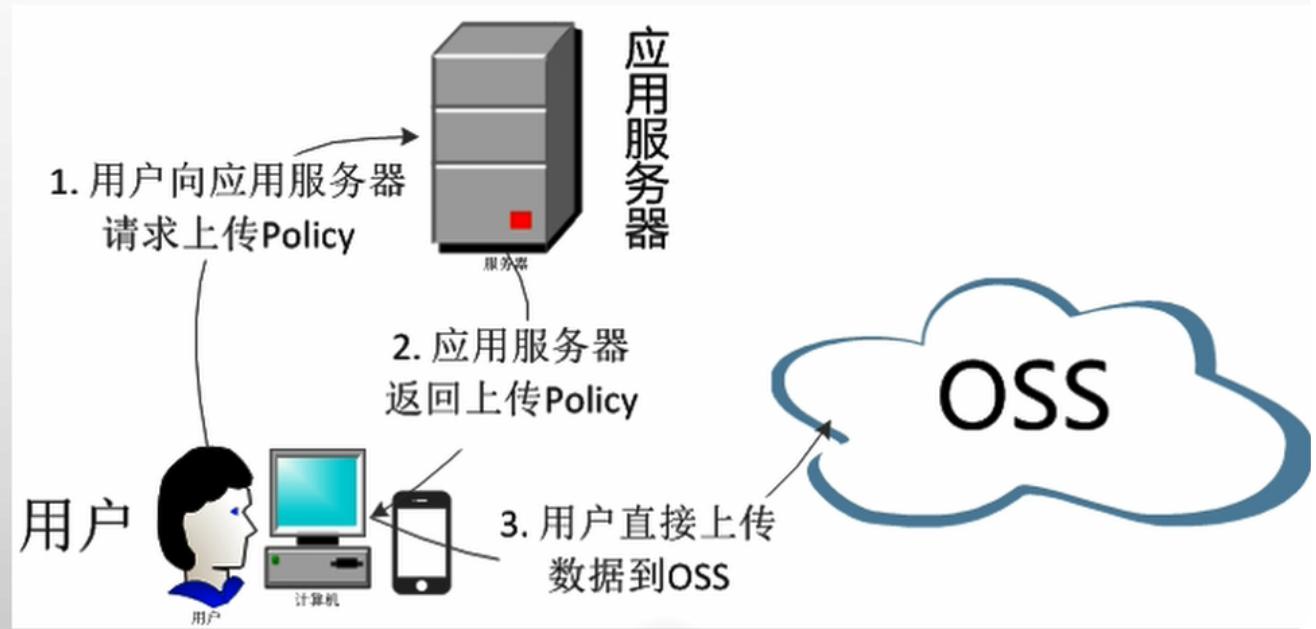
上传成功后，取得图片的URL

This screenshot shows the details for a single file named '5b5e74d0978360a1.jpg' in the 'gulimall-images' bucket. The file was uploaded on April 28, 2020, at 18:34. It has a size of 92.549KB and a low frequency access level. The 'More' link is highlighted with a red box. Below the file details, there is a expanded view showing the file name, ETag, and a toggle for using HTTPS. The URL is displayed as a copyable link: <https://gulimall-images.oss-cn-shanghai.aliyuncs.com/5b5e74d0978360a1.jpg>. There are also 'Download' and 'Open File URL' buttons.

这种方式是手动上传图片，实际上我们可以在程序中设置自动上传图片到阿里云对象存储。

上传模型：

阿里云对象存储-服务端签名后直传



查看阿里云关于文件上传的帮助: https://help.aliyun.com/document_detail/32009.html?spm=a2c4g.11186623.6.768.549d59aaWuZMGJ

1) 添加依赖包

在Maven项目中加入依赖项 (推荐方式)

在 Maven 工程中使用 OSS Java SDK, 只需在 pom.xml 中加入相应依赖即可。以 3.8.0 版本为例, 在 内加入如下内容:

```
<dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>3.8.0</version>
</dependency>
```

2) 上传文件流

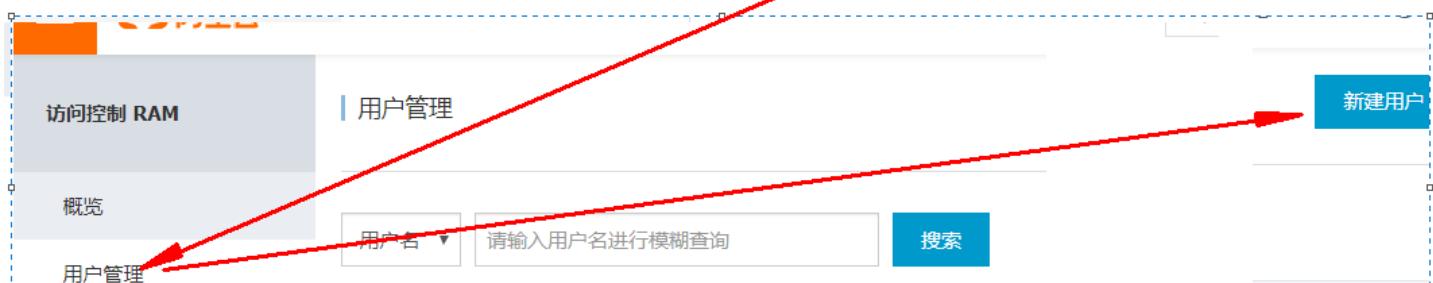
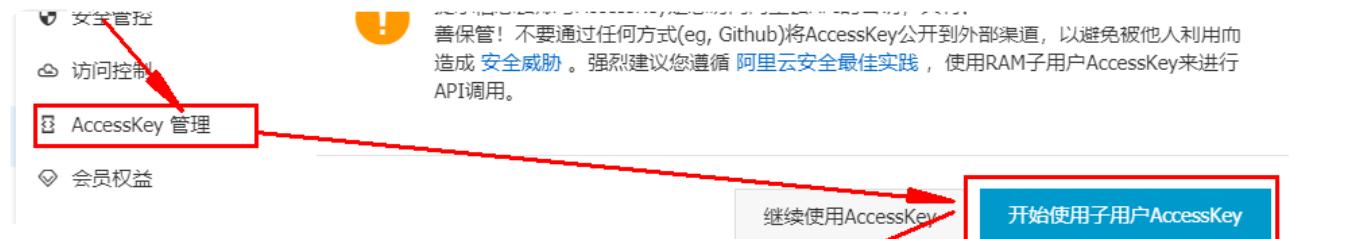
以下代码用于上传文件流：

```
// Endpoint以杭州为例，其它Region请按实际情况填写。  
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";  
// 云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用RAM子账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建。  
String accessKeyId = "<yourAccessKeyId>";  
String accessKeySecret = "<yourAccessKeySecret>";  
  
// 创建OSSClient实例。  
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);  
  
// 上传文件流。  
InputStream inputStream = new FileInputStream("<yourlocalFile>");  
ossClient.putObject("<yourBucketName>", "<yourObjectName>", inputStream);  
  
// 关闭OSSClient。  
ossClient.shutdown();
```

endpoint的取值：



accessKeyId和accessKeySecret需要创建一个RAM账号：



创建用户完毕后，会得到一个“AccessKey ID”和“AccessKeySecret”，然后复制这两个值到代码的“AccessKey ID”和“AccessKeySecret”。

另外还需要添加访问控制权限：

添加授权策略后，该账户即具有该条策略的权限，同一条授权策略不能被重复添加。

可选授权策略名称	类型
请输入关键词查询	<input type="text"/>
AdministratorAccess 管理所有阿里云资源的权限	系统
AliyunOSSReadOnlyAccess 只读访问对象存储服务(OSS)的权限	系统
AliyunECSFullAccess 管理云服务器服务(ECS)的权限	系统

已选授权策略名称	类型
AliyunOSSFullAccess 管理对象存储服务(OSS)权限	系统

```
@RunWith(MockitoJUnitRunner.class)
public class OSSTest {
    @Test
    public void testUpload() throws FileNotFoundException {
        // Endpoint以杭州为例，其它Region请按实际情况填写。
        String endpoint = "oss-cn-shanghai.aliyuncs.com";
        // 云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用RAM子账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建。
    }
}
```

```

String accessKeyId = "LTAI4G4W1RA4JXz2QhoDwHhi";
String accessKeySecret = "R991mDOJumF2x43ZBKT259Qpe70Oxw";

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

// 上传文件流。
InputStream inputStream = new
FileInputStream("C:\\\\Users\\\\Administrator\\\\Pictures\\\\timg.jpg");
ossClient.putObject("gulimall-images", "time.jpg", inputStream);

// 关闭OSSClient。
ossClient.shutdown();
System.out.println("上传成功。");
}

```

更为简单的使用方式，是使用SpringCloud Alibaba

接入 OSS

在启动示例进行演示之前，我们先了解一下如何接入 OSS。

注意：本节只是为了便于您理解接入方式，本示例代码中已经完成接入工作，您只需修改 accessKey、secretKey、endpoint 即可。

1. 修改 pom.xml 文件，引入 alicloud-oss starter。

```

<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alicloud-oss</artifactId>
</dependency>

```

2. 在配置文件中配置 OSS 服务对应的 accessKey、secretKey 和 endpoint。

```

// application.properties
spring.cloud.alicloud.access-key=your-ak
spring.cloud.alicloud.secret-key=your-sk
spring.cloud.alicloud.oss.endpoint=***

```

详细使用方法，见：https://help.aliyun.com/knowledge_detail/108650.html

- (1) 添加依赖

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alicloud-oss</artifactId>
    <version>2.2.0.RELEASE</version>
</dependency>
```

(2) 创建“AccessKey ID”和“AccessKeySecret”

(3) 配置key, secret和endpoint相关信息

```
access-key: LTAI4G4W1RA4JXz2QhoDwHhi
secret-key: R991mDOJumF2x43ZBKT259Qpe700xw
oss:
    endpoint: oss-cn-shanghai.aliyuncs.com
```

(4) 注入OSSClient并进行文件上传下载等操作

3. 注入 OSSClient 并进行文件上传下载等操作。

```
@Service
public class YourService {
    @Autowired
    private OSSClient ossClient;

    public void saveFile() {
        // download file to local
        ossClient.getObject(new GetObjectRequest(bucketName, objectName), new File("pathOfYourLocalFi:
    }
}
```

但是这样做还是比较麻烦，如果以后的上传任务都交给gulimall-product来完成，显然耦合度高。最好单独新建一个Module来完成文件上传任务。

其他方式

1) 新建gulimall-third-party

2) 添加依赖，将原来gulimall-common中的“spring-cloud-starter-alicloud-oss”依赖移动到该项目中

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alicloud-oss</artifactId>
    <version>2.2.0.RELEASE</version>
</dependency>

<dependency>
    <groupId>com.bigdata.gulimall</groupId>
    <artifactId>gulimall-common</artifactId>
    <version>1.0-SNAPSHOT</version>
    <exclusions>
        <exclusion>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

另外也需要在“pom.xml”文件中，添加如下的依赖管理

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<scope>import</scope>
</dependency>
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alibaba-dependencies</artifactId>
    <version>2.2.1.RELEASE</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

3) 在主启动类中开启服务的注册和发现



```
@EnableDiscoveryClient
```

4) 在nacos中注册

(1) 创建命名空间“gulimall-third-party”

新建命名空间



命名空间ID(不填则

自动生成):

* 命名空间名:

* 描述:

确定

取消

(2) 在“gulimall-third-party”命名空间中，创建“gulimall-third-party.yml”文件

```
spring:
  cloud:
    alicloud:
      access-key: LTAI4G4W1RA4JXz2QhoDwHhi
      secret-key: R991mDOJumF2x43ZBKT259Qpe700xw
    oss:
      endpoint: oss-cn-shanghai.aliyuncs.com
```

5) 编写配置文件

application.yml

```
server:
  port: 30000

spring:
  application:
    name: gulimall-third-party
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.137.14:8848

logging:
  level:
    com.bigdata.gulimall.product: debug
```

bootstrap.properties



```
spring.cloud.nacos.config.name=gulimall-third-party
spring.cloud.nacos.config.server-addr=192.168.137.14:8848
spring.cloud.nacos.config.namespace=f995d8ee-c53a-4d29-8316-a1ef54775e00
spring.cloud.nacos.config.extension-configs[0].data-id=gulimall-third-party.yml
spring.cloud.nacos.config.extension-configs[0].group=DEFAULT_GROUP
spring.cloud.nacos.config.extension-configs[0].refresh=true
```

6) 编写测试类



```
package com.bigdata.gulimall.thirdparty;

import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClient;
import com.aliyun.oss.OSSClientBuilder;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

@SpringBootTest
class GulimallThirdPartyApplicationTests {

    @Autowired
    OSSClient ossClient;

    @Test
    public void testUpload() throws FileNotFoundException {
        // Endpoint以杭州为例，其它Region请按实际情况填写。
        String endpoint = "oss-cn-shanghai.aliyuncs.com";
        // 云账号AccessKey有所有API访问权限，建议遵循阿里云安全最佳实践，创建并使用RAM子账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建。
        String accessKeyId = "LTAI4G4W1RA4JXz2QhoDwHhi";
        String accessKeySecret = "R991mDOJumF2x43ZBKT259Qpe70Oxw";
```

```

// 创建OSSClient实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

//上传文件流。
InputStream inputStream = new
FileInputStream("C:\\\\Users\\\\Administrator\\\\Pictures\\\\timg.jpg");
ossClient.putObject("gulimall-images", "time3.jpg", inputStream);

// 关闭OSSClient。
ossClient.shutdown();
System.out.println("上传成功。");
}

}

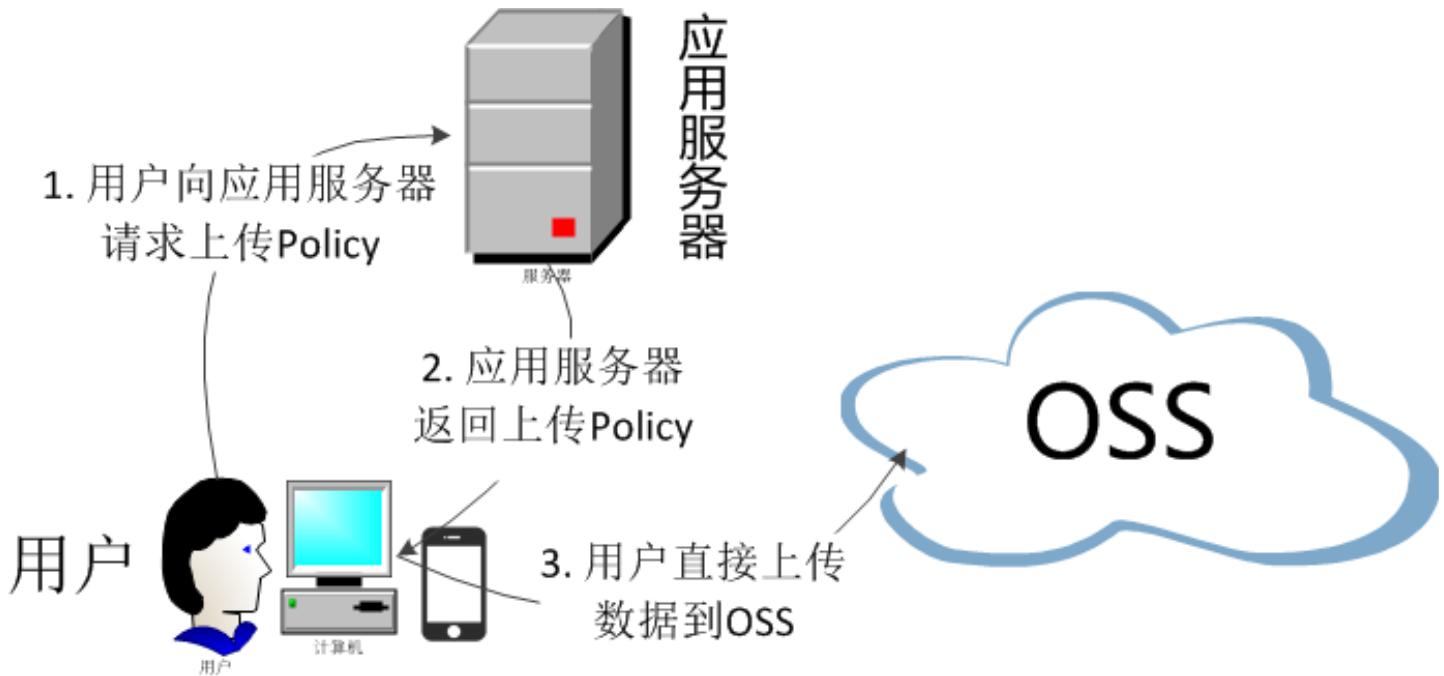
```

https://help.aliyun.com/document_detail/31926.html?spm=a2c4g.11186623.6.1527.228d74b8V6lZuT

背景

采用JavaScript客户端直接签名（参见[JavaScript客户端签名直传](#)）时，AccessKeyId和AccessKeySecret会暴露在前端页面，因此存在严重的安全隐患。因此，OSS提供了服务端签名后直传的方案。

原理介绍



服务端签名后直传的原理如下：

1. 用户发送上传Policy请求到应用服务器。
2. 应用服务器返回上传Policy和签名给用户。
3. 用户直接上传数据到OSS。

编写“com.bigdata.gulimall.thirdparty.controller.OssController”类：



```
package com.bigdata.gulimall.thirdparty.controller;

import com.aliyun.oss.OSS;
import com.aliyun.oss.common.utils.BinaryUtil;
import com.aliyun.oss.model.MatchMode;
import com.aliyun.oss.model.PolicyConditions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.LinkedHashMap;
import java.util.Map;

@RestController
public class OssController {

    @Autowired
    OSS ossClient;
    @Value ("${spring.cloud.alicloud.oss.endpoint}")
    String endpoint ;

    @Value("${spring.cloud.alicloud.oss.bucket}")
    String bucket ;

    @Value("${spring.cloud.alicloud.access-key}")
    String accessId ;
    @Value("${spring.cloud.alicloud.secret-key}")
    String accessKey ;
    @RequestMapping("/oss/policy")
    public Map<String, String> policy(){
```

```

String host = "https://" + bucket + "." + endpoint; // host的格式为
bucketname.endpoint

String format = new SimpleDateFormat("yyyy-MM-dd").format(new Date());
String dir = format; // 用户上传文件时指定的前缀。

Map<String, String> respMap=null;
try {
    long expireTime = 30;
    long expireEndTime = System.currentTimeMillis() + expireTime * 1000;
    Date expiration = new Date(expireEndTime);
    PolicyConditions policyConds = new PolicyConditions();
    policyConds.addConditionItem(PolicyConditions.COND_CONTENT_LENGTH_RANGE, 0,
1048576000);
    policyConds.addConditionItem(MatchMode.StartWith, PolicyConditions.COND_KEY,
dir);

    String postPolicy = ossClient.generatePostPolicy(expiration, policyConds);
    byte[] binaryData = postPolicy.getBytes("utf-8");
    String encodedPolicy = BinaryUtil.toBase64String(binaryData);
    String postSignature = ossClient.calculatePostSignature(postPolicy);

    respMap= new LinkedHashMap<String, String>();
    respMap.put("accessid", accessId);
    respMap.put("policy", encodedPolicy);
    respMap.put("signature", postSignature);
    respMap.put("dir", dir);
    respMap.put("host", host);
    respMap.put("expire", String.valueOf(expireEndTime / 1000));

} catch (Exception e) {
    // Assert.fail(e.getMessage());
    System.out.println(e.getMessage());
} finally {
    ossClient.shutdown();
}
return respMap;
}
}

```

测试: <http://localhost:30000/oss/policy>



```
{ "accessid": "LTAI4G4W1RA4JXz2QhoDwHhi", "policy": "eyJleHBpcmF0aW9uIjoiMjAyMC0wNC0yOVQwMjolODowNy41NzhaiIwiY29uZGl0aW9ucyI6W1siY29udGVudC1sZW5ndGgtcmFuZ2UiLDAsMTA0ODU3NjAwMF0sWyJzdGFydHMtd2l0aCIslIiRrZXkiLCIyMDIwLTA0LTi5LyJdXX0=", "signature": "s42iRxtxGFmHyG40StM3d9v0ffk=", "dir": "2020-04-29/", "host": "https://gulimall-images.oss-cn-shanghai.aliyuncs.com", "expire": "1588129087"}
```

以后在上传文件时的访问路径为“<http://localhost:88/api/thirdparty/oss/policy>”，

在“gulimall-gateway”中配置路由规则：



```
- id: third_party_route
  uri: lb://gulimall-gateway
  predicates:
    - Path=/api/thirdparty/**
  filters:
    - RewritePath=/api/thirdparty/( ?<segment>/?.*) , /$\\{segment}
```

测试是否能够正常跳转：<http://localhost:88/api/thirdparty/oss/policy>

The screenshot shows a browser window with the URL localhost:88/api/thirdparty/oss/policy in the address bar. The page content is a JSON object:

```
accessid: "LTAI4G4W1RA4JXz2QhoDwHhi"
policy: "eyJleHBpcmF0aW9uIjoiMjAyMC0wNC0yOVQwMzoxNDoxNi4yNjlaIiwiY29uZGl0aW9ucyI6W1siY29udGVudC1sZW5ndGgtcmFuZ2UiLDAsMTA0ODU3NjAwMF0sWyJzdGFydHMtd2l0aCIslIiRrZXkiLCIyMDIwLTA0LTi5LyJdXX0="
signature: "i1sYrtA2YKGNPBK2346nJ20N1b0="
dir: "2020-04-29/"
host: "https://gulimall-images.oss-cn-shanghai.aliyuncs.com"
expire: "1588130056"
```

上传组件

放置项目提供的upload文件夹到components目录下，一个是单文件上传，另外一个是多文件上传



```
PS D:\Project\gulimall\renren-fast-vue\src\components\upload> ls
```

目录: D:\Project\gulimall\renren-fast-vue\src\components\upload

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
-a---	2020/4/29 星期三 2020-04-29 12:02	3122	multiUpload.vue
-a---	2019/11/11 星期一 2019-11-11 21:20	343	policy.js
-a---	2020/4/29 星期三 2020-04-29 12:01	3053	singleUpload.vue

```
PS D:\Project\gulimall\renren-fast-vue\src\components\upload>
```

修改这两个文件的配置后

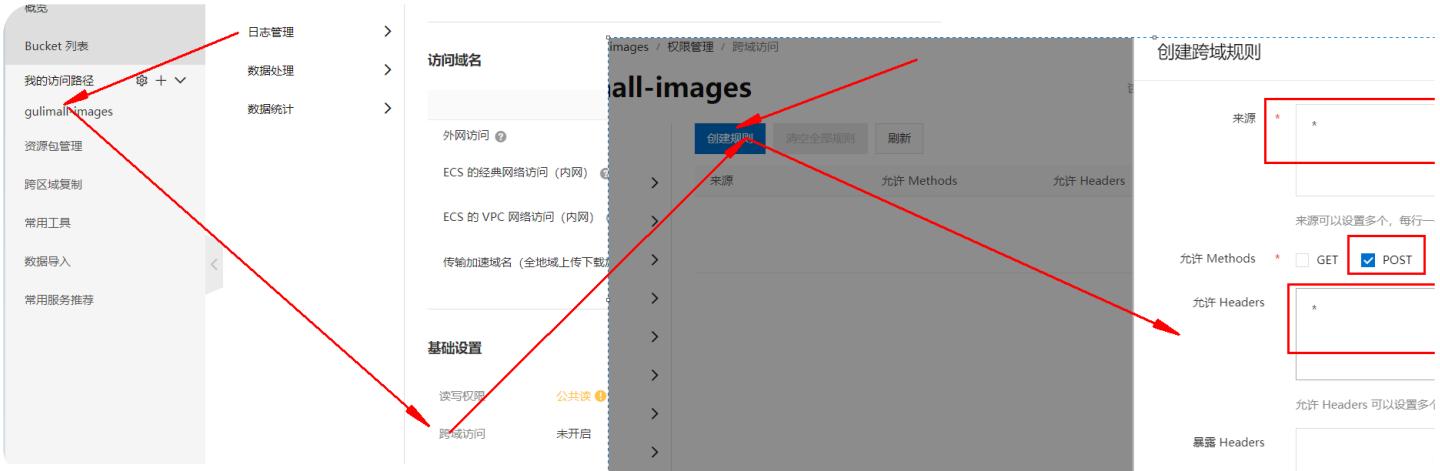
开始执行上传，但是在上传过程中，出现了如下的问题：

```
① Access to XMLHttpRequest at 'http://gulimall-images.oss-cn-shanghai.aliyuncs.com/' from origin 'http://localhost:8001' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. :8001#/product-brand:1
```



```
Access to XMLHttpRequest at 'http://gulimall-images.oss-cn-shanghai.aliyuncs.com/' from origin 'http://localhost:8001' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

这又是一个跨域的问题，解决方法就是在阿里云上开启跨域访问：



再次执行文件上传。

18. JSR303校验

步骤1：使用校验注解

在Java中提供了一系列的校验方式，它这些校验方式在“javax.validation.constraints”包中，提供了如@Email, @NotNull等注解。

在非空处理方式上提供了@NotNull, @Blank和@

(1) @NotNull

The annotated element must not be null. Accepts any type.

注解元素禁止为null，能够接收任何类型

(2) @NotEmpty

the annotated element must not be null nor empty.

该注解修饰的字段不能为null或""

Supported types are:

支持以下几种类型

CharSequence (length of character sequence is evaluated)

字符串序列（字符串序列长度的计算）

Collection (collection size is evaluated)

集合长度的计算

Map (map size is evaluated)

map长度的计算

Array (array length is evaluated)

数组长度的计算

(3) @NotBlank

The annotated element must not be null and must contain at least one non-whitespace character. Accepts CharSequence.

该注解不能为null，并且至少包含一个非空白字符。接收字符序列。

步骤2：在请求方法种，使用校验注解@Valid，开启校验，

```
@RequestMapping("/save")
public R save(@Valid @RequestBody BrandEntity brand){
    brandService.save(brand);

    return R.ok();
}
```

测试：<http://localhost:88/api/product/brand/save>

在postman种发送上面的请求

```
{
  "timestamp": "2020-04-29T09:20:46.383+0000",
  "status": 400,
  "error": "Bad Request",
  "errors": [
    {
      "codes": [
        "NotBlank.brandEntity.name",
        "NotBlank.name",
        "NotNull.name"
      ]
    }
  ]
}
```

```

        "NotBlank.java.lang.String",
        "NotBlank"
    ],
    "arguments": [
        {
            "codes": [
                "brandEntity.name",
                "name"
            ],
            "arguments": null,
            "defaultMessage": "name",
            "code": "name"
        }
    ],
    "defaultMessage": "不能为空",
    "objectName": "brandEntity",
    "field": "name",
    "rejectedValue": "",
    "bindingFailure": false,
    "code": "NotBlank"
}
],
"message": "Validation failed for object='brandEntity'. Error count: 1",
"path": "/product/brand/save"
}

```

能够看到"defaultMessage": "不能为空", 这些错误消息定义在"hibernate-validator"的"\org\hibernate\validator\ValidationMessages_zh_CN.properties"文件中。在该文件中定义了很多的错误规则:



javax.validation.constraints.AssertFalse.message	= 只能为false
javax.validation.constraints.AssertTrue.message	= 只能为true
javax.validation.constraints.DecimalMax.message	= 必须小于或等于{value}
javax.validation.constraints.DecimalMin.message	= 必须大于或等于{value}
javax.validation.constraints.Digits.message {integer}位整数和{fraction}位小数范围内)	= 数字的值超出了允许范围(只允许在
javax.validation.constraints.Email.message	= 不是一个合法的电子邮件地址
javax.validation.constraints.Future.message	= 需要是一个将来的时间
javax.validation.constraints.FutureOrPresent.message	= 需要是一个将来或现在的时间
javax.validation.constraints.Max.message	= 最大不能超过{value}
javax.validation.constraints.Min.message	= 最小不能小于{value}
javax.validation.constraints.Negative.message	= 必须是负数
javax.validation.constraints.NegativeOrZero.message	= 必须是负数或零
javax.validation.constraints.NotBlank.message	= 不能为空

javax.validation.constraints.NotEmpty.message	= 不能为空
javax.validation.constraints.NotNull.message	= 不能为null
javax.validation.constraints.Null.message	= 必须为null
javax.validation.constraints.Past.message	= 需要是一个过去的时间
javax.validation.constraints.PastOrPresent.message	= 需要是一个过去或现在的时间
javax.validation.constraints.Pattern.message	= 需要匹配正则表达式"{{regexp}}
javax.validation.constraints.Positive.message	= 必须是正数
javax.validation.constraints.PositiveOrZero.message	= 必须是正数或零
javax.validation.constraints.Size.message	= 个数必须在{{min}}和{{max}}之间
org.hibernate.validator.constraints.CreditCardNumber.message	= 不合法的信用卡号码
org.hibernate.validator.constraints.Currency.message {value}其中之一)	= 不合法的货币 (必须是
org.hibernate.validator.constraints.EAN.message	= 不合法的{{type}}条形码
org.hibernate.validator.constraints.Email.message 地址	= 不是一个合法的电子邮件
org.hibernate.validator.constraints.Length.message {max}之间	= 长度需要在{{min}}和
org.hibernate.validator.constraints.CodePointLength.message {max}之间	= 长度需要在{{min}}和
org.hibernate.validator.constraints.LuhnCheck.message 校验码不合法, Luhn模10校验和不匹配	= \${validatedValue}的
org.hibernate.validator.constraints.Mod10Check.message 校验码不合法, 模10校验和不匹配	= \${validatedValue}的
org.hibernate.validator.constraints.Mod11Check.message 校验码不合法, 模11校验和不匹配	= \${validatedValue}的
org.hibernate.validator.constraints.ModCheck.message 校验码不合法, \${modType}校验和不匹配	= \${validatedValue}的
org.hibernate.validator.constraints.NotBlank.message	= 不能为空
org.hibernate.validator.constraints.NotEmpty.message	= 不能为空
org.hibernate.validator.constraints.ParametersScriptAssert.message {script}"没有返回期望结果	= 执行脚本表达式"
org.hibernate.validator.constraints.Range.message 间	= 需要在{{min}}和{{max}}之间
org.hibernate.validator.constraints.SafeHtml.message	= 可能有不安全的HTML内容
org.hibernate.validator.constraints.ScriptAssert.message {script}"没有返回期望结果	= 执行脚本表达式"
org.hibernate.validator.constraints.URL.message	= 需要是一个合法的URL
org.hibernate.validator.constraints.time.DurationMax.message == true ? '或等于' : ''}\${days == 0 ? '' : days += '天'}\${hours == 0 ? '' : hours += '小时'}\${minutes == 0 ? '' : minutes += '分钟'}\${seconds == 0 ? '' : seconds += '秒'}\${millis == 0 ? '' : millis += '毫秒'}\${nanos == 0 ? '' : nanos += '纳秒'}	= 必须小于\${inclusive}

```
org.hibernate.validator.constraints.time.DurationMin.message      = 必须大于${inclusive}
== true ? '或等于' : ''}${days == 0 ? '' : days += '天'}${hours == 0 ? '' : hours += '小
时'}${minutes == 0 ? '' : minutes += '分钟'}${seconds == 0 ? '' : seconds += '秒'}${millis
== 0 ? '' : millis += '毫秒'}${nanos == 0 ? '' : nanos += '纳秒'}
```

想要自定义错误消息，可以覆盖默认的错误提示信息，如@NotBlank的默认message是

```
public @interface NotBlank {  
  
    String message() default "{javax.validation.constraints.NotBlank.message}";
```

可以在添加注解的时候，修改message：

```
@NotBlank(message = "品牌名必须非空")
private String name;
```

当再次发送请求时，得到的错误提示信息：

```
{
    "timestamp": "2020-04-29T09:36:04.125+0000",
    "status": 400,
    "error": "Bad Request",
    "errors": [
        {
            "codes": [
                "NotBlank.brandEntity.name",
                "NotBlank.name",
                "NotBlank.java.lang.String",
                "NotBlank"
            ],
            "arguments": [
                {
                    "name": "name"
                }
            ]
        }
    ]
}
```

```

        "codes": [
            "brandEntity.name",
            "name"
        ],
        "arguments": null,
        "defaultMessage": "name",
        "code": "name"
    }
],
"defaultMessage": "品牌名必须非空",
"objectName": "brandEntity",
"field": "name",
"rejectedValue": "",
"bindingFailure": false,
"code": "NotBlank"
}
],
"message": "Validation failed for object='brandEntity'. Error count: 1",
"path": "/product/brand/save"
}

```

但是这种返回的错误结果并不符合我们的业务需要。

步骤3：给校验的Bean后，紧跟一个BindResult，就可以获取到校验的结果。拿到校验的结果，就可以自定义的封装。



```

@RequestMapping("/save")
public R save(@Valid @RequestBody BrandEntity brand, BindingResult result){
    if( result.hasErrors()){
        Map<String, String> map=new HashMap<>();
        //1. 获取错误的校验结果
        result.getFieldErrors().forEach((item)->{
            //获取发生错误时的message
            String message = item.getDefaultMessage();
            //获取发生错误的字段
            String field = item.getField();
            map.put(field,message);
        });
        return R.error(400,"提交的数据不合法").put("data",map);
    }
}

```

```
        }else {
            }
            brandService.save(brand);

            return R.ok();
        }
    }
```

这种是针对于该请求设置了一个内容校验，如果针对于每个请求都单独进行配置，显然不是太合适，实际上可以统一的对于异常进行处理。

步骤4：统一异常处理

可以使用SpringMvc所提供的@ControllerAdvice，通过“basePackages”能够说明处理哪些路径下的异常。

(1) 抽取一个异常处理类



```
package com.bigdata.gulimall.product.exception;

import com.bigdata.common.utils.R;
import lombok.extern.slf4j.Slf4j;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.HashMap;
import java.util.Map;

/**
 * 集中处理所有异常
 */
@Slf4j
@RestControllerAdvice(basePackages = "com.bigdata.gulimall.product.controller")
public class GulimallExceptionAdvice {

    @ExceptionHandler(value = Exception.class)
    public R handleValidException(MethodArgumentNotValidException exception){}
```

```

        Map<String, String> map = new HashMap<>();
        BindingResult bindingResult = exception.getBindingResult();
        bindingResult.getFieldErrors().forEach(fieldError -> {
            String message = fieldError.getDefaultMessage();
            String field = fieldError.getField();
            map.put(field, message);
        });

        log.error("数据校验出现问题{}, 异常类型
        {}, exception.getMessage(), exception.getClass());
        return R.error(400, "数据校验出现问题").put("data", map);
    }

}

```

(2) 测试: <http://localhost:88/api/product/brand/save>

POST <http://localhost:88/api/product/brand/save>

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

1 [{"name": ""}]

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON

```

1 {
2     "msg": "数据校验出现问题",
3     "code": 400,
4     "data": {
5         "name": "不能为空",
6         "logo": "不能为空",
7         "sort": "不能为空",
8         "firstLetter": "不能为空"
9     }
10 }

```

(3) 默认异常处理



```
@ExceptionHandler(value = Throwable.class)
public R handleException(Throwable throwable){
    log.error("未知异常{},异常类型{}", throwable.getMessage(), throwable.getClass());
    return
}
R.error(BizCodeEnum.UNKNOW_EXCEPTION.getCode(), BizCodeEnum.UNKNOW_EXCEPTION.getMsg());
```

(4) 错误状态码

上面代码中，针对于错误状态码，是我们进行随意定义的，然而正规开发过程中，错误状态码有着严格的定义规则，如该在项目中我们的错误状态码定义

`@ControllerAdvice+@ExceptionHandler`

`系统错误码`

```
/*
 * 错误码和错误信息定义类
 * 1. 错误码定义规则为 5 位数字
 * 2. 前两位表示业务场景，最后三位表示错误码。例如：100001。10:通用 001:系统未知
 * 异常
 * 3. 维护错误码后需要维护错误描述，将他们定义为枚举形式
 * 错误码列表：
 * 10: 通用
 * 001: 参数格式校验
 * 11: 商品
 * 12: 订单
 * 13: 购物车
 * 14: 物流
 *
 */

```

为了定义这些错误状态码，我们可以单独定义一个常量类，用来存储这些错误状态码



```
package com.bigdata.common.exception;
```

```
/*
 * 错误码和错误信息定义类
 * 1. 错误码定义规则为5位数字
 * 2. 前两位表示业务场景，最后三位表示错误码。例如：100001。10：通用 001：系统未知异常
 * 3. 维护错误码后需要维护错误描述，将他们定义为枚举形式
 * 错误码列表：
 * 10：通用
 *      001：参数格式校验
 * 11：商品
 * 12：订单
 * 13：购物车
 * 14：物流
 */
public enum BizCodeEnum {

    UNKNOW_EXCEPTION(10000, "系统未知异常"),

    VALID_EXCEPTION( 10001, "参数格式校验失败");

    private int code;
    private String msg;

    BizCodeEnum(int code, String msg) {
        this.code = code;
        this.msg = msg;
    }

    public int getCode() {
        return code;
    }

    public String getMsg() {
        return msg;
    }
}
```

(5) 测试：<http://localhost:88/api/product/brand/save>

The screenshot shows a Postman request to `http://localhost:88/api/product/brand/save`. The request method is `POST`. In the `Body` tab, the `raw` JSON payload is:

```
{ "name": "" }
```

In the `Test Results` tab, the response body is:

```
{ "msg": "参数格式校验失败", "code": 10001, "data": [ { "name": "不能为空", "logo": "不能为空", "sort": "不能为空为null", "firstLetter": "不能为空" } ] }
```

19. 分组校验功能（完成多场景的复杂校验）

1、给校验注解，标注上groups，指定什么情况下才需要进行校验

如：指定在更新和添加的时候，都需要进行校验

```
@NotEmpty  
@NotBlank(message = "品牌名必须非空", groups = {UpdateGroup.class, AddGroup.class})  
private String name;
```

在这种情况下，没有指定分组的校验注解，默认是不起作用的。想要起作用就必须加groups。

2、业务方法参数上使用@Validated注解

@Validated的value方法：

Specify one or more validation groups to apply to the validation step kicked off by this annotation.
指定一个或多个验证组以应用于此注释启动的验证步骤。

JSR-303 defines validation groups as custom annotations which an application declares for the sole purpose of using them as type-safe group arguments, as implemented in SpringValidatorAdapter.

JSR-303 将验证组定义为自定义注释，应用程序声明的唯一目的是将它们用作类型安全组参数，如 SpringValidatorAdapter 中实现的那样。

Other SmartValidator implementations may support class arguments in other ways as well.

其他SmartValidator 实现也可以以其他方式支持类参数。

3、默认情况下，在分组校验情况下，没有指定指定分组的校验注解，将不会生效，它只会在不分组的情况下生效。

20. 自定义校验功能

1、编写一个自定义的校验注解



```
@Documented
@Constraint(validatedBy = { ListValueConstraintValidator.class })
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER, TYPE_USE })
@Retention(RUNTIME)
public @interface ListValue {
    String message() default "{com.bigdata.common.valid.ListValue.message}";

    Class<?>[] groups() default { };
}
```

```
Class<? extends Payload>[] payload() default { };  
  
int[] value() default {};  
}
```

2、编写一个自定义的校验器



```
public class ListValueConstraintValidator implements  
ConstraintValidator<ListValue, Integer> {  
    private Set<Integer> set=new HashSet<>();  
    @Override  
    public void initialize(ListValue constraintAnnotation) {  
        int[] value = constraintAnnotation.value();  
        for (int i : value) {  
            set.add(i);  
        }  
    }  
  
    @Override  
    public boolean isValid(Integer value, ConstraintValidatorContext context) {  
  
        return set.contains(value);  
    }  
}
```

3、关联自定义的校验器和自定义的校验注解



```
@Constraint(validatedBy = { ListValueConstraintValidator.class })
```

4、使用实例



```
/**  
 * 显示状态[0-不显示; 1-显示]  
 */  
@ListValue(value = {0,1}, groups ={AddGroup.class})  
private Integer showStatus;
```

21. 商品SPU和SKU管理

重新执行“sys_menus.sql”

22. 点击子组件，父组件触发事件

现在想要实现点击菜单的左边，能够实现在右边展示数据

The screenshot shows a Vue.js application interface. On the left, there is a sidebar with a tree view of categories: '手机' (selected), '图书、音像、电子书刊' (selected), and '电子书'. The '手机' category has children: '手机回收', '手机通讯', '运营商', and '手机配件'. The '图书、音像、电子书刊' category has a child '电子书'. On the right, there is a search bar with fields for '参数名' and '查询', a green '新增' button, and a red '批量删除' button. Below the search bar is a table with columns: '分组id', '组名', '排序', '描述', '组图标', '所属分类id', and '操作'. A red arrow points from the '手机' item in the sidebar to the main content area, indicating a data binding or event delegation mechanism.

父子组件传递数据：

1) 子组件给父组件传递数据，事件机制；

在category中绑定node-click事件，



```
<el-tree :data="menus" :props="defaultProps" node-key="catId" ref="menuTree" @node-click="nodeClick" ></el-tree>
```

2) 子组件给父组件发送一个事件，携带上数据；



```
nodeClick(data,Node,component){  
    console.log("子组件",data,Node,component);  
    this.$emit("tree-node-click",data,Node,component);  
},
```

this.\$emit(事件名,"携带的数据");

3) 父组件中的获取发送的事件



```
<category @tree-node-click="treeNodeClick"></category>
```



```
//获取发送的事件数据  
treeNodeClick(data,Node,component){  
    console.log("attgroup感知到的category的节点被点击",data,Node,component);  
    console.log("刚才被点击的菜单ID",data.catId);  
},
```

23. 规格参数新增与VO

规格参数新增时，请求的URL：Request URL:

<http://localhost:88/api/product/attr/base/list/0?t=1588731762158&page=1&limit=10&key=>

当有新增字段时，我们往往会在entity实体类中新建一个字段，并标注数据库中不存在该字段，然而这种方式并不规范



```
54
55     private Long enable;
56
57     /**
58      * 所属分类
59      */
60     private Long catalogId;
61
62     /**
63      * 快速展示【是否展示在介绍上；0-否 1-是】，在
64      */
65     @TableField(exist = false)
66     private Long attrGroupId;
67
```

比较规范的做法是，新建一个vo文件夹，将每种不同的对象，按照它的功能进行了划分。在java中，涉及到了这几种类型



2、SPU 与 SKU
3、基本属性【规格参数】与销售属性

二、接口编写

- 1、HTTP 请求模板
- 2、JSR303 数据校验
 - 1)、使用步骤
 - 3、全局异常处理
 - 4、接口文档地址
- 5、Object 划分
 - 1.PO(persistent object) 持久对 ...
 - 2.DO (Domain Object) 领域对 ...
 - 3.TO(Transfer Object) , 数据传 ...
 - 4.DTO (Data Transfer Object) ...
 - 5.VO(value object) 值对象
 - 6.BOO(business object) 业务对象
 - 7.POJO(plain ordinary java obje ...
 - 8.DAO(data access object) 数据 ...

1.PO(persistent object)

PO 就是对应数据库中某个表中的一条记录，多个记录可以用 PO 的集...
含任何对数据库的操作。

2.DO (Domain Object) 领域对象

就是从现实世界中抽象出来的有形或无形的业务实体。

3.TO(Transfer Object) , 数据传输对象

不同的应用程序之间传输的对象

4.DTO (Data Transfer Object) 数据传输对象

Request URL: <http://localhost:88/api/product/attr/save>，现在的情况是，它在保存的时候，只是保存了attr，并没有保存attrgroup，为了解决这个问题，我们新建了一个vo/AttrVo，在原AttrEntity基础上增加了attrGroupId字段，使得保存新增数据的时候，也保存了它们之间的关系。

通过" BeanUtils.copyProperties(attr,attrEntity);"能够实现在两个Bean之间拷贝数据，但是两个Bean的字段要相同

```
@Override  
public void saveAttr(AttrVo attr) {  
    AttrEntity attrEntity = new AttrEntity();  
    BeanUtils.copyProperties(attr,attrEntity);  
    this.save(attrEntity);  
}
```

问题：现在有两个查询，一个是查询部分，另外一个是查询全部，但是又必须这样做吗？还是有必要的，但是可以在后台进行设计，两种查询是根据catId是否为零进行区分的。

24. 查询分组关联属性和删除关联

获取属性分组的关联的所有属性

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/LnjzZHPj>

发送请求: /product/attrgroup/{attrgroupId}/attr/relation

获取当前属性分组所关联的属性

标	属性名	可选值	操作	所属分类id	操作
	3	品牌	1	brand	226
	4	网络类型	1	type	236
		电信、联通、移动、全网通			235
					228

如何查找：既然给出了attr_group_id，那么到中间表中查询出来所关联的attr_id，然后得到最终的所有属性即可。

可能出现null值的问题

25. 查询分组未关联的属性

/product/attrgroup/{attrgroupId}/noattr/relation

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/d3EezLdO>

获取属性分组里面还没有关联的本分类里面的其他基本属性，方便添加新的关联

Request URL: <http://localhost:88/api/product/attrgroup/1/noattr/relation?t=1588780783441&page=1&limit=10&key=>

The screenshot shows a modal dialog titled '选择属性' (Select Attribute) in the foreground. It contains a search bar, a table header with columns '属性id' (Attribute ID), '属性名' (Attribute Name), '属性图标' (Attribute Icon), and '可选' (Selectable). Below the table, it says '暂无数据' (No data available). At the bottom are '取消' (Cancel) and '确认新增' (Confirm New Addition) buttons. A red arrow points from the '新添加' (Newly Added) button in the sidebar to the '选择属性' dialog. Another red arrow points from the '关联' (Associate) button in the table's '操作' (Operations) column to the '确认新增' button.

属性id	属性名	属性图标	可选	操作	
226	brand	235	type	228	关联
236	236	231	231	231	关联

属性分组，对应于“pms_attr_group”表，每个分组下，需要查看到关联了哪些属性信息，销售属性不需要和分组进行关联，但是规格参数要和属性分组进行关联。

规格参数：对应于 pms_attr 表，attr_type=1，需要显示分组信息

销售属性：对应于pms_attr`表，attr_type=0，不需要显示分组信息

分组ID为9的分组：Request URL: <http://localhost:88/api/product/attrgroup/9/noattr/relation?t=1588822258669&page=1&limit=10&key=>

对应的数据库字段

attr_group_id attr_group_name sort descript icon catalog_id

9	主体	1	型号 平台	wu	454
10	显卡	1	显存容量	wu	454
11	输入设备	1	鼠标 键盘	wu	454
12	主板	1	显卡类型 芯片组	wu	454
13	规格	1	尺寸	wu	454

查询attrgroupId=9的属性分组：

```
AttrGroupEntity attrGroupEntity = attrGroupDao.selectById(attrgroupId);
```

获取到分类信息：

```
Long catalogId = attrGroupEntity.getCatalogId();
```

目标：获取属性分组没有关联的其他属性

也就是获取attrgroupId=9的属性分组中，关联的分类catalog_id =454（台式机），其他基本属性

在该属性分组中，现在已经关联的属性：

新建关联		批量删除		
#	属性名	可选值	操作	
<input type="checkbox"/>	4	机身颜色	天空之境 ...	移除
<input type="checkbox"/>	1	CPU品牌	aa ...	移除

本分类下，存在哪些基本属性？

没有关联的其他属性

已经关联的属性，这些属性是如何关联上的？

答：在创建规格参数的时候，已经设置了需要关联哪些属性分组。

想要知道还没有关联哪些，先查看关联了哪些，如何排除掉这些就是未关联的

在中间表中显示了属性和属性分组之间的关联关系，在属性表中显示了所有的属性，

先查询中间表，得到所有已经关联的属性的id，然后再次查询属性表，排除掉已经建立关联的属性ID，将剩下的属性ID和属性建立起关联关系

26. 添加属性和分组的关联关系

请求类型：Request URL: <http://localhost:88/api/product/attrgroup/attr/relation>

请求方式：POST

请求数据：[{"attrId":10,"attrGroupId":9}]

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/VhgnaedC>

响应数据：

```
{
  "msg": "success",
  "code": 0
}
```

本质就是在中间表pms_attr_attrgroup_relation中，添加一条记录的过程

27. 发布商品

获取所有会员等级：/member/memberlevel/list

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/jCEganpf>

在“gulimall-gateway”中修改“”文件，添加对于member的路由

```
- id: gulimall-member
  uri: lb://gulimall-member
  predicates:
    - Path=/api/member/**
  filters:
    - RewritePath=/api/(?<segment>/?.*)/$\{segment}
```

在“gulimall-member”中，创建“bootstrap.properties”文件，内容如下：

```
spring.cloud.nacos.config.name=gulimall-member
spring.cloud.nacos.config.server-addr=192.168.137.14:8848
spring.cloud.nacos.config.namespace=795521fa-77ef-411e-a8d8-0889fdfe6964
spring.cloud.nacos.config.extension-configs[0].data-id=gulimall-member.yml
spring.cloud.nacos.config.extension-configs[0].group=DEFAULT_GROUP
spring.cloud.nacos.config.extension-configs[0].refresh=true
```

获取分类关联的品牌：/product/categorybrandrelation/brands/list

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/HgVjlzWV>

遇到PubSub问题

1. 首先安装pubsub-js

```
`npm install --save pubsub-js`
```

2. 订阅方组件



```
`import PubSub from 'pubsub-js'`
```

该this.PubSub为PubSub。

获取分类下所有分组&关联属性

请求类型: /product/attrgroup/{catalogId}/withattr

请求方式: GET

请求URL: <http://localhost:88/api/product/attrgroup/225/withattr?t=1588864569478>

mysql默认的隔离级别为读已提交，为了能够在调试过程中，获取到数据库中的数据信息，可以调整隔离级别为读未提交：



```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

但是它对于当前的事务窗口生效，如果想要设置全局的，需要加上global字段。

28. 商品管理

当新建时：



```
t: 1588983621569
status: 0
key:
brandId: 0
catalogId: 0
page: 1
limit: 10
```

当上架时：

```
t: 1588983754030  
status: 1  
key:  
brandId: 0  
catalogId: 0  
page: 1  
limit: 10
```

当下架时：

```
t: 1588983789089  
status: 2  
key:  
brandId: 0  
catalogId: 0  
page: 1  
limit: 10
```

在SPU中，写出的日期数据都不符合规则：

	id	名称	描述	分类	品牌	重量	上架状态	创建时间	修改时间	操作
	9	华为Mate30	华为Mate30	225	1	0.2	<button>新建</button>	2020-05-08 T16:02:20.0 00+0000	2020-05-08 T16:02:20.0 00+0000	上架 规格

想要符合规则，可以设置写出数据的规则：

spring.jackson

```
jackson:  
date-format: yyyy-MM-dd HH:mm:ss
```

SKU检索：

Request URL: <http://localhost:88/api/product/skuinfo/list?t=1588989437944&page=1&limit=10&key=&catalogId=0&brandId=0&min=0&max=0>

请求体:

```
t: 1588989437944
page: 1
limit: 10
key:
catalogId: 0
brandId: 0
min: 0
max: 0
```

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/ucirLq1D>

29. 仓库管理

库存信息表: wms_ware_info

【1】仓库列表功能:

【2】查询商品库存:

【3】查询采购需求:

【4】合并采购需求:

合并整单选中parchaseID: Request URL: <http://localhost:88/api/ware/purchase/merge>

请求数据:

```
{purchaseId: 1, items: [1, 2]}
items: [1, 2]
```

合并整单未选择parchaseID :Request URL: <http://localhost:88/api/ware/purchase/merge>

提示

×



没有选择任何【采购单】，将自动创建新单进行合并。

确认吗？

取消

确定



items: [1, 2]

涉及到两张表: wms_purchase_detail, wms_purchase

现在采购单中填写数据，然后关联用户，关联用户后，

总的含义，就是根据采购单中的信息，更新采购需求，在采购单中填写采购人员，采购单号，采购的时候，更新采购细节表中的采购人员ID和采购状态。

采购简要流程



领取采购单

<http://localhost:88/api/ware/purchase/received>

(1) 某个人领取了采购单后，先看采购单是否处于未分配状态，只有采购单是新建或以领取状态时，才更新采购单的状态

(2)

【1】仓库列表功能：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/mZgdqOWe>

【2】查询商品库存：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/hwXrEXBZ>

【3】查询采购需求：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/Ss4zsV7R>

【4】合并采购需求：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/cUl9QvK>

【5】查询未领取的采购单：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/hI12DNrH>

【6】领取采购单：<https://easydoc.xyz/doc/75716633/ZUqEdvA4/vXMBBgw1>

完成采购，在完成采购过程中，需要涉及到设置SKU的name信息到仓库中，这是通过远程调用“gulimall-product”来实现根据sku_id查询得到sku_name的，如果这个过程发生了异常，事务不想要回滚，目前采用的方式是通过捕获异常的方式，防止事务回滚，是否还有其他的方式呢？这个问题留待以后解决。



```
@Override
public void addStock(Long skuId, Long wareId, Integer skuNum) {

    List<WareSkuEntity> wareSkuEntities = wareSkuDao.selectList(new
QueryWrapper<WareSkuEntity>().eq("sku_id", skuId).eq("ware_id", wareId));

    if(wareSkuEntities == null || wareSkuEntities.size() == 0 ){
        //新增
        WareSkuEntity wareSkuEntity = new WareSkuEntity();
        wareSkuEntity.setSkuId(skuId);
        wareSkuEntity.setWareId(wareId);
        wareSkuEntity.setStock(skuNum);
        wareSkuEntity.setStockLocked(0);

        //远程查询SKU的name，若失败无需回滚
        try {
            R info = productFeignService.info(skuId);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
    if(info.getCode() == 0){
        Map<String, Object> data=(Map<String, Object>)info.get("skuInfo");
        wareSkuEntity.setSkuName((String) data.get("skuName"));
    }
} catch (Exception e) {

}

wareSkuDao.insert(wareSkuEntity);
}else{
    //插入
    wareSkuDao.addStock(skuId,wareId,skuNum);
}

}
```

30. 获取spu规格

在SPU管理页面，获取商品规格的时候，出现400异常，浏览器显示跳转不了

问题现象：

选择 状态 检索 检索

分类	品牌	重量	上架状态	创建时间	修改时间	操作
225	1	0.2	新建	2020-05-08 16:02:20	2020-05-08 16:02:20	上架 规格

localhost:8001/#/404?spuid=9&catalogId=225

400

出现问题的代码：

```
attrUpdateShow(row) {
  console.log(row);
  this.$router.push({
    path: "/product-attrupdate",
    query: { spuId: row.id, catalogId: row.catalogId }
  });
},
```

暂时不知道如何解决问题。只能留待以后解决。

经过测试发现，问题和上面的代码没有关系，问题出现在“attrupdate.vue”上，该vue页面无法通过浏览器访问，当输入访问URL（<http://localhost:8001/#/product-attrupdate>）的时候，就会出现404，而其他的请求则不会出现这种情况，不知为何。

通过POSTMAN进行请求的时候，能够请求到数据。

经过分析发现，是因为在数据库中没有该页面的导航所导致的，为了修正这个问题，可以在“sys-menu”表中添加一行，内容位：

73	41 商品管理	product/manager		1 zonghe	0
74	42 会员价格	coupon/memberprice		1 admin	0
75	42 每日秒杀	coupon/seckillsession		1 job	0
76	37 规格维护	product/attrupdate	(Null)	1 (Null)	0

这样当再次访问的时候，在“平台属性”下，会出现“规格维护”菜单，



当再次点击“规格”的时候，显示出菜单

A screenshot of a product management application. At the top, there are search filters for 'Category', 'Name', 'Brand', 'Status', and a 'Search' button. Below is a table listing products, with one row selected for a Huawei Mate 30. To the right of the table, a red arrow points from the 'Specification' link in the product details to a modal window. This modal has tabs for 'Specification Maintenance' (highlighted with a red box), 'SPU Management', 'Release Product', 'Product Management', 'Product Inventory', 'Sales Properties', and 'Specification Parameters'. The 'Specification Maintenance' tab is active. Inside the modal, there's a form for modifying product details like 'Network Support' (5G selected) and a 'Quick Preview' checkbox. A red box highlights the 'Basic Information' section of the form. A green 'Confirm Modification' button is at the bottom right of the modal.

不过这种菜单并不符合我们的需要，我们需要让它以弹出框的形式出现。

31. 修改商品规格

API: <https://easydoc.xyz/doc/75716633/ZUqEdvA4/GhnJ0L85>

URL: /product/attr/update/{spuld}

小结：

1. 在open fen中会将调用的数据转换为JSON，接收方接收后，将JSON转换为对象，此时调用方和被调用方的处理JSON的对象不一定都是同一个类，只要它们的字段类型吻合即可。

调用方：

```
@FeignClient(value = "gulimall-coupon")
public interface CouponFenService {

    @PostMapping("/coupon/spubounds/save")
    R saveSpuBounds(@RequestBody SpuBoundTo spuBoundTo);

    @PostMapping("/coupon/skufullreduction/saveInfo")
    R saveSkuReduction(@RequestBody SkuReductionTo skuReductionTo);
}
```

被调用方：

```
    @PostMapping("/save")
    public R save(@RequestBody SpuBoundsEntity spuBounds) {
        spuBoundsService.save(spuBounds);

        return R.ok();
    }

    @PostMapping("/saveInfo")
    public R saveInfo(@RequestBody SkuReductionTo skuReductionTo) {
        skuFullReductionService.saveSkuReduction(skuReductionTo);
        return R.ok();
    }
}
```

调用方JSON化时的对象SpuBoundTo:

```
@Data
public class SpuBoundTo {
    private Long spuId;
    private BigDecimal buyBounds;
    private BigDecimal growBounds;
}
```

被调用方JSON数据对象化时的对象SpuBoundsEntity:

```
/**
 * 商品spu积分设置
 *
 * @author cosmoswong
 * @email cosmoswong@sina.com
 * @date 2020-04-23 23:38:48
 */
@Data
@TableName("sms_spu_bounds")
public class SpuBoundsEntity implements Serializable {
    private static final long serialVersionUID = 1L;
```

```

/**
 * id
 */
@TableId
private Long id;
/** 
 *
 */
private Long spuId;
/** 
 * 成长积分
 */
private BigDecimal growBounds;
/** 
 * 购物积分
 */
private BigDecimal buyBounds;
/** 
 * 优惠生效情况[1111 (四个状态位, 从右到左) ;0 - 无优惠, 成长积分是否赠送;1 - 无优惠, 购物积分是否赠送;2 - 有优惠, 成长积分是否赠送;3 - 有优惠, 购物积分是否赠送【状态位0: 不赠送, 1: 赠送】 ]
 */
private Integer work;

}

```

2. 事务究竟要如何加上?

存在Batch操作的时候，才需要加上事务，单个操作无需添加事务控制。

SpringBoot中的是事务

批量操作的时候，才需要事务

一个事务标注的方法上，方法内存在这些操作：

- (1) 批量更新一个表中字段
- (2) 更新多张表中的操作

实际上不论是哪种类型，方法中所有对于数据库的写操作，都会被整体当做一个事务，在这个事务过程中，如果某个操作出现了异常，则整体都不会被提交。这就是对于SpringBoot中的@Transactional的理解。

@EnableTransactionManagement和@Transactional的区别？

<https://blog.csdn.net/abysscarry/article/details/80189232>

https://blog.csdn.net/Driver_tu/article/details/99679145

<https://www.cnblogs.com/leaveast/p/11765503.html>

其他

1. 文档参考地址

<http://www.jayh.club/#/02.PassJava%E6%9E%B6%E6%9E%84%E7%AF%87/01.%E5%88%9B%E5%BB%BA%E9%A1%B9%E7%9B%AE%E5%92%8C%E6%B7%BB%E5%8A%A0%E6%A8%A1%E5%9D%97>

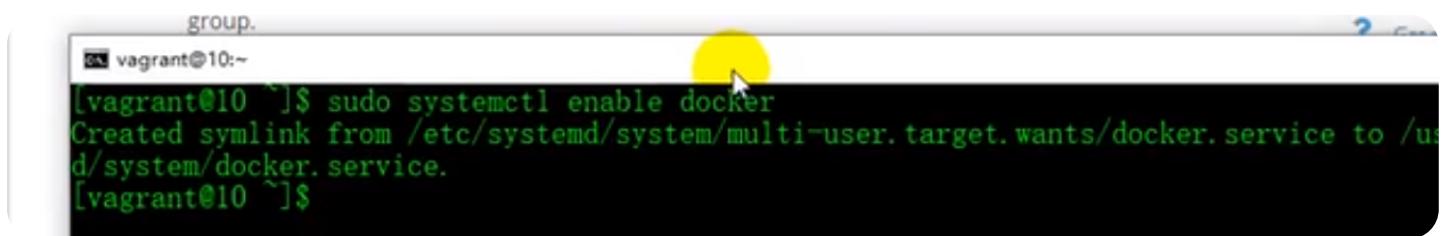
https://blog.csdn.net/ok_wolf/article/details/105400748

<https://www.cnblogs.com/javalbb/p/12690862.html>

https://blog.csdn.net/ok_wolf/article/details/105456170

<https://easydoc.xyz/doc/75716633/ZUqEdvA4/jCFganpf>

2. 开机启动docker



```
group.
[vagrant@10:~]
[vagrant@10 ~]$ sudo systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/docker.service.
[vagrant@10 ~]$
```

在Docker中设置开机启动容器

```
sudo docker update mysql --restart=always
```



#查看防火墙状态

```
[root@hadoop-104 module]# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
    Active: active (running) since Wed 2020-04-22 21:26:23 EDT; 10min ago
      Docs: man:firewalld(1)
 Main PID: 5947 (firewalld)
    CGroup: /system.slice/firewalld.service
           └─5947 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
```

```
Apr 22 21:26:20 hadoop-104 systemd[1]: Starting firewalld - dynamic firewall daemon...
Apr 22 21:26:23 hadoop-104 systemd[1]: Started firewalld - dynamic firewall daemon.
```

```
#查看防火墙是否是开机启动
```

[root@hadoop-104 ~]

#关闭开机启动防火墙

```
[root@hadoop-104 module]# systemctl disable firewalld
```

Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.

Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.

#停止防火墙

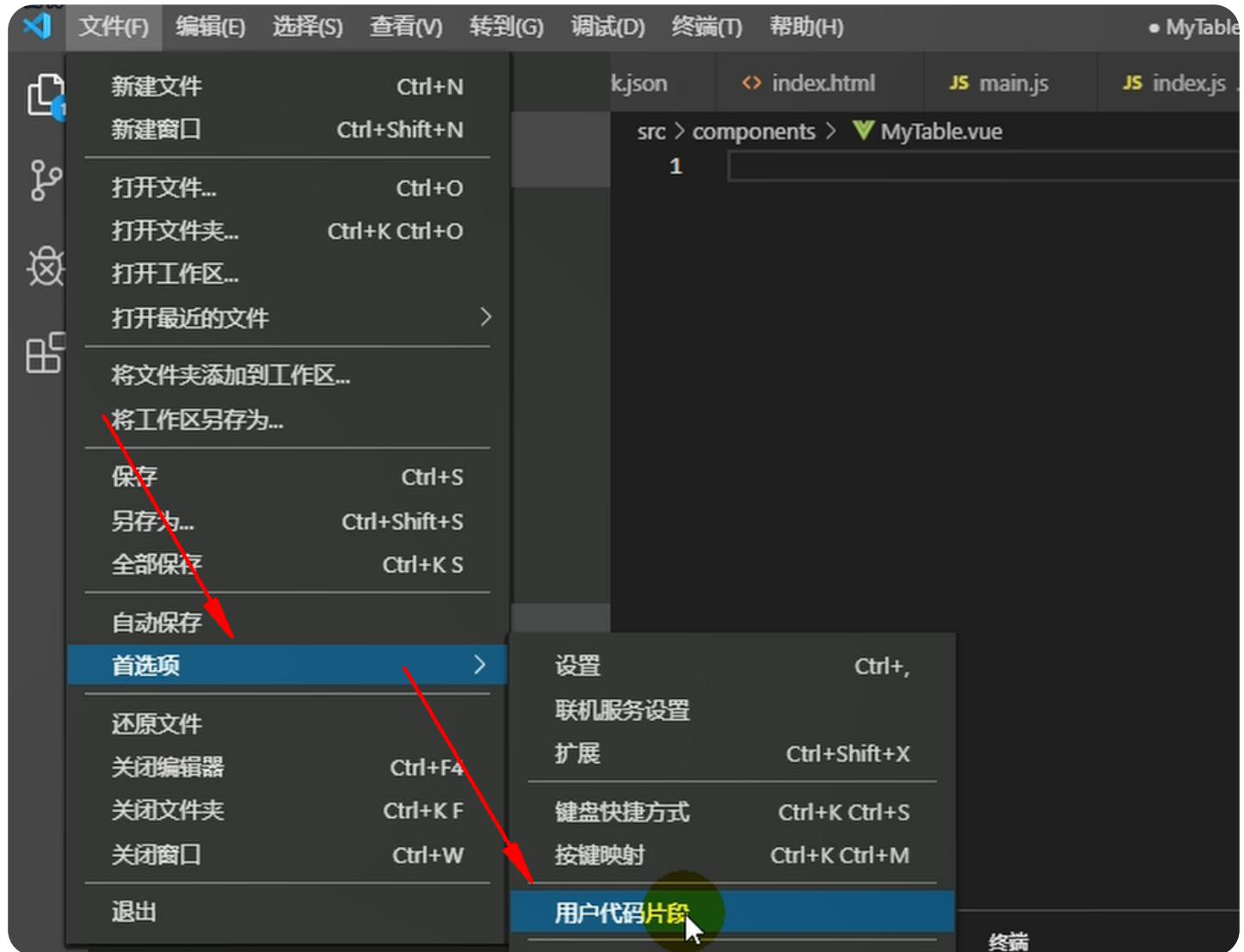
```
[root@hadoop-104 module]# systemctl stop firewalld
```

#再次查看防火墙

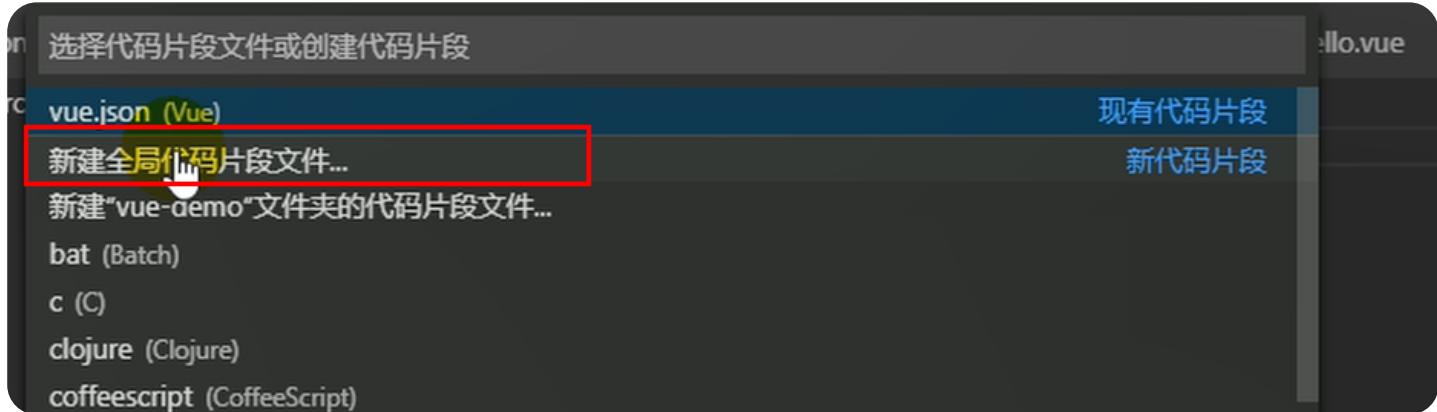
3. 查看命令的安装位置

whereis mysql: 查看mysql的安装位置

4. vscode中生成代码片段



新建一个全局的代码片段，名字为vue，然后回车：



将下面的代码片段粘贴到“vue.code-snippets”

```
{  
    // Place your 全局 snippets here. Each snippet is defined under a snippet name and has  
    // a scope, prefix, body and  
    // description. Add comma separated ids of the languages where the snippet is  
    // applicable in the scope field. If scope  
    // is left empty or omitted, the snippet gets applied to all languages. The prefix is  
    // what is  
    // used to trigger the snippet and the body will be expanded and inserted. Possible  
    // variables are:  
    // $1, $2 for tab stops, $0 for the final cursor position, and ${1:label},  
    // ${2:another} for placeholders.  
    // Placeholders with the same ids are connected.  
    // Example:  
    // "Print to console": {  
    //   "scope": "javascript,typescript",  
    //   "prefix": "log",  
    //   "body": [  
    //     "console.log('$1');",  
    //     "$2"  
    //   ],  
    //   "description": "Log output to console"  
    // }  
    "生成vue模板": {  
        "prefix": "vue",  
        "body": [  
            "<!-- $1 -->",  
            "<template>",  
            "<div class='$2'>$5</div>",  
            "</template>",  
            "<script>$3</script>",  
            "<style>$4</style>"  
        ]  
    }  
}
```

```
"",
"<script>",
"//这里可以导入其他文件（比如：组件，工具js，第三方插件js，json文件，图片文件等等）",
"//例如：import 《组件名称》 from '《组件路径》';",
"",
"export default {",
"//import引入的组件需要注入到对象中才能使用",
"components: {},",
"data() {",
"//这里存放数据",
"return {",
"",
"};",
"},",
"//监听属性 类似于data概念",
"computed: {},",
"//监控data中的数据变化",
"watch: {},",
"//方法集合",
"methods: {},",
"",
"},",
"//生命周期 - 创建完成（可以访问当前this实例）",
"created() {",
"",
"},",
"//生命周期 - 挂载完成（可以访问DOM元素）",
"mounted() {",
"",
"},",
"beforeCreate() {}, //生命周期 - 创建之前",
"beforeMount() {}, //生命周期 - 挂载之前",
"beforeUpdate() {}, //生命周期 - 更新之前",
"updated() {}, //生命周期 - 更新之后",
"beforeDestroy() {}, //生命周期 - 销毁之前",
"destroyed() {}, //生命周期 - 销毁完成",
"activated() {}, //如果页面有keep-alive缓存功能，这个函数会触发",
"}",
"</script>",
"<style lang='scss' scoped>",
"//@import url($3); 引入公共css类",
"${$4}",
"</style>"
],
"description": "生成VUE模板"
},
```

```
"http-get请求": {
    "prefix": "httpget",
    "body": [
        "this.\$http({",
        "url: this.\$http.adornUrl(''),",
        "method: 'get',",
        "params: this.\$http.adornParams({})",
        "}).then(({ data }) => {",
        "})"
    ],
    "description": "httpGET请求"
},
"http-post请求": {
    "prefix": "httppost",
    "body": [
        "this.\$http({",
        "url: this.\$http.adornUrl(''),",
        "method: 'post',",
        "data: this.\$http.adornData(data, false)",
        "}).then(({ data }) => { })"
    ],
    "description": "httpPOST请求"
}
}
```

更多详细说明见: <https://blog.csdn.net/z772330927/article/details/105730430/>

5. vscode快捷键

ctrl+shift+f 全局搜索

alt+shift+f 格式化代码

6. 关闭eslint的语法检查

```
8 |     return path.join(__dirname, '..', dir)
9 |
10|
11|     const createLintingRule = () => ({
12|         //...test: /\.js|vue$/,
13|         //...loader: 'eslint-loader',
14|         //...enforce: 'pre',
15|         //...include: [resolve('src'), resolve('test')],
16|         //...options: {
17|             //....formatter: require('eslint-friendly-formatter'),
18|             //....emitWarning: !config.dev.showEslintErrorsInOverlay
19|         }
20|     })

```

7. 安装mybatisx插件

在Marketplace中搜索“mybatisx”，安装后重启IDEA，使用时会自动在@Mapper标注的接口上，产生小图标，然后 alt+enter，generate statement，就会自动的在xml文件中生成SQL。

```
@Mapper
public interface CategoryBrandRelationDao extends BaseMapper<CategoryBrandRelation> {
    void updateCategory(@Param("catId") Long catId, @Param("brandId") Long brandId);
}
```

- ! Generate statement
- ! Make 'updateCategory()' default
- ! [Mybatis] Generate new statement
- ! Add Javadoc
- R Generate overloaded method with default parameter values

8. mysql的批量删除



```
DELETE FROM `pms_attr_attrgroup_relation` WHERE (attr_id= ? AND attr_group_id ) OR  
(attr_id= ? AND attr_group_id )
```

9. String.join



```
java.lang.String @NotNull  
public static String join(@NotNull CharSequence delimiter,  
                           @NotNull Iterable<? extends CharSequence> elements)
```

Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

返回一个由CharSequence元素的副本和指定分隔符的副本组成的新字符串。

For example,



```
List<String> strings = new LinkedList<>();  
strings.add("Java");strings.add("is");  
strings.add("cool");  
String message = String.join(" ", strings);  
//message returned is: "Java is cool"  
  
Set<String> strings = new LinkedHashSet<>();  
strings.add("Java"); strings.add("is");  
strings.add("very"); strings.add("cool");  
String message = String.join("-", strings);  
//message returned is: "Java-is-very-cool"
```

Note that if an individual element is null, then "null" is added.

注意，如果单个元素为null，则添加“null”。

Params:

delimiter – a sequence of characters that is used to separate each of the elements in the resulting String

用于分隔结果字符串中的每个元素的字符序列

elements – an Iterable that will have its elements joined together.

将其元素连接在一起的可迭代的。

Returns:

a new String that is composed from the elements argument

由elements参数组成的新字符串

Throws:

NullPointerException – If delimiter or elements is null

```
public static String join(CharSequence delimiter,
    Iterable<? extends CharSequence> elements) {
    Objects.requireNonNull(delimiter);
    Objects.requireNonNull(elements);
    StringJoiner joiner = new StringJoiner(delimiter);
    for (CharSequence cs: elements) {
        joiner.add(cs);
    }
    return joiner.toString();
}
```

能够看到实际上它就是通过创建StringJoiner，然后遍历elements，加入每个元素来完成的。

StringJoiner



```
java.util public final class StringJoiner
extends Object
```

StringJoiner is used to construct a sequence of characters separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix.

StringJoiner用于构造由分隔符分隔的字符序列，可以选择以提供的前缀开始，以提供的后缀结束。

Prior to adding something to the StringJoiner, its sj.toString() method will, by default, return prefix + suffix. However, if the setEmptyValue method is called, the emptyValue supplied will be returned instead. This can be used, for example, when creating a string using set notation to indicate an empty set, i.e. "{}", where the prefix is "{}", the suffix is "}" and nothing has been added to the StringJoiner.

在向StringJoiner添加内容之前，它的sj.toString()方法在默认情况下会返回前缀+后缀。但是，如果调用setEmptyValue方法，则返回所提供的emptyValue。例如，当使用set符号创建一个字符串来表示一个空集时，可以使这种办法。“{}”，其中前缀是“{}”，后缀是“}”，没有向StringJoiner添加任何内容。

apiNote:

The String "[George:Sally:Fred]" may be constructed as follows:

```
StringJoiner sj = new StringJoiner(":", "[", "]");
sj.add("George").add("Sally").add("Fred");
String desiredString = sj.toString();
```

A StringJoiner may be employed to create formatted output from a java.util.stream.Stream using java.util.stream.Collectors.joining(CharSequence). For example:

使用StringJoiner从java.util.stream创建格式化输出流，使用java.util.stream.Collectors.joining (CharSequence进行)。例如：

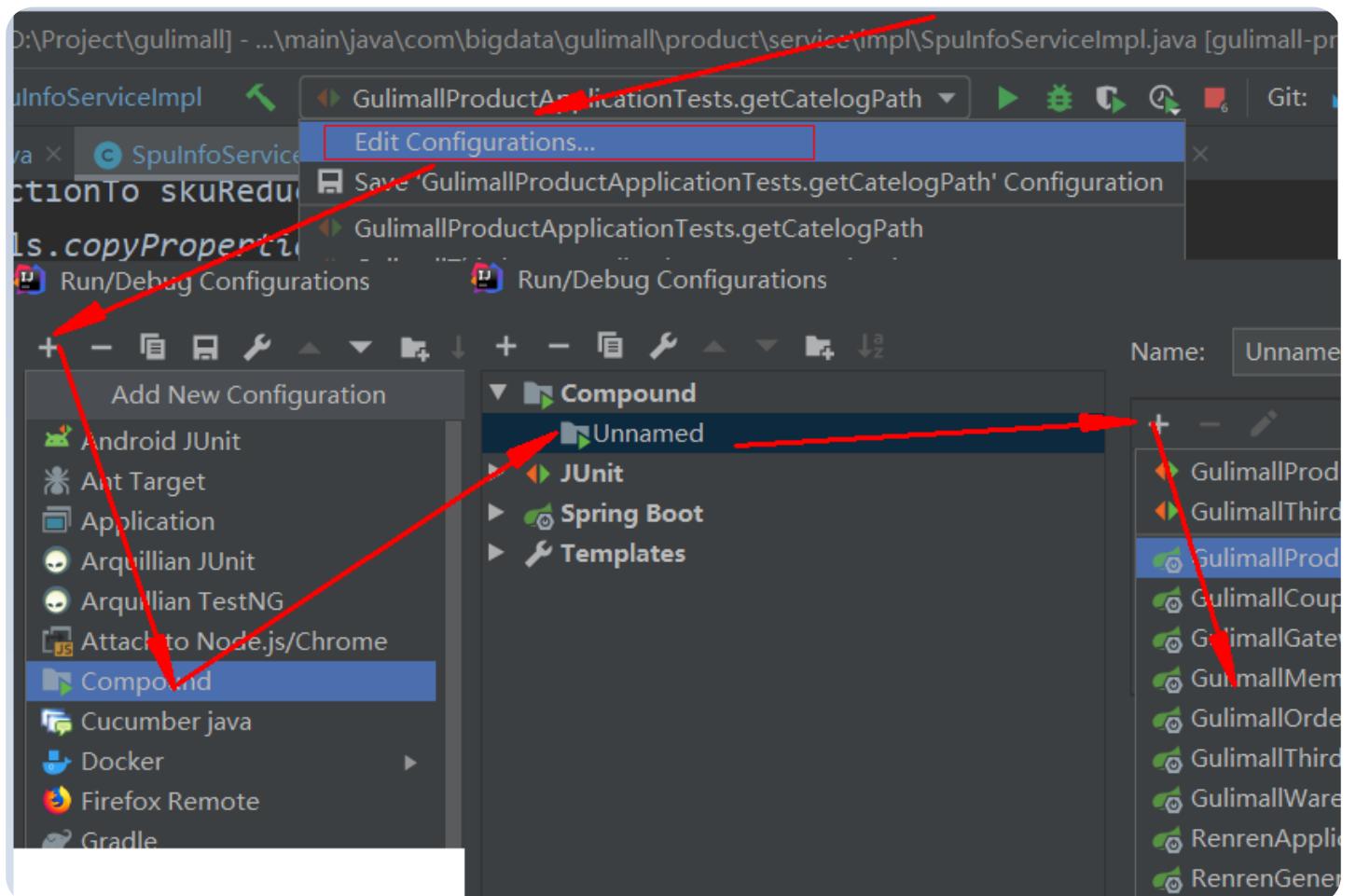
```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
String commaSeparatedNumbers = numbers.stream()
    .map(i -> i.toString())
    .collect(Collectors.joining(", "));
```

通过分析源码发现，在“”内部维护了一个StringBuilder，所有加入到它内部的元素都会先拼接上分割符，然后再拼接上加入的元素

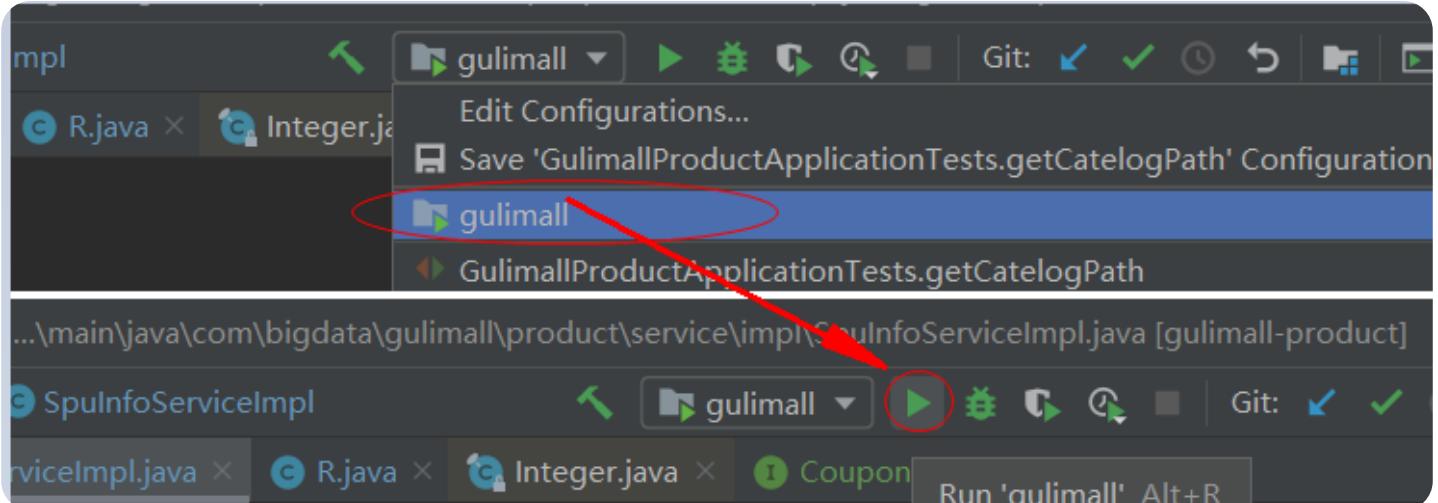
```
public StringJoiner add(CharSequence newElement) {
    prepareBuilder().append(newElement);
    return this;
}
```

```
private StringBuilder prepareBuilder() {  
    if (value != null) {  
        value.append(delimiter);  
    } else {  
        value = new StringBuilder().append(prefix);  
    }  
    return value;  
}
```

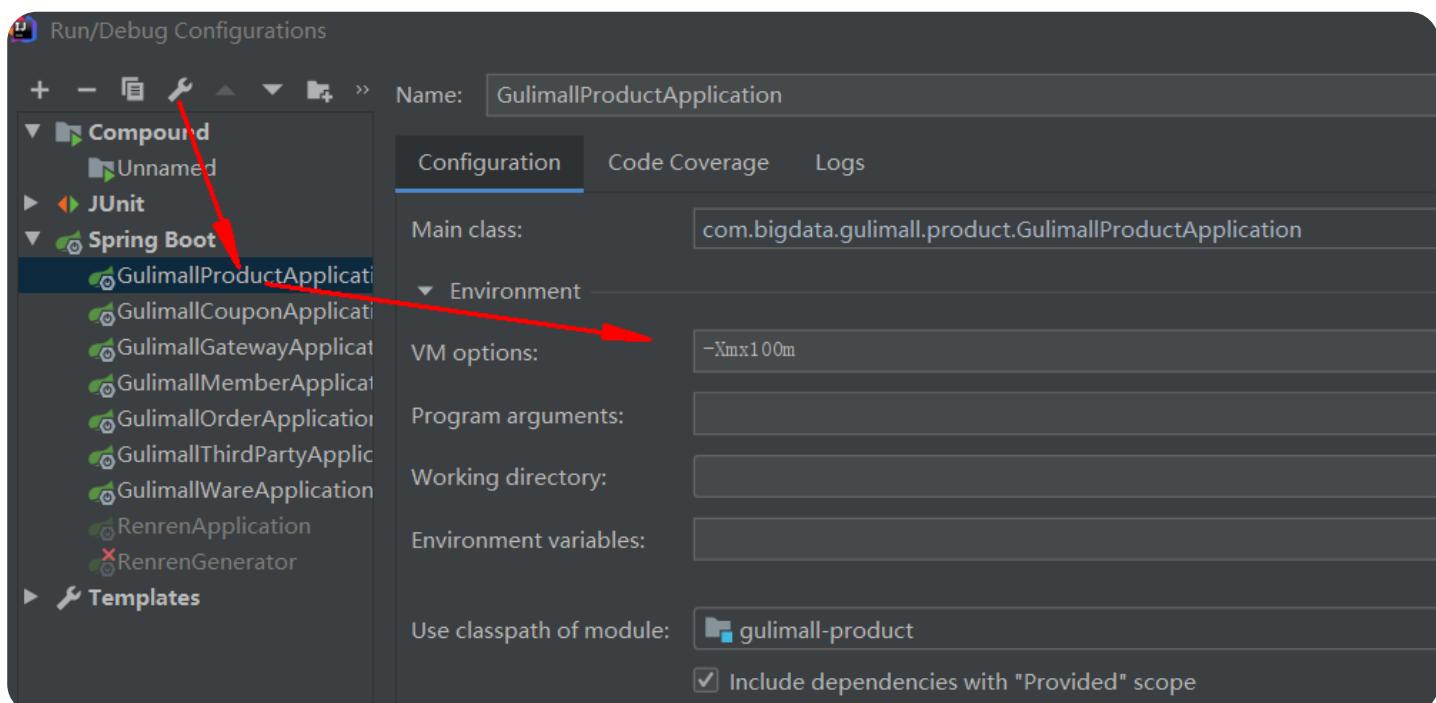
10. 在Service中微服务比较多的时候，可以配置将一些微服务放置到compound中，组成一个小组



以后再运行时，直接选择这个compound即可很方便的运行或停止一组微服务：



另外可以单独为每个微服务，设置需要的运行时最大堆内存大小：



11. mysql的dateTime和timestamp的区别？

[MySQL中datetime和timestamp的区别及使用](#)

TIMESTAMP和DATETIME的相同点：

1> 两者都可用来表示YYYY-MM-DD HH:MM:SS[.fraction]类型的日期。

TIMESTAMP和DATETIME的不同点：

1> 两者的存储方式不一样

对于TIMESTAMP，它把客户端插入的时间从当前时区转化为UTC（世界标准时间）进行存储。查询时，将其又转化为客户端当前时区进行返回。

而对于DATETIME，不做任何改变，基本上是原样输入和输出。

2> 两者所能存储的时间范围不一样

timestamp所能存储的时间范围为：'1970-01-01 00:00:01.000000' 到 '2038-01-19 03:14:07.999999'。

datetime所能存储的时间范围为：'1000-01-01 00:00:00.000000' 到 '9999-12-31 23:59:59.999999'。

总结：TIMESTAMP和DATETIME除了存储范围和存储方式不一样，没有太大区别。当然，对于跨时区的业务，TIMESTAMP更为合适。

<https://www.cnblogs.com/Jashinck/p/10472398.html>

12. SpringBoot中的事务

https://blog.csdn.net/Z_Sheng/article/details/89489053

13. IDEA RESTFULL clinet

[IntelliJ IDEA 使用 rest client](#)

###

FAQ

1. TypeError: _vm.previewHandle is not a function

1. ELASTICSEARCH

1、安装elastic search

2、初步检索

 1) _CAT

 2) 索引一个文档

 3) 查看文档

 4) 更新文档

 5) 删除文档或索引

 6) elasticsearch的批量操作——bulk

 7) 样本测试数据

3、检索

 1) search Api

 2) Query DSL

 (1) 基本语法格式

 (2) 返回部分字段

 (3) match匹配查询

 (4) match_phrase [短句匹配]

 (5) multi_math 【多字段匹配】

 (6) bool用来做复合查询

 (7) Filter 【结果过滤】

 (8) term

 (9) Aggregation (执行聚合)

3) Mapping

 (1) 字段类型

 (2) 映射

 (3) 新版本改变

 创建映射

 查看映射

 添加新的字段映射

 更新映射

 数据迁移

4) 分词

 (1) 安装ik分词器

 (1) 查看elasticsearch版本号：

 (2) 进入es容器内部plugin目录

 (2) 测试分词器

 (3) 自定义词库

4、elasticsearch-Rest-Client

 1) 9300: TCP

 2) 9200: HTTP

5、附录：安装Nginx

SpringBoot整合ElasticSearch

1、导入依赖

- 2、编写测试类
 - 1) 测试保存数据
 - 2) 测试获取数据

其他

1. kibana控制台命令

1. ELASTICSEARCH

1、安装elastic search

dokcer中安装elasticsearch

- (1) 下载elasticsearch和kibana



```
docker pull elasticsearch:7.6.2  
docker pull kibana:7.6.2
```

- (2) 配置



```
mkdir -p /mydata/elasticsearch/config  
mkdir -p /mydata/elasticsearch/data  
echo "http.host: 0.0.0.0" >/mydata/elasticsearch/config/elasticsearch.yml  
chmod -R 777 /mydata/elasticsearch/
```

- (3) 启动Elastic search

```
docker run --name elasticsearch -p 9200:9200 -p 9300:9300 \
-e "discovery.type=single-node" \
-e ES_JAVA_OPTS="-Xms64m -Xmx512m" \
-v
/mydata/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml \
-v /mydata/elasticsearch/data:/usr/share/elasticsearch/data \
-v /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins \
-d elasticsearch:7.6.2
```

设置开机启动elasticsearch

```
docker update elasticsearch --restart=always
```

(4) 启动kibana:

```
docker run --name kibana -e ELASTICSEARCH_HOSTS=http://192.168.137.14:9200 -p 5601:5601 \
-d kibana:7.6.2
```

设置开机启动kibana

```
docker update kibana --restart=always
```

(5) 测试

查看elasticsearch版本信息: <http://192.168.137.14:9200/>

```
{  
  "name": "0adeb7852e00",  
  "cluster_name": "elasticsearch",  
  "cluster_uuid": "9gglpP0HTfyOTRAaSe2rIg",  
  "version": {  
    "number": "7.6.2",  
    "build_flavor": "default",  
    "build_type": "docker",  
    "build_hash": "ef48eb35cf30adf4db14086e8aabd07ef6fb113f",  
    "build_date": "2020-03-26T06:34:37.794943Z",  
    "build_snapshot": false,  
    "lucene_version": "8.4.0",  
    "minimum_wire_compatibility_version": "6.8.0",  
    "minimum_index_compatibility_version": "6.0.0-beta1"  
  },  
  "tagline": "You Know, for Search"  
}
```

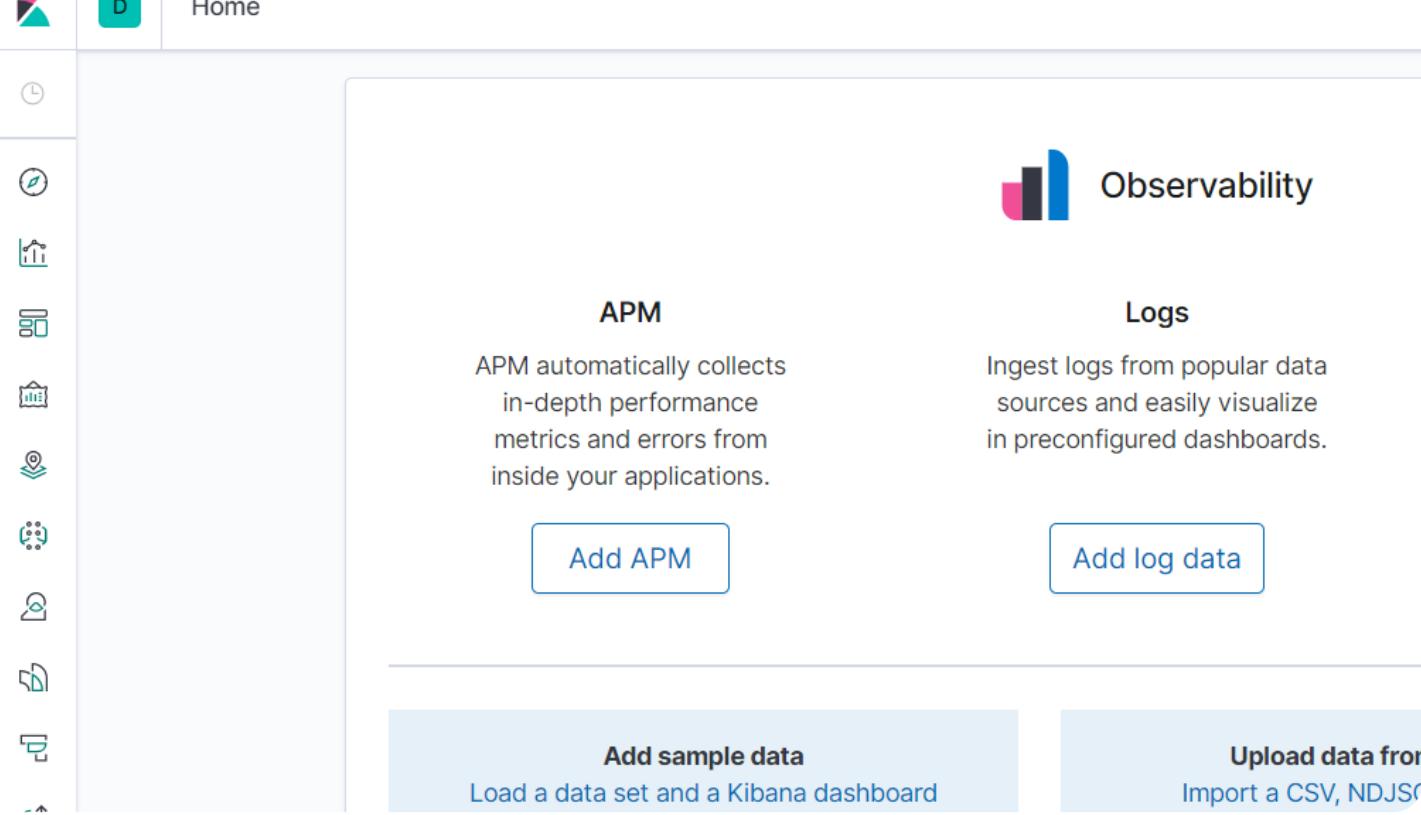
显示elasticsearch 节点信息http://192.168.137.14:9200/_cat/nodes ,



```
127.0.0.1 76 95 1 0.26 1.40 1.22 dilm * 0adeb7852e00
```

访问Kibana: <http://192.168.137.14:5601/app/kibana>

Not secure | 192.168.137.14:5601/app/kibana#/home



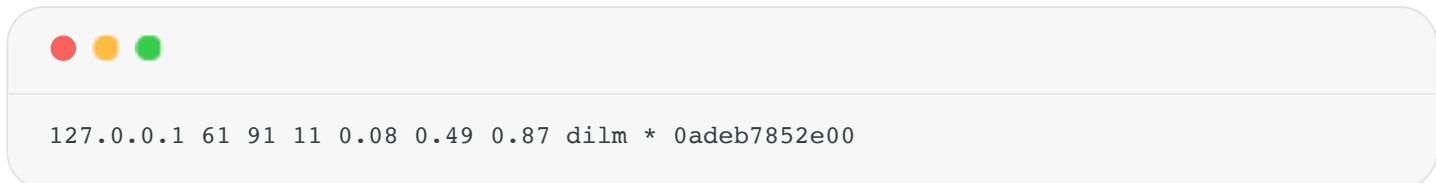
The screenshot shows the Kibana home page. On the left is a sidebar with various icons: a magnifying glass, a document, a house, a gear, a bar chart, a scatter plot, a line graph, a funnel, a gear, a user icon, a gear, a bar chart, and a downward arrow. At the top right, there's a blue button labeled 'D' and a link to 'Home'. Below the sidebar, there are two main sections: 'APM' and 'Logs'. The 'APM' section contains the text: 'APM automatically collects in-depth performance metrics and errors from inside your applications.' with a 'Add APM' button. The 'Logs' section contains the text: 'Ingest logs from popular data sources and easily visualize in preconfigured dashboards.' with a 'Add log data' button. At the bottom center is a light blue button labeled 'Add sample data Load a data set and a Kibana dashboard'. To its right is another light blue button labeled 'Upload data from Import a CSV, NDJSON'.

2、初步检索

1) _CAT

(1) GET/cat/nodes: 查看所有节点

如: http://192.168.137.14:9200/_cat/nodes:



```
127.0.0.1 61 91 11 0.08 0.49 0.87 dilm * 0adeb7852e00
```

注: *表示集群中的主节点

(2) GET/cat/health: 查看es健康状况

如: http://192.168.137.14:9200/_cat/health



```
1588332616 11:30:16 elasticsearch green 1 1 3 3 0 0 0 0 - 100.0%
```

注: green表示健康值正常

(3) GET/cat/master: 查看主节点

如: http://192.168.137.14:9200/_cat/master



```
vfpqxbusTC6-W3C2Np31EQ 127.0.0.1 127.0.0.1 0adeb7852e00
```

(4) GET/_cat/indices: 查看所有索引 , 等价于mysql数据库的show databases;

如: http://192.168.137.14:9200/_cat/indices



```
green open .kibana_task_manager_1 KWLtjcKRRuaV9so_v15WYg 1 0 2 0 39.8kb 39.8kb
green open .apm-agent-configuration cuwCpJ5ER0OYsSgAJ7bVYA 1 0 0 0 283b 283b
green open .kibana_1 PqK_LdUYRpWMy4fK0tMSPw 1 0 7 0 31.2kb 31.2kb
```

2) 索引一个文档

保存一个数据, 保存在哪个索引的哪个类型下, 指定用那个唯一标识

PUT customer/external/1;在customer索引下的external类型下保存1号数据为



```
PUT customer/external/1
```



```
{
  "name": "John Doe"
}
```

PUT和POST都可以

POST新增。如果不指定id，会自动生成id。指定id就会修改这个数据，并新增版本号；

PUT可以新增也可以修改。PUT必须指定id；由于PUT需要指定id，我们一般用来做修改操作，不指定id会报错。

下面是在postman中的测试数据：

The screenshot shows a POST request in Postman. The URL is `http://192.168.137.14:9200/customer/external/1`. The request method is highlighted with a red box. The body is set to `JSON (application/json)` and contains the following JSON:

```
1 {  
2   "name": "John Doe"  
3 }
```

The response status is `201 Created` and the time taken is `226 ms`. The response body is displayed in the JSON tab:

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 1,  
6   "result": "created",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 0,  
13  "_primary_term": 1  
14}
```

创建数据成功后，显示201 created表示插入记录成功。

```
{  
  "_index": "customer",  
  "_type": "external",  
  "_id": "1",  
  "_version": 1,  
  "result": "created",  
  "_shards": {  
    "total": 2,  
    "successful": 1,  
    "failed": 0  
  },  
  "_seq_no": 0,  
  "_primary_term": 1  
}
```

这些返回的JSON串的含义；这些带有下划线开头的，称为元数据，反映了当前的基本信息。

"_index": "customer" 表明该数据在哪个数据库下;

"_type": "external" 表明该数据在哪个类型下;

"_id": "1" 表明被保存数据的id;

"_version": 1, 被保存数据的版本

"result": "created" 这里是创建了一条数据, 如果重新put一条数据, 则该状态会变为updated, 并且版本号也会发生变化。

下面选用POST方式:

添加数据的时候, 不指定ID, 会自动的生成id, 并且类型是新增:

The screenshot shows a Postman request configuration. The method is set to POST, and the URL is `http://192.168.137.14:9200/customer/external/`. The Body tab is selected, and the content type is set to `JSON (application/json)`. The raw JSON payload is:

```
1 {  
2   "name": "John Doe"  
3 }
```

The response body is displayed in the Body tab under the JSON tab. It shows the created document with an auto-generated ID and version, and the result is "created".

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "L1MX0HEBHYK_MJXUv4JB",  
5   "_version": 1,  
6   "result": "created",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 2,  
13  "_primary_term": 1
```

再次使用POST插入数据, 仍然是新增的:

POST ▼ http://192.168.137.14:9200/customer/external/

Authorization Headers (1) Body ● Pre-request Script Tests

● form-data ● x-www-form-urlencoded ● raw ● binary JSON (application/json) ▾

```
1 {  
2   "name": "John Doe"  
3 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview JSON ▾

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "id": "LyMY0HEBHYK MJXU24I0",  
5   "version": 1,  
6   "result": "created",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 3,  
13  "_primary_term": 1  
14 }
```

添加数据的时候，指定ID，会使用该ID，并且类型是新增：

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://192.168.137.14:9200/customer/external/2
- Body (JSON):**

```
1 {  
2   "name": "John Doe"  
3 }
```
- Response Body (Pretty JSON):**

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "2",  
5   "_version": 1,  
6   "result": "created",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 4,  
13  "_primary_term": 1
```

再次使用POST插入数据，类型为updated

The screenshot shows a POST request to `http://192.168.137.14:9200/customer/external/2`. The request body is a JSON object with a single field `"name": "John Doe"`. The response body is a JSON document indicating a successful update:

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "2",  
5   "_version": 2,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 5,  
13  "_primary_term": 1  
14 }
```

3) 查看文档

GET /customer/external/1

<http://192.168.137.14:9200/customer/external/1>



```
{  
    "_index": "customer", //在哪个索引  
    "_type": "external", //在哪个类型  
    "_id": "1", //记录id  
    "_version": 3, //版本号  
    "_seq_no": 6, //并发控制字段，每次更新都会+1，用来做乐观锁  
    "_primary_term": 1, //同上，主分片重新分配，如重启，就会变化  
    "found": true,  
    "_source": {  
        "name": "John Doe"  
    }  
}
```

通过“if_seq_no=1&if_primary_term=1”，当序列号匹配的时候，才进行修改，否则不修改。

实例：将id=1的数据更新为name=1，然后再次更新为name=2，起始seq_no=6, primary_term=1

(1) 将name更新为1

http://192.168.137.14:9200/customer/external/1?if_seq_no=6&if_primary_term=1

PUT ▾ http://192.168.137.14:9200/customer/external/1?if_seq_no=6&if_primary_term=1

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json) ▾

```
1 [ {  
2   "name": "1"  
3 }]
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON ▾

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 4,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 7,  
13  "_primary_term": 1  
14 }
```

(2) 将name更新为2, 更新过程中使用seq_no=6

http://192.168.137.14:9200/customer/external/1?if_seq_no=6&if_primary_term=1

PUT http://192.168.137.14:9200/customer/external/1?if_seq_no=6&if_primary_term=1

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 [{}  
2 {"name": "2"}  
3 ]
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON

```
1 {  
2   "error": {  
3     "root_cause": [  
4       {  
5         "type": "version_conflict_engine_exception",  
6         "reason": "[1]: version conflict, required seqNo [6], primary term [1]. current document has seqNo [7] and primary term [1]",  
7         "index_uuid": "nzDYCdnvQjSsapJrAIT8Zw",  
8         "shard": "0",  
9         "index": "customer"  
10      }  
11    ],  
12    "type": "version_conflict_engine_exception",  
13    "reason": "[1]: version conflict, required seqNo [6], primary term [1]. current document has seqNo [7] and primary term [1]",  
14    "index_uuid": "nzDYCdnvQjSsapJrAIT8Zw",  
15    "shard": "0",  
16    "index": "customer"  
17  },  
18  "status": 409
```

出现更新错误。

(3) 查询新的数据

<http://192.168.137.14:9200/customer/external/1>

GET ▾ http://192.168.137.14:9200/customer/external/1

Authorization Headers Body Pre-request Script Tests

TYPE Inherit auth from parent

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

This request is not in

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON ▾

```
1 {  
2     "_index": "customer",  
3     "_type": "external",  
4     "_id": "1",  
5     "_version": 4,  
6     "_seq_no": 7,     
7     "_primary_term": 1,  
8     "found": true,  
9     "_source": {  
10         "name": "1"  
11     }  
12 }
```

能够看到_seq_no变为7。

(4) 再次更新，更新成功

http://192.168.137.14:9200/customer/external/1?if_seq_no=7&if_primary_term=1

PUT ▾ http://192.168.137.14:9200/customer/external/1?if_seq_no=7&if_primary_term=1

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json) ▾

```
1 [{  
2   "name": "2"  
3 }]
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON ▾

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 5,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 8,  
13  "_primary_term": 1  
14 }
```

4) 更新文档

```
POST customer/external/1/_update
```

```
{  
  "doc":{  
    "name": "John Doe"  
  }  
}
```

或者

```
POST customer/external/1
```

```
{  
  "name": "John Doe2"  
}
```

或者

```
PUT customer/external/1
```

```
{  
  "name": "John Doe"  
}
```

- 不同：POST 操作会对比源文档数据，如果相同不会有操作，文档 version 不增加
PUT 操作总会将数据重新保存并增加 version 版本；

带_update 对比元数据如果一样就不进行任何操作。

看场景：

对于大并发更新，不带 update；

对于大并发查询偶尔更新，带 update；对比更新，重新计算分配规则。

- 更新同时增加属性

18668062061

```
POST customer/external/1/_update
```

```
{  
  "doc":{ "name": "Jane Doe", "age": 20 }  
}
```

(1) POST更新文档，带有_update

http://192.168.137.14:9200/customer/external/1/_update

POST ▾ http://192.168.137.14:9200/customer/external/1/_update

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json) ▾

```
1 {  
2   "doc":{  
3     "name":"john"  
4   }  
5 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON ▾

```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 6,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 9,  
13  "_primary_term": 1
```

如果再次执行更新，则不执行任何操作，序列号也不发生变化

POST ▾

http://192.168.137.14:9200/customer/external/1/_update

Authorization

Headers (1)

Body ●

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json) ▾

```
1 [{}  
2   "doc":{  
3     "name":"john"  
4   }  
5 }]
```

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

JSON ▾



```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 6,  
6   "result": "noop",  
7   "_shards": {  
8     "total": 0,  
9     "successful": 0,  
10    "failed": 0  
11  },  
12  "_seq_no": 9,  
13  "_primary_term": 1
```

POST更新方式，会对比原来的数据，和原来的相同，则不执行任何操作（version和_seq_no）都不变。

(2) POST更新文档，不带_update

POST

http://192.168.137.14:9200/customer/external/1/

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

```
1 {  
2   "doc": {  
3     "name": "john"  
4   }  
5 }
```

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

JSON



```
1 {  
2   "_index": "customer",  
3   "_type": "external",  
4   "_id": "1",  
5   "_version": 7,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 10,  
13  "_primary_term": 1
```

在更新过程中，重复执行更新操作，数据也能够更新成功，不会和原来的数据进行对比。

5) 删除文档或索引



```
DELETE customer/external/1  
DELETE customer
```

注：elasticsearch并没有提供删除类型的操作，只提供了删除索引和文档的操作。

实例：删除id=1的数据，删除后继续查询

The screenshot shows two parallel Postman requests side-by-side.

Left Request (DELETE):

- Method: **DELETE**
- URL: **http://192.168.137.14:9200/customer/external/1/**
- Authorization: **Inherit auth from parent**
- Body: **Pretty** view shows a JSON response with a red box around the "result": "deleted" field.

```

1 {
2   "_index": "customer",
3   "_type": "external",
4   "_id": "1",
5   "_version": 10,
6   "result": "deleted",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12  "_seq_no": 13,
13  "_primary_term": 1
  
```

Right Request (GET):

- Method: **GET**
- URL: **http://192.168.137.14:9200/customer/external/1**
- Authorization: **Inherit auth from parent**
- Body: **Pretty** view shows a JSON response with a red box around the "found": false field.

```

1 {
2   "_index": "customer",
3   "_type": "external",
4   "_id": "1",
5   "found": false
6 }
  
```

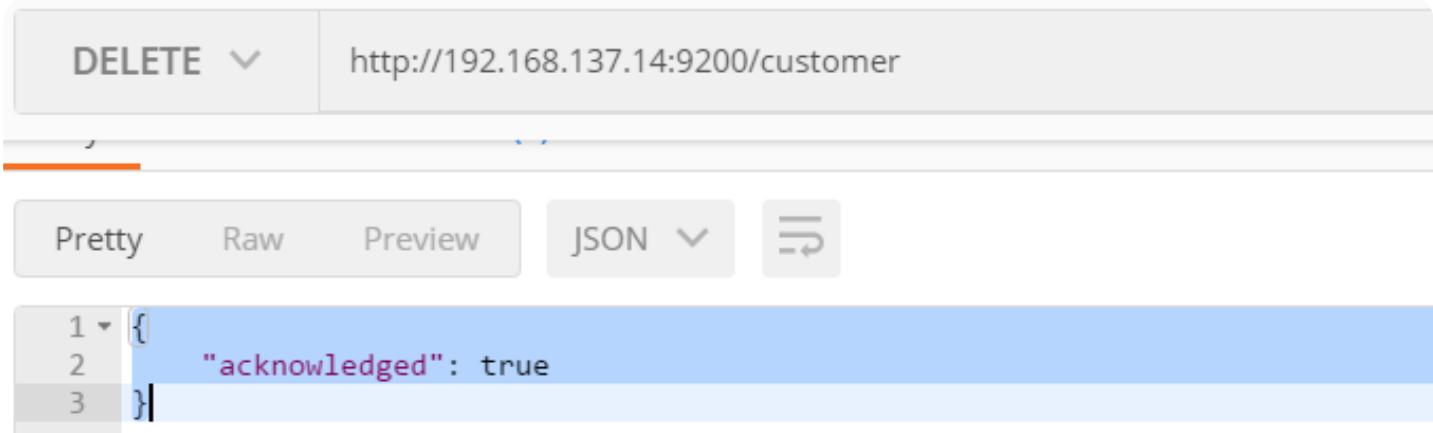
实例：删除整个costomer索引数据

删除前，所有的索引

The screenshot shows the Kibana interface displaying a list of indices:

- green open .kibana_task_manager_1 KWLtjcKRRuaV9so_v15WYg 1 0 2 0 39.8kb 39.8kb
- green open .apm-agent-configuration cuwCpJ5ER0OYsSgAJ7bVYA 1 0 0 0 283b 283b
- green open .kibana_1 PqK_LdUYRpWMy4fK0tMSPw 1 0 7 0 31.2kb 31.2kb
- yellow open customer nzDYCdnnQjSsapJrAIT8Zw 1 1 4 0 4.4kb 4.4kb

删除“customer”索引



DELETE ▾ http://192.168.137.14:9200/customer

Pretty Raw Preview JSON ▾

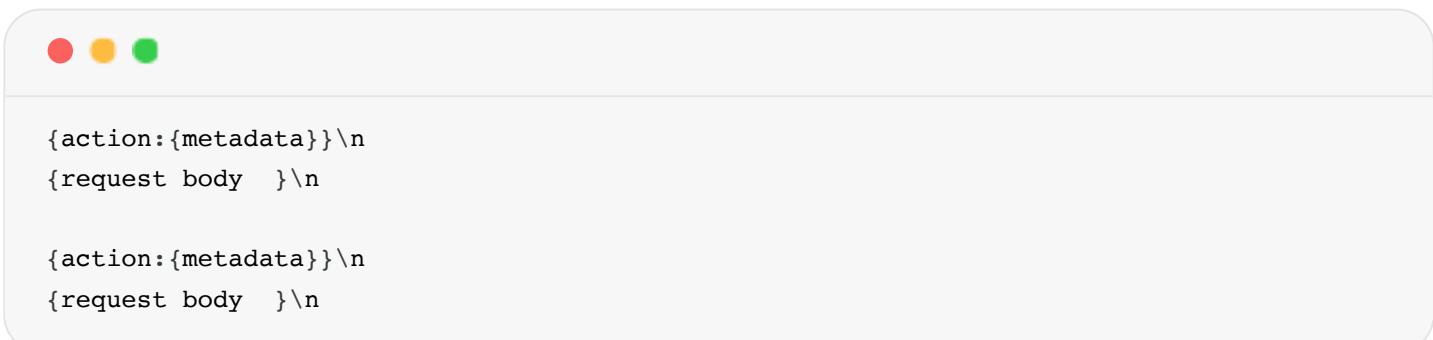
```
1 {  
2   "acknowledged": true  
3 }
```

删除后，所有的索引

```
green open .kibana_task_manager_1 KWLtjcKRRuaV9so_v15WYg 1 0 2 0 39.8kb 39.8kb  
green open .apm-agent-configuration cuwCpJ5ER0OYsSgAJ7bVYA 1 0 0 0 283b 283b  
green open .kibana_1 PqK_LdUYRpWMy4fK0tMSPw 1 0 7 0 31.2kb 31.2kb
```

6) elasticsearch的批量操作——bulk

语法格式：



```
{action:{metadata}}\n{request body }\n\n{action:{metadata}}\n{request body }\n
```

这里的批量操作，当发生某一条执行发生失败时，其他的数据仍然能够接着执行，也就是说彼此之间是独立的。

bulk api以此按顺序执行所有的action（动作）。如果一个单个的动作因任何原因失败，它将继续处理它后面剩余的动作。当bulk api返回时，它将提供每个动作的状态（与发送的顺序相同），所以您可以检查是否一个指定的动作是否失败了。

实例1：执行多条数据

```
POST customer/_bulk
{"index": {"_id": "1"}}
{"name": "John Doe"}
{"index": {"_id": "2"}}
{"name": "John Doe"}
```

执行结果

```
#! Deprecation: [types removal] Specifying types in bulk requests is deprecated.
{
  "took" : 491,
  "errors" : false,
  "items" : [
    {
      "index" : {
        "_index" : "customer",
        "_type" : "external",
        "_id" : "1",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "_seq_no" : 0,
        "_primary_term" : 1,
        "status" : 201
      }
    },
    {
      "index" : {
        "_index" : "customer",
        "_type" : "external",
        "_id" : "2",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
```

```
        "failed" : 0
    },
    "_seq_no" : 1,
    "_primary_term" : 1,
    "status" : 201
}
]
}
```

实例2：对于整个索引执行批量操作

```
POST /_bulk
{"delete": {"_index": "website", "_type": "blog", "_id": "123"}}
{"create": {"_index": "website", "_type": "blog", "_id": "123"}}
{"title": "my first blog post"}
{"index": {"_index": "website", "_type": "blog"}}
{"title": "my second blog post"}
{"update": {"_index": "website", "_type": "blog", "_id": "123"}}
{"doc": {"title": "my updated blog post"}}
```

运行结果：

```
#! Deprecation: [types removal] Specifying types in bulk requests is deprecated.
{
  "took" : 608,
  "errors" : false,
  "items" : [
    {
      "delete" : {
        "_index" : "website",
        "_type" : "blog",
        "_id" : "123",
        "_version" : 1,
        "result" : "not_found",
        "_shards" : {
          "total" : 2,
```

```
        "successful" : 1,
        "failed" : 0
    },
    "_seq_no" : 0,
    "_primary_term" : 1,
    "status" : 404
}
},
{
"create" : {
    "_index" : "website",
    "_type" : "blog",
    "_id" : "123",
    "_version" : 2,
    "result" : "created",
    "_shards" : {
        "total" : 2,
        "successful" : 1,
        "failed" : 0
    },
    "_seq_no" : 1,
    "_primary_term" : 1,
    "status" : 201
}
},
{
"index" : {
    "_index" : "website",
    "_type" : "blog",
    "_id" : "MCOs0HEBHYK_MJXUyYIz",
    "_version" : 1,
    "result" : "created",
    "_shards" : {
        "total" : 2,
        "successful" : 1,
        "failed" : 0
    },
    "_seq_no" : 2,
    "_primary_term" : 1,
    "status" : 201
}
},
{
"update" : {
    "_index" : "website",
    "_type" : "blog",

```

```
        "_id" : "123",
        "_version" : 3,
        "result" : "updated",
        "_shards" : {
            "total" : 2,
            "successful" : 1,
            "failed" : 0
        },
        "_seq_no" : 3,
        "_primary_term" : 1,
        "status" : 200
    }
}
]
```

7) 样本测试数据

准备了一份顾客银行账户信息的虚构的JSON文档样本。每个文档都有下列的schema（模式）。

```
{  
    "account_number": 1,  
    "balance": 39225,  
    "firstname": "Amber",  
    "lastname": "Duke",  
    "age": 32,  
    "gender": "M",  
    "address": "880 Holmes Lane",  
    "employer": "Pyrami",  
    "email": "amberduke@pyrami.com",  
    "city": "Brogan",  
    "state": "IL"  
}
```

<https://github.com/elastic/elasticsearch/blob/master/docs/src/test/resources/accounts.json>，导入测试数据，

POST bank/account/_bulk

3、检索

1) search Api

ES支持两种基本方式检索：

- 通过REST request uri 发送搜索参数 (uri +检索参数) ；
- 通过REST request body 来发送它们 (uri+请求体) ；

信息检索

● 一切检索从 `_search` 开始

<code>GET bank/_search</code>	检索 bank 下所有信息，包括 type 和 docs
<code>GET bank/_search?q=*&sort=account_number:asc</code>	请求参数方式检索

响应结果解释：

`took - Elasticsearch` 执行搜索的时间 (毫秒)

`time_out` - 告诉我们搜索是否超时

`_shards` - 告诉我们多少个分片被搜索了，以及统计了成功/失败的搜索分片

`hits` - 搜索结果

`hits.total` - 搜索结果

`hits.hits` - 实际的搜索结果数组 (默认为前 10 的文档)

`sort` - 结果的排序 key (键) (没有则按 `score` 排序)

`score` 和 `max_score` - 相关性得分和最高得分 (全文检索用)

● uri+请求体进行检索

```
GET bank/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "account_number": {
        "order": "desc"
      }
    }
  ]
}
```

HTTP 客户端工具（POSTMAN），get 请求不能携带请求体，我们变为 post 也是一样的
我们 POST 一个 JSON 风格的查询请求体到 _search API。

需要了解，一旦搜索的结果被返回，Elasticsearch 就完成了这次请求，并且不会维护任何

HTTP 客户端工具（POSTMAN），get 请求不能携带请求体，我们变为 post 也是一样的
我们 POST 一个 JSON 风格的查询请求体到 _search API。

需要了解，一旦搜索的结果被返回，Elasticsearch 就完成了这次请求，并且不会维护任何
服务端的资源或者结果的 cursor (游标) 18668062

uri+请求体进行检索



```
GET /bank/_search
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" },
    {"balance": "desc"}
  ]
}
```

HTTP客户端工具 () , get请求不能够携带请求体,



```
GET bank/_search?q=*&sort=account_number:asc
```

返回结果:



```
{
  "took" : 235,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "0",
        "_score" : null,
        "_source" : {
          "account_number" : 0,
          "balance" : 16623,
          "firstname" : "Bradshaw",
          "lastname" : "Mckenzie",
          "age" : 29,
          "gender" : "F",
          "address" : "123 Main St, Anytown, USA"
        }
      }
    ]
  }
}
```

```
    "address" : "244 Columbus Place",
    "employer" : "Euron",
    "email" : "bradshawmckenzie@euron.com",
    "city" : "Hobucken",
    "state" : "CO"
},
"sort" : [
  0
]
},
{
  "_index" : "bank",
  "_type" : "account",
  "_id" : "1",
  "_score" : null,
  "_source" : {
    "account_number" : 1,
    "balance" : 39225,
    "firstname" : "Amber",
    "lastname" : "Duke",
    "age" : 32,
    "gender" : "M",
    "address" : "880 Holmes Lane",
    "employer" : "Pyrami",
    "email" : "amberduke@pyrami.com",
    "city" : "Brogan",
    "state" : "IL"
},
"sort" : [
  1
]
},
{
  "_index" : "bank",
  "_type" : "account",
  "_id" : "2",
  "_score" : null,
  "_source" : {
    "account_number" : 2,
    "balance" : 28838,
    "firstname" : "Roberta",
    "lastname" : "Bender",
    "age" : 22,
    "gender" : "F",
    "address" : "560 Kingsway Place",
    "employer" : "Chillium",
    "city" : "Kingsway"
}
}
```

```
        "email" : "robertabender@chillium.com",
        "city" : "Bennett",
        "state" : "LA"
    },
    "sort" : [
        2
    ]
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "3",
    "_score" : null,
    "_source" : {
        "account_number" : 3,
        "balance" : 44947,
        "firstname" : "Levine",
        "lastname" : "Burks",
        "age" : 26,
        "gender" : "F",
        "address" : "328 Wilson Avenue",
        "employer" : "Amtap",
        "email" : "levineburks@amtap.com",
        "city" : "Cochranville",
        "state" : "HI"
    },
    "sort" : [
        3
    ]
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "4",
    "_score" : null,
    "_source" : {
        "account_number" : 4,
        "balance" : 27658,
        "firstname" : "Rodriquez",
        "lastname" : "Flores",
        "age" : 31,
        "gender" : "F",
        "address" : "986 Wyckoff Avenue",
        "employer" : "Tourmania",
        "email" : "rodriquezflores@tourmania.com",
        "city" : "Eastvale",
        "state" : "NY"
    }
}
```

```
        "state" : "HI"
    },
    "sort" : [
        4
    ]
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "5",
    "_score" : null,
    "_source" : {
        "account_number" : 5,
        "balance" : 29342,
        "firstname" : "Leola",
        "lastname" : "Stewart",
        "age" : 30,
        "gender" : "F",
        "address" : "311 Elm Place",
        "employer" : "Diginetic",
        "email" : "leolastewart@diginetic.com",
        "city" : "Fairview",
        "state" : "NJ"
    },
    "sort" : [
        5
    ]
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "6",
    "_score" : null,
    "_source" : {
        "account_number" : 6,
        "balance" : 5686,
        "firstname" : "Hattie",
        "lastname" : "Bond",
        "age" : 36,
        "gender" : "M",
        "address" : "671 Bristol Street",
        "employer" : "Netagy",
        "email" : "hattiebond@netagy.com",
        "city" : "Dante",
        "state" : "TN"
    }
},
```

```
"sort" : [
  6
]
},
{
  "_index" : "bank",
  "_type" : "account",
  "_id" : "7",
  "_score" : null,
  "_source" : {
    "account_number" : 7,
    "balance" : 39121,
    "firstname" : "Levy",
    "lastname" : "Richard",
    "age" : 22,
    "gender" : "M",
    "address" : "820 Logan Street",
    "employer" : "Teraprene",
    "email" : "levyrichard@teraprene.com",
    "city" : "Shrewsbury",
    "state" : "MO"
  },
  "sort" : [
    7
  ]
},
{
  "_index" : "bank",
  "_type" : "account",
  "_id" : "8",
  "_score" : null,
  "_source" : {
    "account_number" : 8,
    "balance" : 48868,
    "firstname" : "Jan",
    "lastname" : "Burns",
    "age" : 35,
    "gender" : "M",
    "address" : "699 Visitation Place",
    "employer" : "Glasstep",
    "email" : "janburns@glasstep.com",
    "city" : "Wakulla",
    "state" : "AZ"
  },
  "sort" : [
    8
  ]
}
```

```

        ],
    },
    {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "9",
        "_score" : null,
        "_source" : {
            "account_number" : 9,
            "balance" : 24776,
            "firstname" : "Opal",
            "lastname" : "Meadows",
            "age" : 39,
            "gender" : "M",
            "address" : "963 Neptune Avenue",
            "employer" : "Cedward",
            "email" : "opalmeadows@cedward.com",
            "city" : "Olney",
            "state" : "OH"
        },
        "sort" : [
            9
        ]
    }
]
}
}

```

(1) 只有6条数据，这是因为存在分页查询；

(2) 详细的字段信息，参照：<https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started-search.html>

The response also provides the following information about the search request:

- `took` – how long it took Elasticsearch to run the query, in milliseconds
- `timed_out` – whether or not the search request timed out
- `_shards` – how many shards were searched and a breakdown of how many shards succeeded, failed, or were skipped.
- `max_score` – the score of the most relevant document found

- `hits.total.value` - how many matching documents were found
- `hits.sort` - the document's sort position (when not sorting by relevance score)
- `hits._score` - the document's relevance score (not applicable when using `match_all`)

2) Query DSL

(1) 基本语法格式

Elasticsearch提供了一个可以执行查询的Json风格的DSL。这个被称为Query DSL，该查询语言非常全面。

一个查询语句的典型结构

```
QUERY_NAME:{  
    ARGUMENT:VALUE,  
    ARGUMENT:VALUE,...  
}
```

如果针对于某个字段，那么它的结构如下：

```
{  
    QUERY_NAME:{  
        FIELD_NAME:{  
            ARGUMENT:VALUE,  
            ARGUMENT:VALUE,...  
        }  
    }  
}
```

```
GET bank/_search  
{
```

```
"query": {
    "match_all": {}
},
"from": 0,
"size": 5,
"sort": [
    {
        "account_number": {
            "order": "desc"
        }
    }
]
}
```

query定义如何查询；

- match_all查询类型【代表查询所有的所有】，es中可以在query中组合非常多的查询类型完成复杂查询；
- 除了query参数之外，我们也可传递其他的参数以改变查询结果，如sort, size；
- from+size限定，完成分页功能；
- sort排序，多字段排序，会在前序字段相等时后续字段内部排序，否则以前序为准；

(2) 返回部分字段

```
GET bank/_search
{
    "query": {
        "match_all": {}
    },
    "from": 0,
    "size": 5,
    "sort": [
        {
            "account_number": {
                "order": "desc"
            }
        }
    ],
    "_source": [ "balance", "firstname" ]
}
```

```
}
```

查询结果：



```
{
  "took" : 18,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "999",
        "_score" : null,
        "_source" : {
          "firstname" : "Dorothy",
          "balance" : 6087
        },
        "sort" : [
          999
        ]
      },
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "998",
        "_score" : null,
        "_source" : {
          "firstname" : "Letha",
          "balance" : 16869
        }
      }
    ]
  }
}
```

```
        },
        "sort" : [
          998
        ]
      },
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "997",
        "_score" : null,
        "_source" : {
          "firstname" : "Combs",
          "balance" : 25311
        },
        "sort" : [
          997
        ]
      },
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "996",
        "_score" : null,
        "_source" : {
          "firstname" : "Andrews",
          "balance" : 17541
        },
        "sort" : [
          996
        ]
      },
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "995",
        "_score" : null,
        "_source" : {
          "firstname" : "Phelps",
          "balance" : 21153
        },
        "sort" : [
          995
        ]
      }
    ]
  }
```

```
}
```

(3) match匹配查询

- 基本类型（非字符串），精确控制



```
GET bank/_search
{
  "query": {
    "match": {
      "account_number": "20"
    }
  }
}
```

match返回account_number=20的数据。

查询结果：



```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "index" : "bank",
        "score" : 1.0,
        "type" : "account",
        "id" : 1,
        "account_number" : "20"
      }
    ]
}
```

```

    "_index" : "bank",
    "_type" : "account",
    "_id" : "20",
    "_score" : 1.0,
    "_source" : {
        "account_number" : 20,
        "balance" : 16418,
        "firstname" : "Elinor",
        "lastname" : "Ratliff",
        "age" : 36,
        "gender" : "M",
        "address" : "282 Kings Place",
        "employer" : "Scentric",
        "email" : "elinorratliff@scentric.com",
        "city" : "Ribera",
        "state" : "WA"
    }
}
]
}
}

```

- 字符串，全文检索

●

```

GET bank/_search
{
  "query": {
    "match": {
      "address": "kings"
    }
  }
}

```

全文检索，最终会按照评分进行排序，会对检索条件进行分词匹配。

查询结果：



```
{  
    "took" : 30,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 1,  
        "successful" : 1,  
        "skipped" : 0,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : {  
            "value" : 2,  
            "relation" : "eq"  
        },  
        "max_score" : 5.990829,  
        "hits" : [  
            {  
                "_index" : "bank",  
                "_type" : "account",  
                "_id" : "20",  
                "_score" : 5.990829,  
                "_source" : {  
                    "account_number" : 20,  
                    "balance" : 16418,  
                    "firstname" : "Elinor",  
                    "lastname" : "Ratliff",  
                    "age" : 36,  
                    "gender" : "M",  
                    "address" : "282 Kings Place",  
                    "employer" : "Scentric",  
                    "email" : "elinorratliff@scentric.com",  
                    "city" : "Ribera",  
                    "state" : "WA"  
                }  
            },  
            {  
                "_index" : "bank",  
                "_type" : "account",  
                "_id" : "722",  
                "_score" : 5.990829,  
                "_source" : {  
                    "account_number" : 722,  
                    "balance" : 27256,  
                    "firstname" : "Roberts",  
                    "lastname" : "Kings",  
                    "age" : 36,  
                    "gender" : "M",  
                    "address" : "282 Kings Place",  
                    "employer" : "Scentric",  
                    "email" : "roberts@scentric.com",  
                    "city" : "Ribera",  
                    "state" : "WA"  
                }  
            }  
        ]  
    }  
}
```

```
        "lastname" : "Beasley",
        "age" : 34,
        "gender" : "F",
        "address" : "305 Kings Hwy",
        "employer" : "Quintity",
        "email" : "robertsbeasley@quintity.com",
        "city" : "Hayden",
        "state" : "PA"
    }
}
]
}
}
```

(4) match_phrase [短句匹配]

将需要匹配的值当成一整个单词（不分词）进行检索

```
GET bank/_search
{
  "query": {
    "match_phrase": {
      "address": "mill road"
    }
  }
}
```

查处address中包含mill_road的所有记录，并给出相关性得分

查看结果：

```
{
  "took" : 32,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
```

```
"skipped" : 0,
"failed" : 0
},
"hits" : {
  "total" : {
    "value" : 1,
    "relation" : "eq"
  },
  "max_score" : 8.926605,
  "hits" : [
    {
      "_index" : "bank",
      "_type" : "account",
      "_id" : "970",
      "_score" : 8.926605,
      "_source" : {
        "account_number" : 970,
        "balance" : 19648,
        "firstname" : "Forbes",
        "lastname" : "Wallace",
        "age" : 28,
        "gender" : "M",
        "address" : "990 Mill Road",
        "employer" : "Pheast",
        "email" : "forbeswallace@pheast.com",
        "city" : "Lopezoz",
        "state" : "AK"
      }
    }
  ]
}
}
```

match_phrase和Match的区别，观察如下实例：

```
GET bank/_search
{
  "query": {
    "match_phrase": {
      "address": "990 Mill"
    }
  }
}
```

查询结果：

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 10.806405,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "970",
        "_score" : 10.806405,
        "_source" : {
          "account_number" : 970,
          "balance" : 19648,
          "firstname" : "Forbes",
          "lastname" : "Wallace",
          "age" : 28,
          "gender" : "M",
          "address" : "990 Mill Road",
          "city" : "London"
        }
      }
    ]
}
```

```
        "employer" : "Pheast",
        "email" : "forbeswallace@pheast.com",
        "city" : "Lopezó",
        "state" : "AK"
    }
}
]
}
}
```

使用match的keyword

```
● ○ ●
GET bank/_search
{
  "query": {
    "match": {
      "address.keyword": "990 Mill"
    }
  }
}
```

查询结果，一条也未匹配到

```
● ○ ●
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
      "relation" : "eq"
    },
    "hits" : [
      {
        "id" : 1,
        "name" : "John Doe"
      }
    ]
  }
}
```

```
        "max_score" : null,  
        "hits" : [ ]  
    }  
}
```

修改匹配条件为“990 Mill Road”

```
GET bank/_search
{
  "query": {
    "match": {
      "address.keyword": "990 Mill Road"
    }
  }
}
```

查询出一条数据

```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 1,  
      "relation" : "eq"  
    },  
    "max_score" : 6.5032897,  
    "hits" : [  
      {  
        "_index" : "bank",  
        "type" : "account",  
        "id" : 1,  
        "score" : 6.5032897,  
        "fields" : {  
          "name" : {  
            "value" : "Bank of America"  
          }  
        }  
      },  
      {  
        "_index" : "bank",  
        "type" : "account",  
        "id" : 2,  
        "score" : 6.5032897,  
        "fields" : {  
          "name" : {  
            "value" : "Wells Fargo"  
          }  
        }  
      },  
      {  
        "_index" : "bank",  
        "type" : "account",  
        "id" : 3,  
        "score" : 6.5032897,  
        "fields" : {  
          "name" : {  
            "value" : "JP Morgan Chase"  
          }  
        }  
      },  
      {  
        "_index" : "bank",  
        "type" : "account",  
        "id" : 4,  
        "score" : 6.5032897,  
        "fields" : {  
          "name" : {  
            "value" : "Citi"  
          }  
        }  
      },  
      {  
        "_index" : "bank",  
        "type" : "account",  
        "id" : 5,  
        "score" : 6.5032897,  
        "fields" : {  
          "name" : {  
            "value" : "U.S. Bank"  
          }  
        }  
      }  
    ]  
  }  
}
```

```
"_id" : "970",
"_score" : 6.5032897,
"_source" : {
    "account_number" : 970,
    "balance" : 19648,
    "firstname" : "Forbes",
    "lastname" : "Wallace",
    "age" : 28,
    "gender" : "M",
    "address" : "990 Mill Road",
    "employer" : "Pheast",
    "email" : "forbeswallace@pheast.com",
    "city" : "Lopezo",
    "state" : "AK"
}
}
]
}
}
```

文本字段的匹配，使用keyword，匹配的条件就是要显示字段的全部值，要进行精确匹配的。

match_phrase是做短语匹配，只要文本中包含匹配条件，就能匹配到。

(5) multi_math 【多字段匹配】

```
GET bank/_search
{
  "query": {
    "multi_match": {
      "query": "mill",
      "fields": [
        "state",
        "address"
      ]
    }
  }
}
```

state或者address中包含mill，并且在查询过程中，会对于查询条件进行分词。

查询结果：

```
{
  "took" : 28,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 5.4032025,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "970",
        "_score" : 5.4032025,
        "_source" : {
          "account_number" : 970,
          "balance" : 19648,
          "branch" : "West Branch",
          "customer_id" : 1234567890,
          "date_opened" : "2018-01-01T00:00:00Z",
          "last_transaction" : "2018-01-01T00:00:00Z",
          "opening_balance" : 1000000000000000000.0
        }
      }
    ]
  }
}
```

```
        "firstname" : "Forbes",
        "lastname" : "Wallace",
        "age" : 28,
        "gender" : "M",
        "address" : "990 Mill Road",
        "employer" : "Pheast",
        "email" : "forbeswallace@pheast.com",
        "city" : "Lopezo",
        "state" : "AK"
    },
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "136",
    "_score" : 5.4032025,
    "_source" : {
        "account_number" : 136,
        "balance" : 45801,
        "firstname" : "Winnie",
        "lastname" : "Holland",
        "age" : 38,
        "gender" : "M",
        "address" : "198 Mill Lane",
        "employer" : "Neteria",
        "email" : "winnieholland@neteria.com",
        "city" : "Urie",
        "state" : "IL"
    },
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "345",
    "_score" : 5.4032025,
    "_source" : {
        "account_number" : 345,
        "balance" : 9812,
        "firstname" : "Parker",
        "lastname" : "Hines",
        "age" : 38,
        "gender" : "M",
        "address" : "715 Mill Avenue",
        "employer" : "Baluba",
        "email" : "parkerhines@baluba.com",
        "city" : "Blackgum",
    }
},
```

```
        "state" : "KY"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "472",
    "_score" : 5.4032025,
    "_source" : {
        "account_number" : 472,
        "balance" : 25571,
        "firstname" : "Lee",
        "lastname" : "Long",
        "age" : 32,
        "gender" : "F",
        "address" : "288 Mill Street",
        "employer" : "Comverges",
        "email" : "leelong@comverges.com",
        "city" : "Movico",
        "state" : "MT"
    }
}
]
}
}
```

(6) bool用来做复合查询

复合语句可以合并，任何其他查询语句，包括符合语句。这也就意味着，复合语句之间可以互相嵌套，可以表达非常复杂的逻辑。

must: 必须达到must所列举的所有条件

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": { "address": "mill" }},
        {"match": { "gender": "M" }}
      ]
    }
  }
}
```

must_not, 必须不匹配must_not所列举的所有条件。

should, 应该满足should所列举的条件。

实例：查询gender=m， 并且address=mill的数据

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "gender": "M"
          }
        },
        {
          "match": {
            "address": "mill"
          }
        }
      ]
    }
  }
}
```

查询结果：



```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 3,  
      "relation" : "eq"  
    },  
    "max_score" : 6.0824604,  
    "hits" : [  
      {  
        "_index" : "bank",  
        "_type" : "account",  
        "_id" : "970",  
        "_score" : 6.0824604,  
        "_source" : {  
          "account_number" : 970,  
          "balance" : 19648,  
          "firstname" : "Forbes",  
          "lastname" : "Wallace",  
          "age" : 28,  
          "gender" : "M",  
          "address" : "990 Mill Road",  
          "employer" : "Pheast",  
          "email" : "forbeswallace@pheast.com",  
          "city" : "Lopezo",  
          "state" : "AK"  
        }  
      },  
      {  
        "_index" : "bank",  
        "_type" : "account",  
        "_id" : "136",  
        "_score" : 6.0824604,  
        "_source" : {  
          "account_number" : 136,  
          "balance" : 45801,  
          "firstname" : "John",  
          "lastname" : "Doe",  
          "age" : 34,  
          "gender" : "M",  
          "address" : "123 Main Street",  
          "employer" : "Acme Corp.",  
          "email" : "johndoe@acme.com",  
          "city" : "Anytown",  
          "state" : "CA"  
        }  
      }  
    ]  
  }  
}
```

```

        "firstname" : "Winnie",
        "lastname" : "Holland",
        "age" : 38,
        "gender" : "M",
        "address" : "198 Mill Lane",
        "employer" : "Neteria",
        "email" : "winnieholland@neteria.com",
        "city" : "Urie",
        "state" : "IL"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "345",
    "_score" : 6.0824604,
    "_source" : {
        "account_number" : 345,
        "balance" : 9812,
        "firstname" : "Parker",
        "lastname" : "Hines",
        "age" : 38,
        "gender" : "M",
        "address" : "715 Mill Avenue",
        "employer" : "Baluba",
        "email" : "parkerhines@baluba.com",
        "city" : "Blackgum",
        "state" : "KY"
    }
}
]
}
}

```

must_not: 必须不是指定的情况

实例：查询gender=m，并且address=mill的数据，但是age不等于38的



```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "gender": "M"
          }
        },
        {
          "match": {
            "address": "mill"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "age": "38"
          }
        }
      ]
    }
  }
}
```

查询结果：



```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "hits" : [
      {
        "id" : 1,
        "name" : "John Doe",
        "age" : 38,
        "gender" : "M",
        "address" : "mill"
      }
    ]
  }
}
```

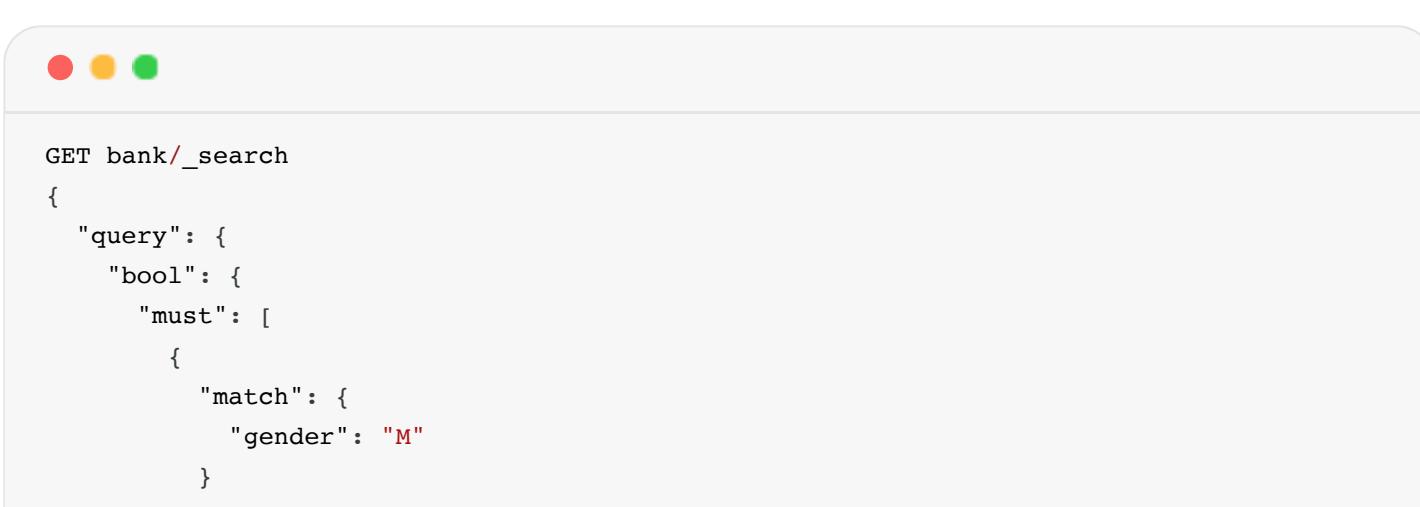
```

"max_score" : 6.0824604,
"hits" : [
  {
    "_index" : "bank",
    "_type" : "account",
    "_id" : "970",
    "_score" : 6.0824604,
    "_source" : {
      "account_number" : 970,
      "balance" : 19648,
      "firstname" : "Forbes",
      "lastname" : "Wallace",
      "age" : 28,
      "gender" : "M",
      "address" : "990 Mill Road",
      "employer" : "Pheast",
      "email" : "forbeswallace@pheast.com",
      "city" : "Lopezo",
      "state" : "AK"
    }
  }
]
}
}

```

should: 应该达到**should**列举的条件，如果到达会增加相关文档的评分，并不会改变查询的结果。如果**query**中只有**should**且只有一种匹配规则，那么**should**的条件就会被作为默认匹配条件二区改变查询结果。

实例：匹配lastName应该等于Wallace的数据



```

GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "gender": "M"
          }
        }
      ]
    }
  }
}

```

```
        },
        {
          "match": {
            "address": "mill"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "age": "18"
          }
        }
      ],
      "should": [
        {
          "match": {
            "lastname": "Wallace"
          }
        }
      ]
    }
  }
}
```

查询结果：

```
  {
    "took" : 5,
    "timed_out" : false,
    "_shards" : {
      "total" : 1,
      "successful" : 1,
      "skipped" : 0,
      "failed" : 0
    },
    "hits" : {
      "total" : {
        "value" : 3,
        "relation" : "eq"
      },
      "hits" : [
        {
          "id" : 1,
          "name" : "John Doe",
          "age" : 25,
          "address" : "mill"
        },
        {
          "id" : 2,
          "name" : "Jane Doe",
          "age" : 25,
          "address" : "mill"
        },
        {
          "id" : 3,
          "name" : "Alice Smith",
          "age" : 25,
          "address" : "mill"
        }
      ]
    }
  }
```

```
"max_score" : 12.585751,
"hits" : [
  {
    "_index" : "bank",
    "_type" : "account",
    "_id" : "970",
    "_score" : 12.585751,
    "_source" : {
      "account_number" : 970,
      "balance" : 19648,
      "firstname" : "Forbes",
      "lastname" : "Wallace",
      "age" : 28,
      "gender" : "M",
      "address" : "990 Mill Road",
      "employer" : "Pheast",
      "email" : "forbeswallace@pheast.com",
      "city" : "Lopezo",
      "state" : "AK"
    }
  },
  {
    "_index" : "bank",
    "_type" : "account",
    "_id" : "136",
    "_score" : 6.0824604,
    "_source" : {
      "account_number" : 136,
      "balance" : 45801,
      "firstname" : "Winnie",
      "lastname" : "Holland",
      "age" : 38,
      "gender" : "M",
      "address" : "198 Mill Lane",
      "employer" : "Neteria",
      "email" : "winnieholland@neteria.com",
      "city" : "Urie",
      "state" : "IL"
    }
  },
  {
    "_index" : "bank",
    "_type" : "account",
    "_id" : "345",
    "_score" : 6.0824604,
    "_source" : {
```

```
        "account_number" : 345,
        "balance" : 9812,
        "firstname" : "Parker",
        "lastname" : "Hines",
        "age" : 38,
        "gender" : "M",
        "address" : "715 Mill Avenue",
        "employer" : "Baluba",
        "email" : "parkerhines@baluba.com",
        "city" : "Blackgum",
        "state" : "KY"
    }
}
]
}
}
```

能够看到相关度越高，得分也越高。

(7) Filter 【结果过滤】

并不是所有的查询都需要产生分数，特别是哪些仅用于filtering过滤的文档。为了不计算分数，elasticsearch会自动检查场景并且优化查询的执行。



```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "address": "mill"
          }
        }
      ],
      "filter": {
        "range": {
          "balance": {
            "gte": "10000",
            "lte": "20000"
          }
        }
      }
    }
  }
}
```

```
        }
    }
}
}
```

这里先是查询所有匹配address=mill的文档，然后再根据10000<=balance<=20000进行过滤查询结果

查询结果：



```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 5.4032025,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "970",
        "_score" : 5.4032025,
        "_source" : {
          "account_number" : 970,
          "balance" : 19648,
          "firstname" : "Forbes",
          "lastname" : "Wallace",
          "age" : 28,
          "gender" : "M",
          "address" : "990 Mill Road",
          "employer" : "Pheast",
          "email" : "forbeswallace@pheast.com",
          "city" : "Lopezo",
          "state" : "Mississippi"
        }
      }
    ]
}
```

```
        "state" : "AK"
    }
}
]
}
}
```

Each `must`, `should`, and `must_not` element in a Boolean query is referred to as a query clause. How well a document meets the criteria in each `must` or `should` clause contributes to the document's *relevance score*. The higher the score, the better the document matches your search criteria. By default, Elasticsearch returns documents ranked by these relevance scores.

在boolean查询中，`must`, `should` 和 `must_not` 元素都被称为查询子句。文档是否符合每个“must”或“should”子句中的标准，决定了文档的“相关性得分”。得分越高，文档越符合您的搜索条件。默认情况下，Elasticsearch返回根据这些相关性得分排序的文档。

The criteria in a `must_not` clause is treated as a *filter*. It affects whether or not the document is included in the results, but does not contribute to how documents are scored. You can also explicitly specify arbitrary filters to include or exclude documents based on structured data.

“must_not”子句中的条件被视为“过滤器”。它影响文档是否包含在结果中，但不影响文档的评分方式。还可以显式地指定任意过滤器来包含或排除基于结构化数据的文档。

filter在使用过程中，并不会计算相关性得分：



```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "address": "mill"
          }
        }
      ],
      "filter": {
        "range": {
          "balance": {
            "gte": "10000",
            "lt": "50000"
          }
        }
      }
    }
  }
}
```

```
        "lte": "20000"
    }
}
}
}
}
```

查询结果：



```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 213,
      "relation" : "eq"
    },
    "max_score" : 0.0,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "20",
        "_score" : 0.0,
        "_source" : {
          "account_number" : 20,
          "balance" : 16418,
          "firstname" : "Elinor",
          "lastname" : "Ratliff",
          "age" : 36,
          "gender" : "M",
          "address" : "282 Kings Place",
          "employer" : "Scentric",
          "email" : "elinor.ratliff@scentric.com",
          "city" : "Ribera",
          "state" : "WA"
        }
      }
    ]
  }
}
```

```
        }
    },
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "37",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 37,
        "balance" : 18612,
        "firstname" : "Mcgee",
        "lastname" : "Mooney",
        "age" : 39,
        "gender" : "M",
        "address" : "826 Fillmore Place",
        "employer" : "Reversus",
        "email" : "mcgeemooney@reversus.com",
        "city" : "Tooleville",
        "state" : "OK"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "51",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 51,
        "balance" : 14097,
        "firstname" : "Burton",
        "lastname" : "Meyers",
        "age" : 31,
        "gender" : "F",
        "address" : "334 River Street",
        "employer" : "Bezal",
        "email" : "burtonmeyers@bezal.com",
        "city" : "Jacksonburg",
        "state" : "MO"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "56",
    "_score" : 0.0,
    "_source" : {
```

```
        "account_number" : 56,
        "balance" : 14992,
        "firstname" : "Josie",
        "lastname" : "Nelson",
        "age" : 32,
        "gender" : "M",
        "address" : "857 Tabor Court",
        "employer" : "Emtrac",
        "email" : "josienelson@emtrac.com",
        "city" : "Sunnyside",
        "state" : "UT"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "121",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 121,
        "balance" : 19594,
        "firstname" : "Acevedo",
        "lastname" : "Dorsey",
        "age" : 32,
        "gender" : "M",
        "address" : "479 Nova Court",
        "employer" : "Netropic",
        "email" : "acevedodorsey@netropic.com",
        "city" : "Islandia",
        "state" : "CT"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "176",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 176,
        "balance" : 18607,
        "firstname" : "Kemp",
        "lastname" : "Walters",
        "age" : 28,
        "gender" : "F",
        "address" : "906 Howard Avenue",
        "employer" : "Eyewax",
    }
}
```

```
        "email" : "kempwalters@eyewax.com",
        "city" : "Why",
        "state" : "KY"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "183",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 183,
        "balance" : 14223,
        "firstname" : "Hudson",
        "lastname" : "English",
        "age" : 26,
        "gender" : "F",
        "address" : "823 Herkimer Place",
        "employer" : "Xinware",
        "email" : "hudsonenglish@xinware.com",
        "city" : "Robbins",
        "state" : "ND"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "222",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 222,
        "balance" : 14764,
        "firstname" : "Rachelle",
        "lastname" : "Rice",
        "age" : 36,
        "gender" : "M",
        "address" : "333 Narrows Avenue",
        "employer" : "Enaut",
        "email" : "rachellerice@enaut.com",
        "city" : "Wright",
        "state" : "AZ"
    }
},
{
    "_index" : "bank",
    "_type" : "account",
```

```
        "_id" : "227",
        "_score" : 0.0,
        "_source" : {
            "account_number" : 227,
            "balance" : 19780,
            "firstname" : "Coleman",
            "lastname" : "Berg",
            "age" : 22,
            "gender" : "M",
            "address" : "776 Little Street",
            "employer" : "Exoteric",
            "email" : "colemanberg@exoteric.com",
            "city" : "Eagleville",
            "state" : "WV"
        }
    },
{
    "_index" : "bank",
    "_type" : "account",
    "_id" : "272",
    "_score" : 0.0,
    "_source" : {
        "account_number" : 272,
        "balance" : 19253,
        "firstname" : "Lilly",
        "lastname" : "Morgan",
        "age" : 25,
        "gender" : "F",
        "address" : "689 Fleet Street",
        "employer" : "Biolive",
        "email" : "lillymorgan@biolive.com",
        "city" : "Sunbury",
        "state" : "OH"
    }
}
]
}
```

能看到所有文档的 `_score` : 0.0。

(8) term

和match一样。匹配某个属性的值。全文检索字段用match，其他非text字段匹配用term。

Avoid using the `term` query for `text` fields.

避免对文本字段使用“term”查询

By default, Elasticsearch changes the values of `text` fields as part of analysis. This can make finding exact matches for `text` field values difficult.

默认情况下，Elasticsearch作为analysis的一部分更改' text '字段的值。这使得为“text”字段值寻找精确匹配变得困难。

To search `text` field values, use the match.

要搜索“text”字段值，请使用匹配。

<https://www.elastic.co/guide/en/elasticsearch/reference/7.6/query-dsl-term-query.html>

使用term匹配查询

```
GET bank/_search
{
  "query": {
    "term": {
      "address": "mill Road"
    }
  }
}
```

查询结果：

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
```

```

    "failed" : 0
},
"hits" : {
  "total" : {
    "value" : 0,
    "relation" : "eq"
  },
  "max_score" : null,
  "hits" : [ ]
}
}

```

一条也没有匹配到

而更换为match匹配时，能够匹配到32个文档

The screenshot shows a search interface with the following query:

```

GET bank/_search
{
  "query": {
    "match": {
      "address": "mill Road"
    }
  }
}

```

The results pane shows the response with the following details:

```

1  {
2   "took" : 6,
3   "timed_out" : false,
4   "shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 32,
13      "relation" : "eq"
14    },
15    "max_score" : 8.926605,
16    "hits" : [
17      {
18        "_index": "bank",
19        "_score": 8.926605,
20        "_type": "document",
21        "_id": "1",
22        "_source": {
23          "name": "John Doe",
24          "address": "mill Road"
25        }
26      }
27    ]
28  }
29}

```

也就是说，全文检索字段用**match**，其他非text字段匹配用**term**。

(9) Aggregation (执行聚合)

聚合提供了从数据中分组和提取数据的能力。最简单的聚合方法大致等于SQL Group by和SQL聚合函数。在elasticsearch中，执行搜索返回this（命中结果），并且同时返回聚合结果，把以响应中的所有hits（命中结果）分隔开的能力。这是非常强大且有效的，你可以执行查询和多个聚合，并且在一次使用中得到各自的（任何一个的）返回结果，使用一次简洁和简化的API啦避免网络往返。

"size":0

size:0不显示搜索数据

aggs：执行聚合。聚合语法如下：

```
"aggs":{  
    "aggs_name这次聚合的名字，方便展示在结果集中":{  
        "AGG_TYPE聚合的类型(avg,term,terms)":{}  
    }  
},
```

搜索address中包含mill的所有人的年龄分布以及平均年龄，但不显示这些人的详情

```
GET bank/_search  
{  
    "query": {  
        "match": {  
            "address": "Mill"  
        }  
    },  
    "aggs": {  
        "ageAgg": {  
            "terms": {  
                "field": "age",  
                "size": 10  
            }  
        },  
        "ageAvg": {  
            "avg": {  
                "field": "age"  
            }  
        },  
        "balanceAvg": {  
            "avg": {  
                "field": "balance"  
            }  
        }  
    },  
    "size": 0  
}
```

查询结果：

```
{  
    "took" : 2,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 1,  
        "successful" : 1,  
        "skipped" : 0,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : {  
            "value" : 4,  
            "relation" : "eq"  
        },  
        "max_score" : null,  
        "hits" : [ ]  
    },  
    "aggregations" : {  
        "ageAgg" : {  
            "doc_count_error_upper_bound" : 0,  
            "sum_other_doc_count" : 0,  
            "buckets" : [  
                {  
                    "key" : 38,  
                    "doc_count" : 2  
                },  
                {  
                    "key" : 28,  
                    "doc_count" : 1  
                },  
                {  
                    "key" : 32,  
                    "doc_count" : 1  
                }  
            ]  
        },  
        "ageAvg" : {  
            "value" : 34.0  
        },  
        "balanceAvg" : {  
            "value" : 25208.0  
        }  
    }  
}
```

```
    }
}
}
```

复杂：

按照年龄聚合，并且求这些年龄段的这些人的平均薪资

```
GET bank/_search
{
  "query": {
    "match_all": {}
  },
  "aggs": {
    "ageAgg": {
      "terms": {
        "field": "age",
        "size": 100
      },
      "aggs": {
        "ageAvg": {
          "avg": {
            "field": "balance"
          }
        }
      }
    }
  },
  "size": 0
}
```

输出结果：

```
{
  "took" : 49,
  "timed_out" : false,
  "_shards" : {
```

```
"total" : 1,
"successful" : 1,
"skipped" : 0,
"failed" : 0
},
"hits" : {
  "total" : {
    "value" : 1000,
    "relation" : "eq"
  },
  "max_score" : null,
  "hits" : [ ]
},
"aggregations" : {
  "ageAgg" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : 31,
        "doc_count" : 61,
        "ageAvg" : {
          "value" : 28312.918032786885
        }
      },
      {
        "key" : 39,
        "doc_count" : 60,
        "ageAvg" : {
          "value" : 25269.583333333332
        }
      },
      {
        "key" : 26,
        "doc_count" : 59,
        "ageAvg" : {
          "value" : 23194.813559322032
        }
      },
      {
        "key" : 32,
        "doc_count" : 52,
        "ageAvg" : {
          "value" : 23951.346153846152
        }
      }
    ],
    "min_doc_count" : 52
  }
}
```

```
{
    "key" : 35,
    "doc_count" : 52,
    "ageAvg" : {
        "value" : 22136.69230769231
    }
},
{
    "key" : 36,
    "doc_count" : 52,
    "ageAvg" : {
        "value" : 22174.71153846154
    }
},
{
    "key" : 22,
    "doc_count" : 51,
    "ageAvg" : {
        "value" : 24731.07843137255
    }
},
{
    "key" : 28,
    "doc_count" : 51,
    "ageAvg" : {
        "value" : 28273.882352941175
    }
},
{
    "key" : 33,
    "doc_count" : 50,
    "ageAvg" : {
        "value" : 25093.94
    }
},
{
    "key" : 34,
    "doc_count" : 49,
    "ageAvg" : {
        "value" : 26809.95918367347
    }
},
{
    "key" : 30,
    "doc_count" : 47,
    "ageAvg" : {
```

```
        "value" : 22841.106382978724
    }
},
{
    "key" : 21,
    "doc_count" : 46,
    "ageAvg" : {
        "value" : 26981.434782608696
    }
},
{
    "key" : 40,
    "doc_count" : 45,
    "ageAvg" : {
        "value" : 27183.17777777778
    }
},
{
    "key" : 20,
    "doc_count" : 44,
    "ageAvg" : {
        "value" : 27741.227272727272
    }
},
{
    "key" : 23,
    "doc_count" : 42,
    "ageAvg" : {
        "value" : 27314.214285714286
    }
},
{
    "key" : 24,
    "doc_count" : 42,
    "ageAvg" : {
        "value" : 28519.04761904762
    }
},
{
    "key" : 25,
    "doc_count" : 42,
    "ageAvg" : {
        "value" : 27445.214285714286
    }
},
{
```

```

    "key" : 37,
    "doc_count" : 42,
    "ageAvg" : {
        "value" : 27022.261904761905
    }
},
{
    "key" : 27,
    "doc_count" : 39,
    "ageAvg" : {
        "value" : 21471.871794871793
    }
},
{
    "key" : 38,
    "doc_count" : 39,
    "ageAvg" : {
        "value" : 26187.17948717949
    }
},
{
    "key" : 29,
    "doc_count" : 35,
    "ageAvg" : {
        "value" : 29483.14285714286
    }
}
]
}
}

```

查出所有年龄段，并且这些年龄段中M的平均薪资和F的平均薪资以及这个年龄段的总体平均薪资



```

GET bank/_search
{
    "query": {
        "match_all": {}
    },
    "aggs": {
        "ageAgg": {
            "terms": {
                "field": "age",

```

```
        "size": 100
    },
    "aggs": {
        "genderAgg": {
            "terms": {
                "field": "gender.keyword"
            },
            "aggs": {
                "balanceAvg": {
                    "avg": {
                        "field": "balance"
                    }
                }
            }
        },
        "ageBalanceAvg": {
            "avg": {
                "field": "balance"
            }
        }
    }
},
"size": 0
}
```

输出结果：

```
{
  "took" : 119,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
```

```
"hits" : [ ]
},
"aggregations" : {
  "ageAgg" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : 31,
        "doc_count" : 61,
        "genderAgg" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [
            {
              "key" : "M",
              "doc_count" : 35,
              "balanceAvg" : {
                "value" : 29565.628571428573
              }
            },
            {
              "key" : "F",
              "doc_count" : 26,
              "balanceAvg" : {
                "value" : 26626.576923076922
              }
            }
          ]
        },
        "ageBalanceAvg" : {
          "value" : 28312.918032786885
        }
      }
    ],
    .......//省略其他
  }
}
```

3) Mapping

(1) 字段类型

核心类型

字符串 (string)

text, keyword

数字类型 (Numeric)

long, integer, short, byte, double, float, half_float, scaled_float

日期类型 (Date)

date

布尔类型 (Boolean)

boolean

二进制类型 (binary)

binary

复合类型

数组类型 (Array)

Array 支持不针对特定的类型

对象类型 (Object)

object 用于单 JSON 对象

嵌套类型 (Nested)

nested 用于 JSON 对象数组

· 地理类型 (Geo)

地理坐标 (Geo-points)

geo_point 用于描述 经纬度坐标

地理图形 (Geo-Shape)

geo_shape 用于描述复杂形状，如多边形

(2) 映射

Mapping(映射)

Maping是用来定义一个文档 (document)，以及它所包含的属性 (field) 是如何存储和索引的。比如：使用maping 来定义：

- 哪些字符串属性应该被看做全文本属性 (full text fields)；
- 哪些属性包含数字，日期或地理位置；
- 文档中的所有属性是否都被索引 (all 配置)；
- 日期的格式；
- 自定义映射规则来执行动态添加属性；
- 查看mapping信息
GET bank/_mapping



```
{  
  "bank" : {  
    "mappings" : {
```

```
"properties" : {
    "account_number" : {
        "type" : "long"
    },
    "address" : {
        "type" : "text",
        "fields" : {
            "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
            }
        }
    },
    "age" : {
        "type" : "long"
    },
    "balance" : {
        "type" : "long"
    },
    "city" : {
        "type" : "text",
        "fields" : {
            "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
            }
        }
    },
    "email" : {
        "type" : "text",
        "fields" : {
            "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
            }
        }
    },
    "employer" : {
        "type" : "text",
        "fields" : {
            "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
            }
        }
    },
},
```

```
"firstname" : {
    "type" : "text",
    "fields" : {
        "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
        }
    }
},
"gender" : {
    "type" : "text",
    "fields" : {
        "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
        }
    }
},
"lastname" : {
    "type" : "text",
    "fields" : {
        "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
        }
    }
},
"state" : {
    "type" : "text",
    "fields" : {
        "keyword" : {
            "type" : "keyword",
            "ignore_above" : 256
        }
    }
}
}
```

- 修改mapping信息

自动猜测的映射类型

JSON type	域 type
布尔型: true 或者 false	boolean
整数: 123	long
浮点数: 123.45	double
字符串, 有效日期: 2014-09-15	date
字符串: foo bar	string

(3) 新版本改变

ElasticSearch7-去掉type概念

- 关系型数据库中两个数据表示是独立的，即使他们里面有相同名称的列也不影响使用，但ES中不是这样的。
elasticsearch是基于Lucene开发的搜索引擎，而ES中不同type下名称相同的filed最终在Lucene中的处理方式是一样的。
 - 两个不同type下的两个user_name，在ES同一个索引下其实被认为是同一个filed，你必须在两个不同的type中定义相同的filed映射。否则，不同type中的相同字段名称就会在处理中出现冲突的情况，导致Lucene处理效率下降。
 - 去掉type就是为了提高ES处理数据的效率。
- Elasticsearch 7.x URL中的type参数为可选。比如，索引一个文档不再要求提供文档类型。
- Elasticsearch 8.x 不再支持URL中的type参数。
- 解决：
 - 将索引从多类型迁移到单类型，每种类型文档一个独立索引
 - 将已存在的索引下的类型数据，全部迁移到指定位置即可。详见数据迁移

Elasticsearch 7.x

- Specifying types in requests is deprecated. For instance, indexing a document no longer requires a document `type`. The new index APIs are `PUT {index}/_doc/{id}` in case of explicit ids and `POST {index}/_doc` for auto-generated ids. Note that in 7.0, `_doc` is a permanent part of the path, and represents the endpoint name rather than the document type.
- The `include_type_name` parameter in the index creation, index template, and mapping APIs will default to `false`. Setting the parameter at all will result in a deprecation warning.
- The `_default_` mapping type is removed.

Elasticsearch 8.x

- Specifying types in requests is no longer supported.
- The `include_type_name` parameter is removed.

创建映射

创建索引并指定映射

```
PUT /my_index
{
  "mappings": {
    "properties": {
      "age": {
        "type": "integer"
      },
      "email": {
        "type": "keyword"
      },
      "name": {
        "type": "text"
      }
    }
  }
}
```

输出:

```
● ○ ●

{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "my_index"
}
```

查看映射

```
● ○ ●

GET /my_index
```

输出结果:

```
● ○ ●

{
  "my_index" : {
    "aliases" : { },
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "integer"
        },
        "email" : {
          "type" : "keyword"
        },
        "employee-id" : {
          "type" : "keyword",
          "index" : false
        },
        "name" : {
          "type" : "text"
        }
      }
    }
  }
}
```

```
},
"settings" : {
  "index" : {
    "creation_date" : "1588410780774",
    "number_of_shards" : "1",
    "number_of_replicas" : "1",
    "uuid" : "ua0lxhtkQCOMn7Kh3iUu0w",
    "version" : {
      "created" : "7060299"
    },
    "provided_name" : "my_index"
  }
}
}
```

添加新的字段映射

```
PUT /my_index/_mapping
{
  "properties": {
    "employee-id": {
      "type": "keyword",
      "index": false
    }
  }
}
```

这里的 "index": false, 表明新增的字段不能被检索，只是一个冗余字段。

更新映射

对于已经存在的字段映射，我们不能更新。更新必须创建新的索引，进行数据迁移。

数据迁移

先创建new_twitter的正确映射。然后使用如下方式进行数据迁移。

```
POST reindex [固定写法]
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitters"
  }
}
```

将旧索引的type下的数据进行迁移

```
POST reindex [固定写法]
{
  "source": {
    "index": "twitter",
    "type": "twitter"
  },
  "dest": {
    "index": "new_twitters"
  }
}
```

更多详情见：<https://www.elastic.co/guide/en/elasticsearch/reference/7.6/docs-reindex.html>

GET /bank/_search

```
{  
    "took" : 0,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 1,  
        "successful" : 1,  
        "skipped" : 0,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : {  
            "value" : 1000,  
            "relation" : "eq"  
        },  
        "max_score" : 1.0,  
        "hits" : [  
            {  
                "_index" : "bank",  
                "_type" : "account", //类型为account  
                "_id" : "1",  
                "_score" : 1.0,  
                "_source" : {  
                    "account_number" : 1,  
                    "balance" : 39225,  
                    "firstname" : "Amber",  
                    "lastname" : "Duke",  
                    "age" : 32,  
                    "gender" : "M",  
                    "address" : "880 Holmes Lane",  
                    "employer" : "Pyrami",  
                    "email" : "amberduke@pyrami.com",  
                    "city" : "Brogan",  
                    "state" : "IL"  
                }  
            },  
            ...  
        ]  
    }  
}
```



GET /bank/_search

```
{  
  "bank" : {  
    "mappings" : {  
      "properties" : {  
        "account_number" : {  
          "type" : "long"  
        },  
        "address" : {  
          "type" : "text",  
          "fields" : {  
            "keyword" : {  
              "type" : "keyword",  
              "ignore_above" : 256  
            }  
          }  
        },  
        "age" : {  
          "type" : "long"  
        },  
        "balance" : {  
          "type" : "long"  
        },  
        "city" : {  
          "type" : "keyword"  
        },  
        "email" : {  
          "type" : "keyword"  
        }  
      }  
    }  
  }  
}
```



想要将年龄修改为integer



```
PUT /newbank  
{  
  "mappings": {  
    "properties": {  
      "account_number": {  
        "type": "long"  
      },  
      "address": {  
        "type": "text"  
      },  
      "age": {  
        "type": "integer"  
      },  
      "balance": {  
        "type": "long"  
      },  
      "city": {  
        "type": "keyword"  
      },  
      "email": {  
        "type": "keyword"  
      }  
    }  
  }  
}
```

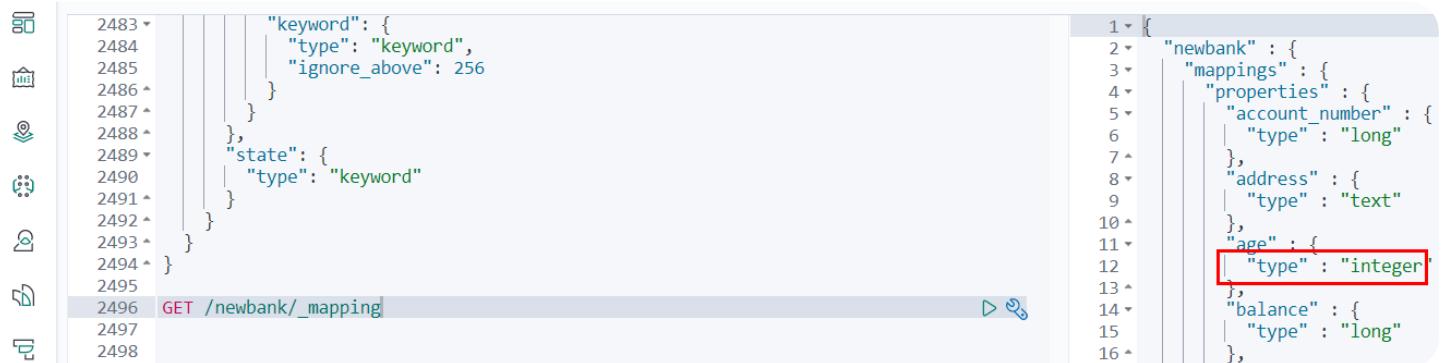
```

},
"employer": {
  "type": "keyword"
},
"firstname": {
  "type": "text"
},
"gender": {
  "type": "keyword"
},
"lastname": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"state": {
  "type": "keyword"
}
}
}
}

```

查看“newbank”的映射：

GET /newbank/_mapping



```

2483 "newbank" : {
2484   "mappings" : {
2485     "properties" : {
2486       "account_number" : {
2487         "type": "long"
2488       }
2489     },
2490     "state" : {
2491       "type": "keyword"
2492     }
2493   }
2494 }
2495
2496 GET /newbank/_mapping
2497
2498

```

```

1 {
2   "newbank" : {
3     "mappings" : {
4       "properties" : {
5         "account_number" : {
6           "type": "long"
7         },
8         "address" : {
9           "type": "text"
10        },
11        "age" : {
12          "type": "integer" // This line is highlighted in red
13        },
14        "balance" : {
15           "type": "long"
16         }
17       }
18     }
19   }
20 }

```

能够看到age的映射类型被修改为了integer.

将bank中的数据迁移到newbank中

```
POST _reindex
{
  "source": {
    "index": "bank",
    "type": "account"
  },
  "dest": {
    "index": "newbank"
  }
}
```

运行输出：

```
#! Deprecation: [types removal] Specifying types in reindex requests is deprecated.
{
  "took" : 768,
  "timed_out" : false,
  "total" : 1000,
  "updated" : 0,
  "created" : 1000,
  "deleted" : 0,
  "batches" : 1,
  "version_conflicts" : 0,
  "noops" : 0,
  "retries" : {
    "bulk" : 0,
    "search" : 0
  },
  "throttled_millis" : 0,
  "requests_per_second" : -1.0,
  "throttled_until_millis" : 0,
  "failures" : [ ]
}
```

查看newbank中的数据

```

GET /newbank/_search
{
  "took": 1049,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1000,
      "relation": "eq"
    },
    "max_score": 1.0,
    "hits": [
      {
        "_index": "newbank",
        "_type": "doc",
        "_id": "1",
        "_score": 1.0
      }
    ]
  }
}

```

4) 分词

一个tokenizer（分词器）接收一个字符流，将之分割为独立的tokens（词元，通常是独立的单词），然后输出tokens流。

例如：whitespace tokenizer遇到空白字符时分割文本。它会将文本“Quick brown fox!”分割为[Quick,brown,fox!]。

该tokenizer（分词器）还负责记录各个terms(词条)的顺序或position位置（用于phrase短语和word proximity词近邻查询），以及term（词条）所代表的原始word（单词）的start（起始）和end（结束）的character offsets（字符串偏移量）（用于高亮显示搜索的内容）。

elasticsearch提供了很多内置的分词器，可以用来构建custom analyzers（自定义分词器）。

关于分词器：<https://www.elastic.co/guide/en/elasticsearch/reference/7.6/analysis.html>

```

POST _analyze
{
  "analyzer": "standard",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}

```

执行结果：

```

{
  "tokens": [
    ...
  ]
}

```

```
{
    "token" : "the",
    "start_offset" : 0,
    "end_offset" : 3,
    "type" : "<ALPHANUM>",
    "position" : 0
},
{
    "token" : "2",
    "start_offset" : 4,
    "end_offset" : 5,
    "type" : "<NUM>",
    "position" : 1
},
{
    "token" : "quick",
    "start_offset" : 6,
    "end_offset" : 11,
    "type" : "<ALPHANUM>",
    "position" : 2
},
{
    "token" : "brown",
    "start_offset" : 12,
    "end_offset" : 17,
    "type" : "<ALPHANUM>",
    "position" : 3
},
{
    "token" : "foxes",
    "start_offset" : 18,
    "end_offset" : 23,
    "type" : "<ALPHANUM>",
    "position" : 4
},
{
    "token" : "jumped",
    "start_offset" : 24,
    "end_offset" : 30,
    "type" : "<ALPHANUM>",
    "position" : 5
},
{
    "token" : "over",
    "start_offset" : 31,
    "end_offset" : 35,
```

```
        "type" : "<ALPHANUM>",
        "position" : 6
    },
    {
        "token" : "the",
        "start_offset" : 36,
        "end_offset" : 39,
        "type" : "<ALPHANUM>",
        "position" : 7
    },
    {
        "token" : "lazy",
        "start_offset" : 40,
        "end_offset" : 44,
        "type" : "<ALPHANUM>",
        "position" : 8
    },
    {
        "token" : "dog's",
        "start_offset" : 45,
        "end_offset" : 50,
        "type" : "<ALPHANUM>",
        "position" : 9
    },
    {
        "token" : "bone",
        "start_offset" : 51,
        "end_offset" : 55,
        "type" : "<ALPHANUM>",
        "position" : 10
    }
]
}
```

(1) 安装ik分词器

+ Configure text analysis

- Built-in analyzer reference

« Dynamic templates

Text analysis overview »

Fingerprint Analyzer

Keyword Analyzer

Language Analyzers

Pattern Analyzer

Simple Analyzer

Standard Analyzer

Stop Analyzer

Whitespace Analyzer

+ Tokenizer reference

所有的语言分词， 默认使用的都是“Standard Analyzer”， 但是这些分词器针对于中文的分词，并不友好。为此需要安装中文的分词器。

注意：不能用默认elasticsearch-plugin install xxx.zip 进行自动安装

<https://github.com/medcl/elasticsearch-analysis-ik/releases/download> 对应es版本安装

在前面安装的elasticsearch时， 我们已经将elasticsearch容器的“/usr/share/elasticsearch/plugins”目录， 映射到宿主机的“/mydata/elasticsearch/plugins”目录下， 所以比较方便的做法就是下载“elasticsearch-analysis-ik-7.6.2.zip”文件， 然后解压到该文件夹下即可。安装完毕后， 需要重启elasticsearch容器。

如果不嫌麻烦， 还可以采用如下的方式。

(1) 查看elasticsearch版本号：

```
[root@hadoop-104 ~]# curl http://localhost:9200
{
  "name" : "0adeb7852e00",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "9gglpP0HTfyOTRAaSe2rIg",
  "version" : {
    "number" : "7.6.2",          #版本号为7.6.2
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef48eb35cf30adf4db14086e8aab07ef6fb113f",
    "build_date" : "2020-03-26T06:34:37.794943Z",
    "build_snapshot" : false,
    "lucene_version" : "8.4.0",
  }
}
```

```
"minimum_wire_compatibility_version" : "6.8.0",
"minimum_index_compatibility_version" : "6.0.0-beta1"
},
"tagline" : "You Know, for Search"
}
[root@hadoop-104 ~]#
```

(2) 进入es容器内部plugin目录

- docker exec -it 容器id /bin/bash

● ● ●

```
[root@hadoop-104 ~]# docker exec -it elasticsearch /bin/bash
[root@0adeb7852e00 elasticsearch]#
```

- wget <https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v7.6.2/elasticsearch-analysis-ik-7.6.2.zip>

● ● ●

```
[root@0adeb7852e00 elasticsearch]# pwd
/usr/share/elasticsearch
#下载ik7.6.2
[root@0adeb7852e00 elasticsearch]# wget https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v7.6.2/elasticsearch-analysis-ik-7.6.2.zip
```

- unzip 下载的文件

● ● ●

```
[root@0adeb7852e00 elasticsearch]# unzip elasticsearch-analysis-ik-7.6.2.zip -d ikn
Archive:  elasticsearch-analysis-ik-7.6.2.zip
  creating: ik/config/
  inflating: ik/config/main.dic
  inflating: ik/config/quantifier.dic
  inflating: ik/config/extra_single_word_full.dic
  inflating: ik/config/IKAnalyzer.cfg.xml
  inflating: ik/config/surname.dic
  inflating: ik/config/suffix.dic
  inflating: ik/config/stopword.dic
```

```
inflating: ik/config/extra_main.dic
inflating: ik/config/extra_stopword.dic
inflating: ik/config/preposition.dic
inflating: ik/config/extra_single_word_low_freq.dic
inflating: ik/config/extra_single_word.dic
inflating: ik/elasticsearch-analysis-ik-7.6.2.jar
inflating: ik/httpclient-4.5.2.jar
inflating: ik/httpcore-4.4.4.jar
inflating: ik/commons-logging-1.2.jar
inflating: ik/commons-codec-1.9.jar
inflating: ik/plugin-descriptor.properties
inflating: ik/plugin-security.policy
[root@0adeb7852e00 elasticsearch]#
#移动到plugins目录下
[root@0adeb7852e00 elasticsearch]# mv ik plugins/
```

- rm -rf *.zip



```
[root@0adeb7852e00 elasticsearch]# rm -rf elasticsearch-analysis-ik-7.6.2.zip
```

确认是否安装好了分词器

(2) 测试分词器

使用默认



```
GET my_index/_analyze
{
  "text": "我是中国人"
}
```

请观察执行结果：



```
{
  "tokens" : [
    {
```

```
        "token" : "我",
        "start_offset" : 0,
        "end_offset" : 1,
        "type" : "<IDEOGRAPHIC>",
        "position" : 0
    },
    {
        "token" : "是",
        "start_offset" : 1,
        "end_offset" : 2,
        "type" : "<IDEOGRAPHIC>",
        "position" : 1
    },
    {
        "token" : "中",
        "start_offset" : 2,
        "end_offset" : 3,
        "type" : "<IDEOGRAPHIC>",
        "position" : 2
    },
    {
        "token" : "国",
        "start_offset" : 3,
        "end_offset" : 4,
        "type" : "<IDEOGRAPHIC>",
        "position" : 3
    },
    {
        "token" : "人",
        "start_offset" : 4,
        "end_offset" : 5,
        "type" : "<IDEOGRAPHIC>",
        "position" : 4
    }
]
```

```
GET my_index/_analyze
{
  "analyzer": "ik_smart",
  "text": "我是中国人"
}
```

输出结果：

```
{
  "tokens": [
    {
      "token": "我",
      "start_offset": 0,
      "end_offset": 1,
      "type": "CN_CHAR",
      "position": 0
    },
    {
      "token": "是",
      "start_offset": 1,
      "end_offset": 2,
      "type": "CN_CHAR",
      "position": 1
    },
    {
      "token": "中国人",
      "start_offset": 2,
      "end_offset": 5,
      "type": "CN_WORD",
      "position": 2
    }
  ]
}
```

```
GET my_index/_analyze
{
  "analyzer": "ik_max_word",
  "text": "我是中国人"
}
```

输出结果：

```
{
  "tokens" : [
    {
      "token" : "我",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "是",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "中国人",
      "start_offset" : 2,
      "end_offset" : 5,
      "type" : "CN_WORD",
      "position" : 2
    },
    {
      "token" : "中国",
      "start_offset" : 2,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 3
    }
  ]
}
```

```
        "token" : "国人",
        "start_offset" : 3,
        "end_offset" : 5,
        "type" : "CN_WORD",
        "position" : 4
    }
]
}
```

(3) 自定义词库

- 修改/usr/share/elasticsearch/plugins/ik/config中的IKAnalyzer.cfg.xml
/usr/share/elasticsearch/plugins/ik/config



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict"></entry>
    <!--用户可以在这里配置自己的扩展停止词字典-->
    <entry key="ext_stopwords"></entry>
    <!--用户可以在这里配置远程扩展字典 -->
    <entry key="remote_ext_dict">http://192.168.137.14/es/fenci.txt</entry>
    <!--用户可以在这里配置远程扩展停止词字典-->
    <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

原来的xml



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict"></entry>
```

```
<!--用户可以在这里配置自己的扩展停止词字典-->
<entry key="ext_stopwords"></entry>
<!--用户可以在这里配置远程扩展字典 -->
<!-- <entry key="remote_ext_dict">words_location</entry> -->
<!--用户可以在这里配置远程扩展停止词字典-->
<!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

修改完成后，需要重启elasticsearch容器，否则修改不生效。

更新完成后，es只会对于新增的数据用更新分词。历史数据是不会重新分词的。如果想要历史数据重新分词，需要执行：

```
POST my_index/_update_by_query?conflicts=proceed
```

<http://192.168.137.14/es/fenci.txt>, 这个是nginx上资源的访问路径

在运行下面实例之前，需要安装nginx（安装方法见安装nginx），然后创建“fenci.txt”文件，内容如下：

```
echo "樱桃萨其马，带你甜蜜入夏" > /mydata/nginx/html/fenci.txt
```

测试效果：

```
GET my_index/_analyze
{
  "analyzer": "ik_max_word",
  "text": "樱桃萨其马，带你甜蜜入夏"
}
```

输出结果：

```
{  
  "tokens" : [  
    {  
      "token" : "櫻桃",  
      "start_offset" : 0,  
      "end_offset" : 2,  
      "type" : "CN_WORD",  
      "position" : 0  
    },  
    {  
      "token" : "萨其马",  
      "start_offset" : 2,  
      "end_offset" : 5,  
      "type" : "CN_WORD",  
      "position" : 1  
    },  
    {  
      "token" : "带你",  
      "start_offset" : 6,  
      "end_offset" : 8,  
      "type" : "CN_WORD",  
      "position" : 2  
    },  
    {  
      "token" : "甜蜜",  
      "start_offset" : 8,  
      "end_offset" : 10,  
      "type" : "CN_WORD",  
      "position" : 3  
    },  
    {  
      "token" : "入夏",  
      "start_offset" : 10,  
      "end_offset" : 12,  
      "type" : "CN_WORD",  
      "position" : 4  
    }  
  ]  
}
```

4、elasticsearch-Rest-Client

1) 9300: TCP

- spring-data-elasticsearch:transport-api.jar;
 - springboot版本不同， transport-api.jar不同， 不能适配es版本
 - 7.x已经不建议使用， 8以后就要废弃

2) 9200: HTTP

- jestClient: 非官方， 更新慢；
- RestTemplate: 模拟HTTP请求， ES很多操作需要自己封装， 麻烦；
- HttpClient: 同上；
- Elasticsearch-Rest-Client: 官方RestClient， 封装了ES操作， API层次分明， 上手简单；
最终选择Elasticsearch-Rest-Client (elasticsearch-rest-high-level-client) ；
<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high.html>

5、附录：安装Nginx

- 随便启动一个nginx实例， 只是为了复制出配置



```
docker run -p80:80 --name nginx -d nginx:1.10
```

- 将容器内的配置文件拷贝到/mydata/nginx/conf/ 下

```
mkdir -p /mydata/nginx/html  
mkdir -p /mydata/nginx/logs  
mkdir -p /mydata/nginx/conf  
docker container cp nginx:/etc/nginx/* /mydata/nginx/conf/  
#由于拷贝完成后会在config中存在一个nginx文件夹，所以需要将它的内容移动到conf中  
mv /mydata/nginx/conf/nginx/* /mydata/nginx/conf/  
rm -rf /mydata/nginx/conf/nginx
```

- 终止原容器：

```
docker stop nginx
```

- 执行命令删除原容器：

```
docker rm nginx
```

- 创建新的Nginx，执行以下命令

```
docker run -p 80:80 --name nginx \  
-v /mydata/nginx/html:/usr/share/nginx/html \  
-v /mydata/nginx/logs:/var/log/nginx \  
-v /mydata/nginx/conf:/etc/nginx \  
-d nginx:1.10
```

- 设置开机启动nginx

```
docker update nginx --restart=always
```

- 创建“/mydata/nginx/html/index.html”文件，测试是否能够正常访问



```
echo '<h2>hello nginx!</h2>' >index.html
```

访问: <http://nginx所在主机的IP:80/index.html>

SpringBoot整合ElasticSearch

1、导入依赖

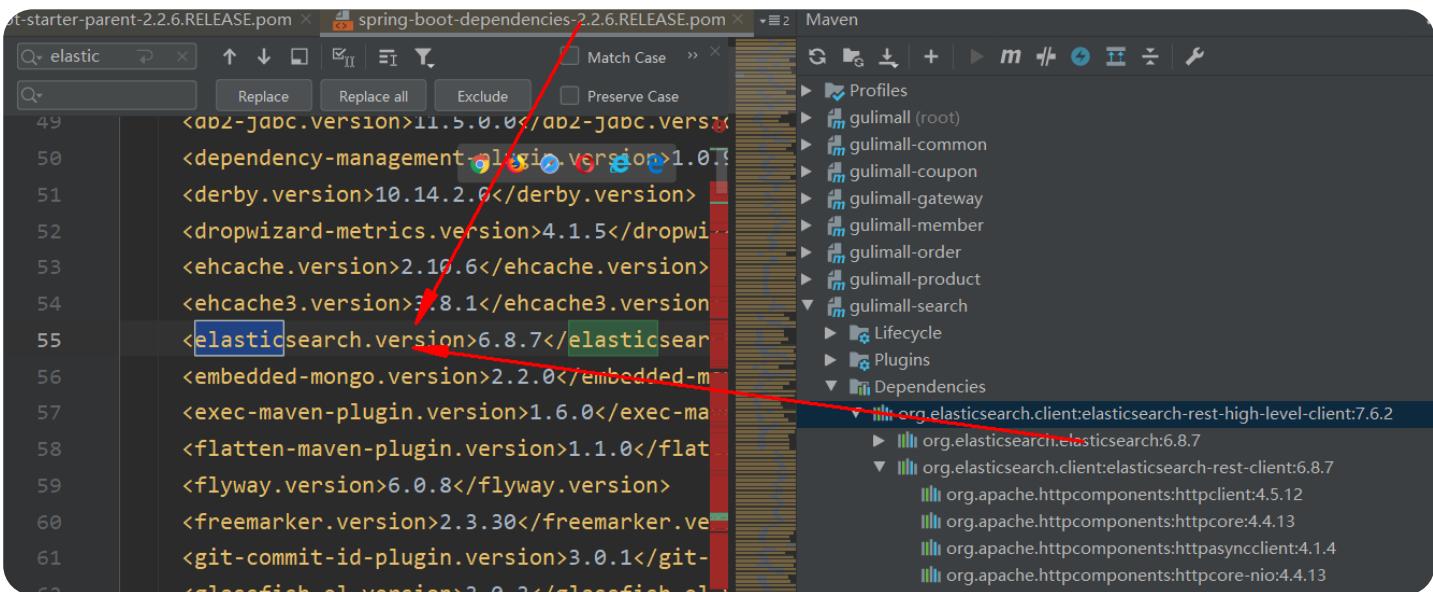
这里的版本要和所按照的ELK版本匹配。



```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>7.6.2</version>
</dependency>
```



```
<elasticsearch.version>6.8.7</elasticsearch.version>
```



需要在项目中将它改为7.6.2

```
<properties>
    ...
    <elasticsearch.version>7.6.2</elasticsearch.version>
</properties>
```

2、编写测试类

1) 测试保存数据

<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high-document-index.html>

```
@Test
public void indexData() throws IOException {
    IndexRequest indexRequest = new IndexRequest("users");
}
```

```

User user = new User();
user.setUserName("张三");
user.setAge(20);
user.setGender("男");
String jsonString = JSON.toJSONString(user);
//设置要保存的内容
indexRequest.source(jsonString, XContentType.JSON);
//执行创建索引和保存数据
IndexResponse index = client.index(indexRequest,
GulimallElasticSearchConfig.COMMON_OPTIONS);

System.out.println(index);

}

```

测试前：

The screenshot shows the Elasticsearch Dev Tools interface. On the left, there is a code editor with several requests listed:

```

1 GET my_index/_search
2
3 GET my_index/_analyze
4 {
5   "analyzer": "ik_max_word",
6   "text": "樱桃萨其马，带你甜蜜入夏"
7 }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

```

A red box highlights the line "GET users/_search". A red arrow points from this line to the right pane, which displays the response for the "users/_search" request. The response is as follows:

```

1 {
2   "error": {
3     "root_cause": [
4       {
5         "type": "index_not_found_exception",
6         "reason": "no such index [users]",
7         "resource.type": "index_or_alias",
8         "resource.id": "users",
9         "index_uuid": "na_",
10        "index": "users"
11      }
12    ],
13    "type": "index_not_found_exception",
14    "reason": "no such index [users]",
15    "resource.type": "index_or_alias",
16    "resource.id": "users",
17    "index_uuid": "na_",
18    "index": "users"
19  },
20  "status": 404
21
22

```

测试后：

```
GET my_index/_search  
  
GET my_index/_analyze  
{  
    "analyzer": "ik_max_word",  
    "text": "樱桃萨其马，带你甜蜜入夏"  
}
```

```
GET users/_search
```

```
1: {  
2:     "took" : 73,  
3:     "timed_out" : false,  
4:     "_shards" : {  
5:         "total" : 1,  
6:         "successful" : 1,  
7:         "skipped" : 0,  
8:         "failed" : 0  
9:     },  
10:    "hits" : {  
11:        "total" : {  
12:            "value" : 1,  
13:            "relation" : "eq"  
14:        },  
15:        "max_score" : 1.0,  
16:        "hits" : [  
17:            {  
18:                "_index" : "users",  
19:                "_type" : "_doc",  
20:                "_id" : "-2vAHIB0nzmLJLkxKwK",  
21:                "_score" : 1.0,  
22:                "_source" : {  
23:                    "age" : 20,  
24:                    "gender" : "男",  
25:                    "userName" : "张三"  
26:                }  
27:            }  
28:        ]  
29:    }  
30: }
```

2) 测试获取数据

<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high-search.html>

```
@Test  
public void searchData() throws IOException {  
    GetRequest getRequest = new GetRequest(  
        "users",  
        "-2vAHIB0nzmLJLkxKwK");  
  
    GetResponse getResponse = client.get(getRequest, RequestOptions.DEFAULT);  
    System.out.println(getResponse);  
    String index = getResponse.getIndex();  
    System.out.println(index);  
    String id = getResponse.getId();  
    System.out.println(id);  
    if (getResponse.exists()) {  
        long version = getResponse.getVersion();  
        System.out.println(version);  
        String sourceAsString = getResponse.getSourceAsString();  
        System.out.println(sourceAsString);  
        Map<String, Object> sourceAsMap = getResponse.getSourceAsMap();  
        System.out.println(sourceAsMap);  
        byte[] sourceAsBytes = getResponse.getSourceAsBytes();  
    } else {  
    }  
}
```

```
    }  
}
```

查询state="AK"的文档：



```
{  
  "took": 1,  
  "timed_out": false,  
  "_shards": {  
    "total": 1,  
    "successful": 1,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": {  
      "value": 22,    //匹配到了22条  
      "relation": "eq"  
    },  
    "max_score": 3.7952394,  
    "hits": [{  
      "_index": "bank",  
      "_type": "account",  
      "_id": "210",  
      "_score": 3.7952394,  
      "_source": {  
        "account_number": 210,  
        "balance": 33946,  
        "firstname": "Cherry",  
        "lastname": "Carey",  
        "age": 24,  
        "state": "AK"  
      }  
    }]  
}
```

```
        "gender": "M",
        "address": "539 Tiffany Place",
        "employer": "Martgo",
        "email": "cherrycarey@martgo.com",
        "city": "Fairacres",
        "state": "AK"
    }
},
....//省略其他
]
}
}
```

搜索address中包含mill的所有人的年龄分布以及平均年龄，平均薪资

```
GET bank/_search
{
  "query": {
    "match": {
      "address": "Mill"
    }
  },
  "aggs": {
    "ageAgg": {
      "terms": {
        "field": "age",
        "size": 10
      }
    },
    "ageAvg": {
      "avg": {
        "field": "age"
      }
    },
    "balanceAvg": {
      "avg": {
        "field": "balance"
      }
    }
  }
}
```

java实现

```
/*
 * 复杂检索:在bank中搜索address中包含mill的所有人的年龄分布以及平均年龄，平均薪资
 * @throws IOException
 */
@Test
public void searchData() throws IOException {
    //1. 创建检索请求
    SearchRequest searchRequest = new SearchRequest();

    //1.1) 指定索引
    searchRequest.indices("bank");
    //1.2) 构造检索条件
    SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
    sourceBuilder.query(QueryBuilders.matchQuery("address", "Mill"));

    //1.2.1)按照年龄分布进行聚合
    TermsAggregationBuilder ageAgg=AggregationBuilders.terms("ageAgg").field("age").size(10);
    sourceBuilder.aggregation(ageAgg);

    //1.2.2)计算平均年龄
    AvgAggregationBuilder ageAvg = AggregationBuilders.avg("ageAvg").field("age");
    sourceBuilder.aggregation(ageAvg);
    //1.2.3)计算平均薪资
    AvgAggregationBuilder balanceAvg =
    AggregationBuilders.avg("balanceAvg").field("balance");
    sourceBuilder.aggregation(balanceAvg);

    System.out.println("检索条件: "+sourceBuilder);
    searchRequest.source(sourceBuilder);
    //2. 执行检索
    SearchResponse searchResponse = client.search(searchRequest,
    RequestOptions.DEFAULT);
    System.out.println("检索结果: "+searchResponse);

    //3. 将检索结果封装为Bean
    SearchHits hits = searchResponse.getHits();
    SearchHit[] searchHits = hits.getHits();
    for (SearchHit searchHit : searchHits) {
```

```

        String sourceAsString = searchHit.getSourceAsString();
        Account account = JSON.parseObject(sourceAsString, Account.class);
        System.out.println(account);

    }

//4. 获取聚合信息
Aggregations aggregations = searchResponse.getAggregations();

Terms ageAgg1 = aggregations.get("ageAgg");

for (Terms.Bucket bucket : ageAgg1.getBuckets()) {
    String keyAsString = bucket.getKeyAsString();
    System.out.println("年龄: " +keyAsString+ " ==> " +bucket.getDocCount());
}

Avg ageAvg1 = aggregations.get("ageAvg");
System.out.println("平均年龄: " +ageAvg1.getValue());

Avg balanceAvg1 = aggregations.get("balanceAvg");
System.out.println("平均薪资: " +balanceAvg1.getValue());


}

```

可以尝试对比打印的条件和执行结果，和前面的ElasticSearch的检索语句和检索结果进行比较；

其他

1. kibana控制台命令

ctrl+home: 回到文档首部；

ctrl+end: 回到文档尾部。

1、K8s快速入门

- 1) 简介
- 2) 架构
 - (1) 整体主从方式
 - (2) master节点架构
 - (3) Node节点架构
- 3) 概念
- 4) 快速体验
 - (1) 安装minikube
 - (2) 体验nginx部署升级

2、K8s集群安装

- 1) kubeadm
- 2) 前置要求
- 3) 部署步骤
- 4) 环境准备
 - (1) 准备工作
 - (2) 启动三个虚拟机
 - (3) 设置Linux环境（三个节点都执行）
- 5) 所有节点安装docker、kubeadm、kubelet、kubectl
 - (1) 安装Docker
 - (2) 添加阿里与Yum源
 - (3) 安装kubeadm, kubelet和kubectl
- 6) 部署k8s-master
 - (1) master节点初始化
 - (2) 测试Kubectl（主节点执行）
- 7) 安装POD网络插件（CNI）
- 8) 加入kubernetes的Node节点
- 9) 入门操作kubernetes集群

3、K8s细节

- 1、kubectl文档
- 2、资源类型
- 3、格式化输出
- 命令参考
- service的意义
- Ingress

安装kubernetes可视化界面——DashBoard

kubesphere

- 1、简洁
- 2、安装前提提交
 - 1、安装helm（master节点执行）
 - 2、安装Tiller（Master执行）

1、K8s快速入门

1) 简介

kubernetes简称k8s。是用于自动部署，扩展和管理容器化应用程序的开源系统。

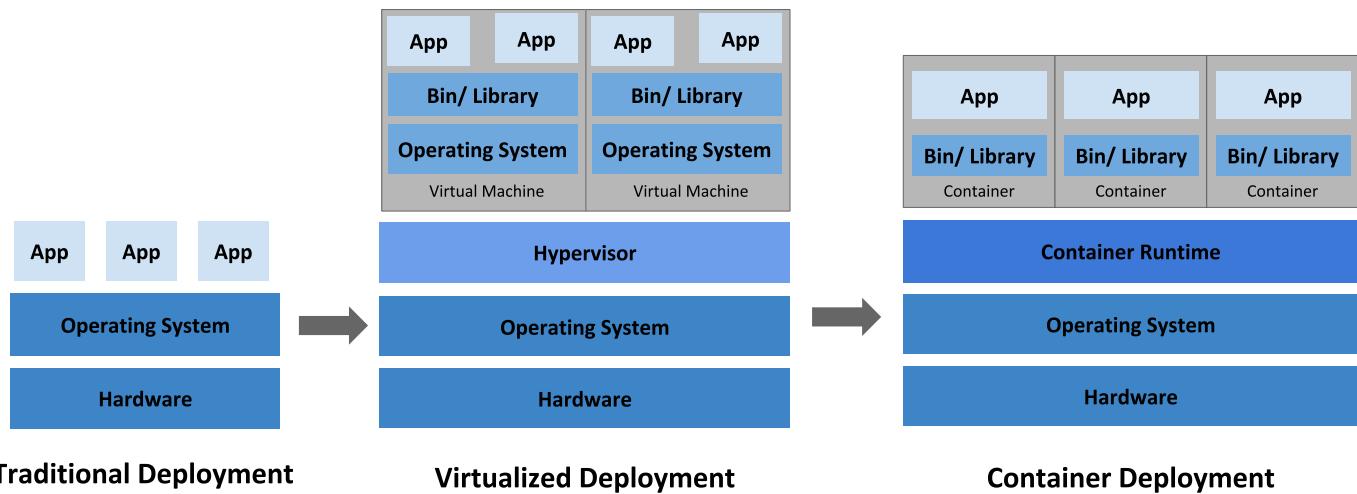
中文官网: <https://kubernetes.io/Zh/>

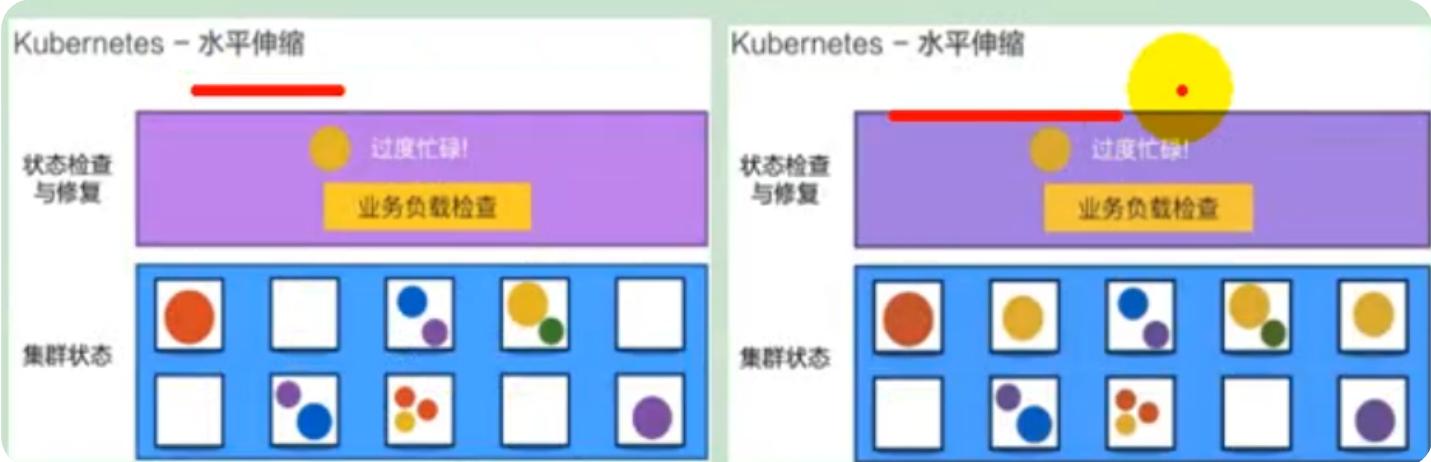
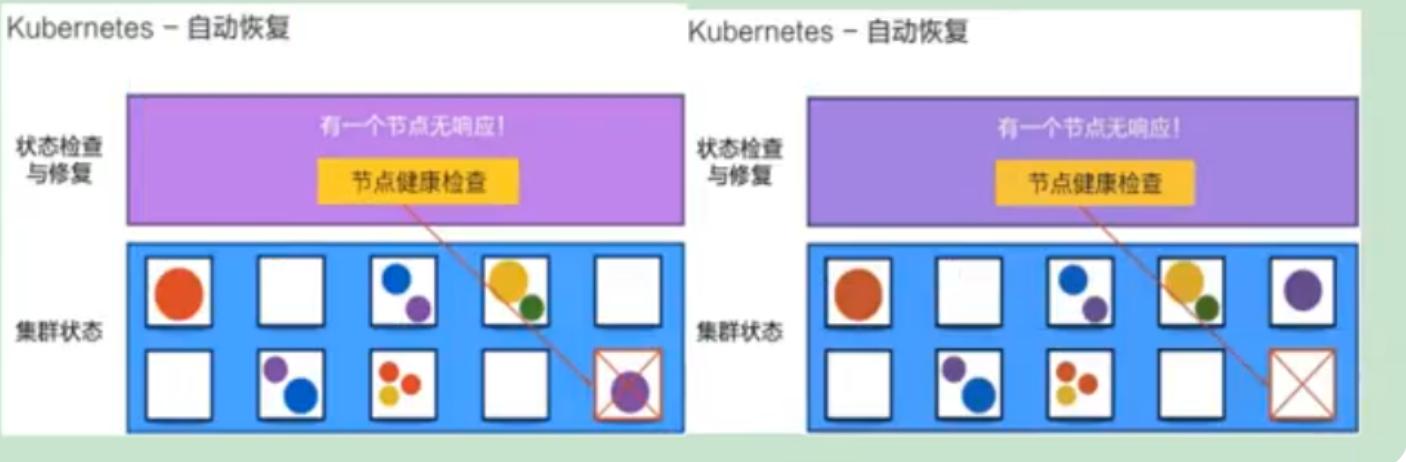
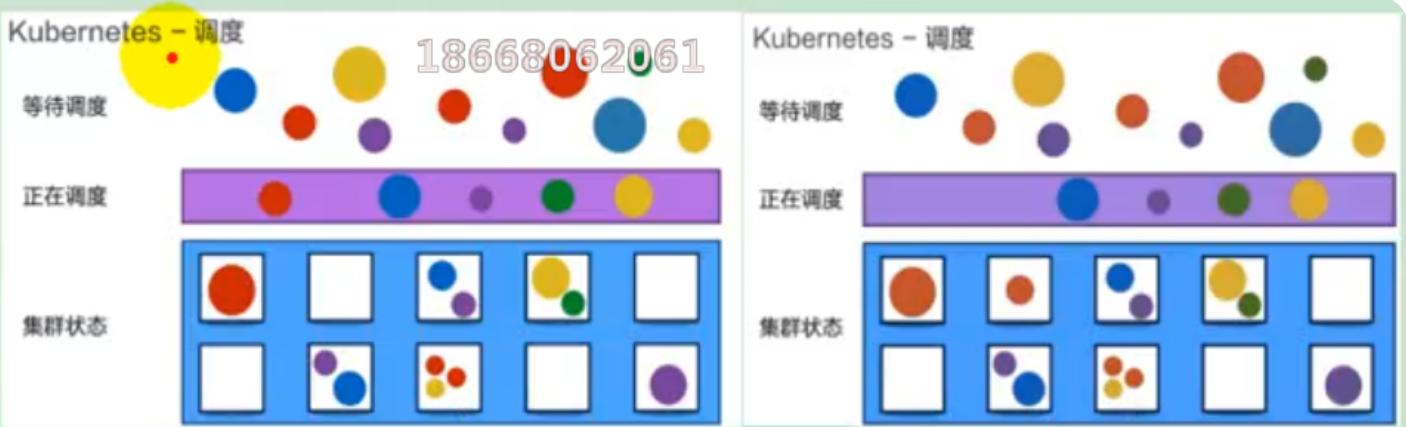
中文社区: <https://www.kubernetes.org.cn/>

官方文档: <https://kubernetes.io/zh/docs/home/>

社区文档: <https://docs.kubernetes.org.cn/>

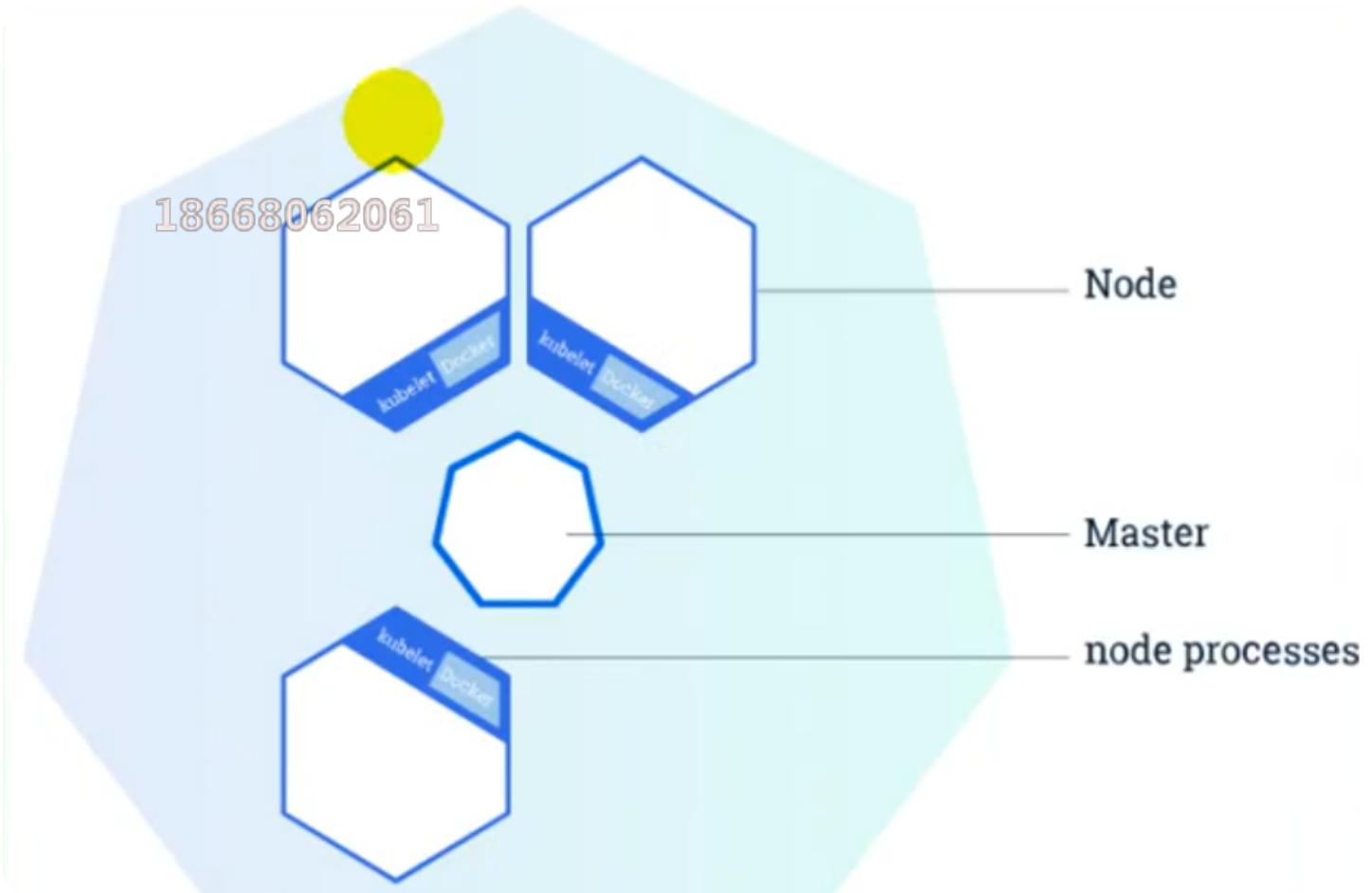
部署方式的进化:



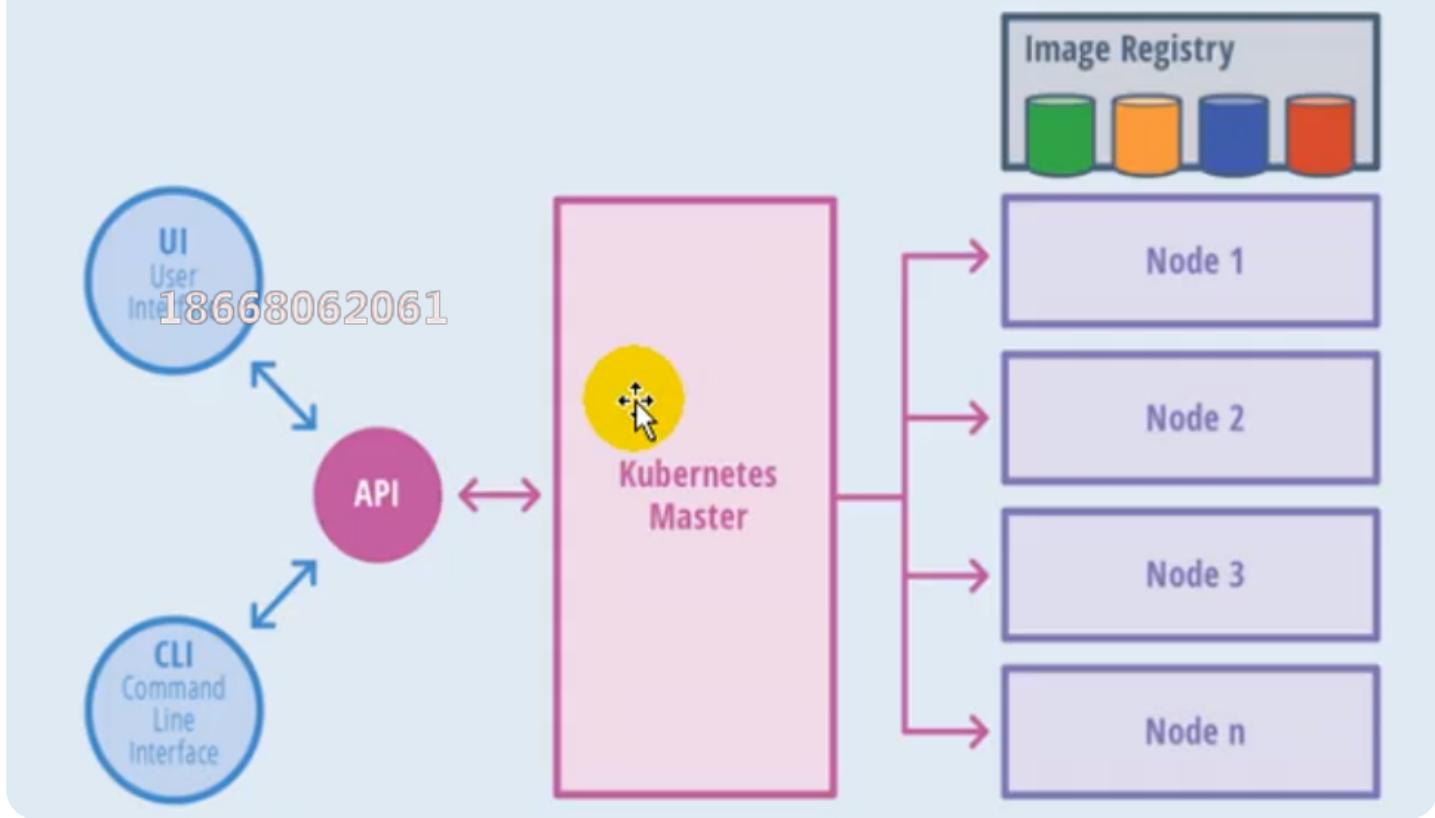


2) 架构

(1) 整体主从方式



Kubernetes Architecture

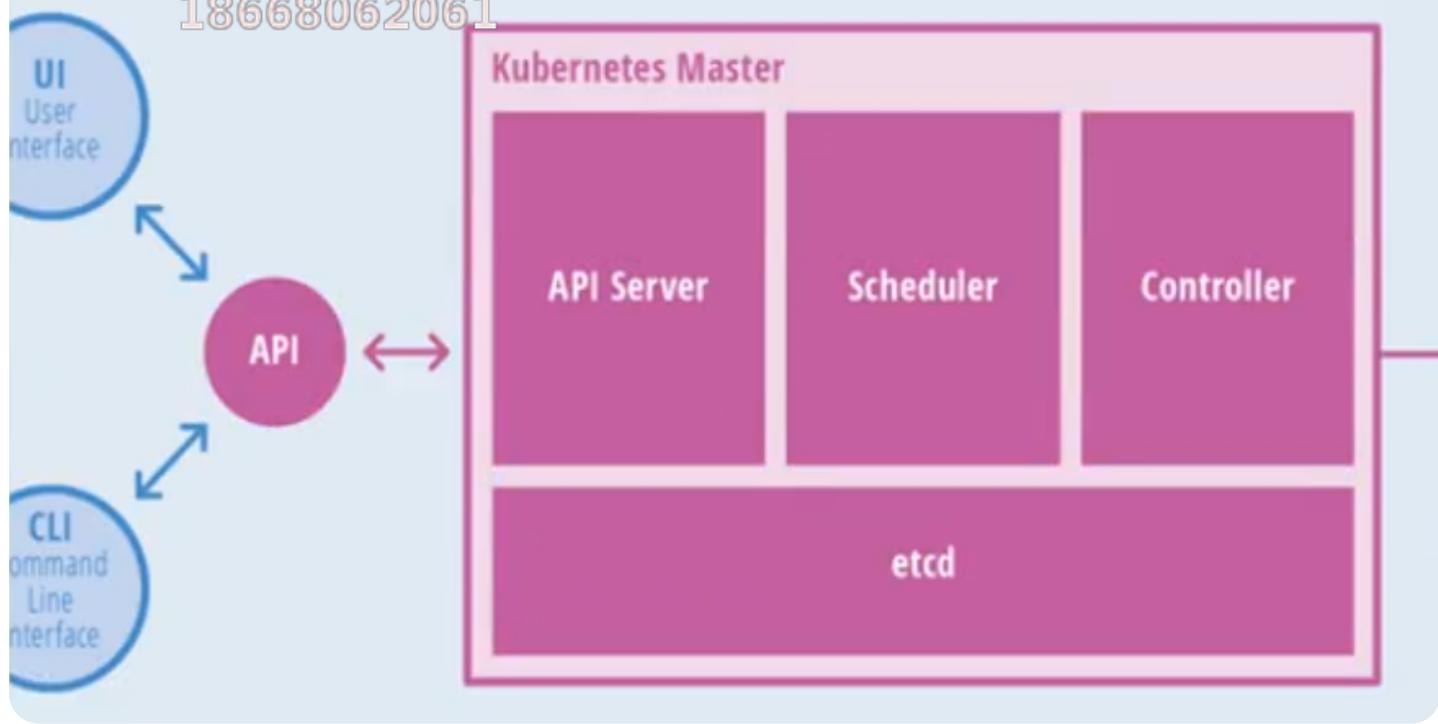


(2) master节点架构

Kubernetes Master



18668062061



- kube-apiserver

- 对外暴露 K8S 的 api 接口，是外界进行资源操作的唯一入口
- 提供认证、授权、访问控制、API 注册和发现等机制

- etcd

68062061 etcd 是兼具一致性和高可用性的键值数据库，可以作为保存 Kubernetes 所有集群数据的后台数据库。

- Kubernetes 集群的 etcd 数据库通常需要有个备份计划

- kube-scheduler

- 主节点上的组件，该组件监视那些新创建的未指定运行节点的 Pod，并选择节点让 Pod 在上面运行。
- 所有对 k8s 的集群操作，都必须经过主节点进行调度

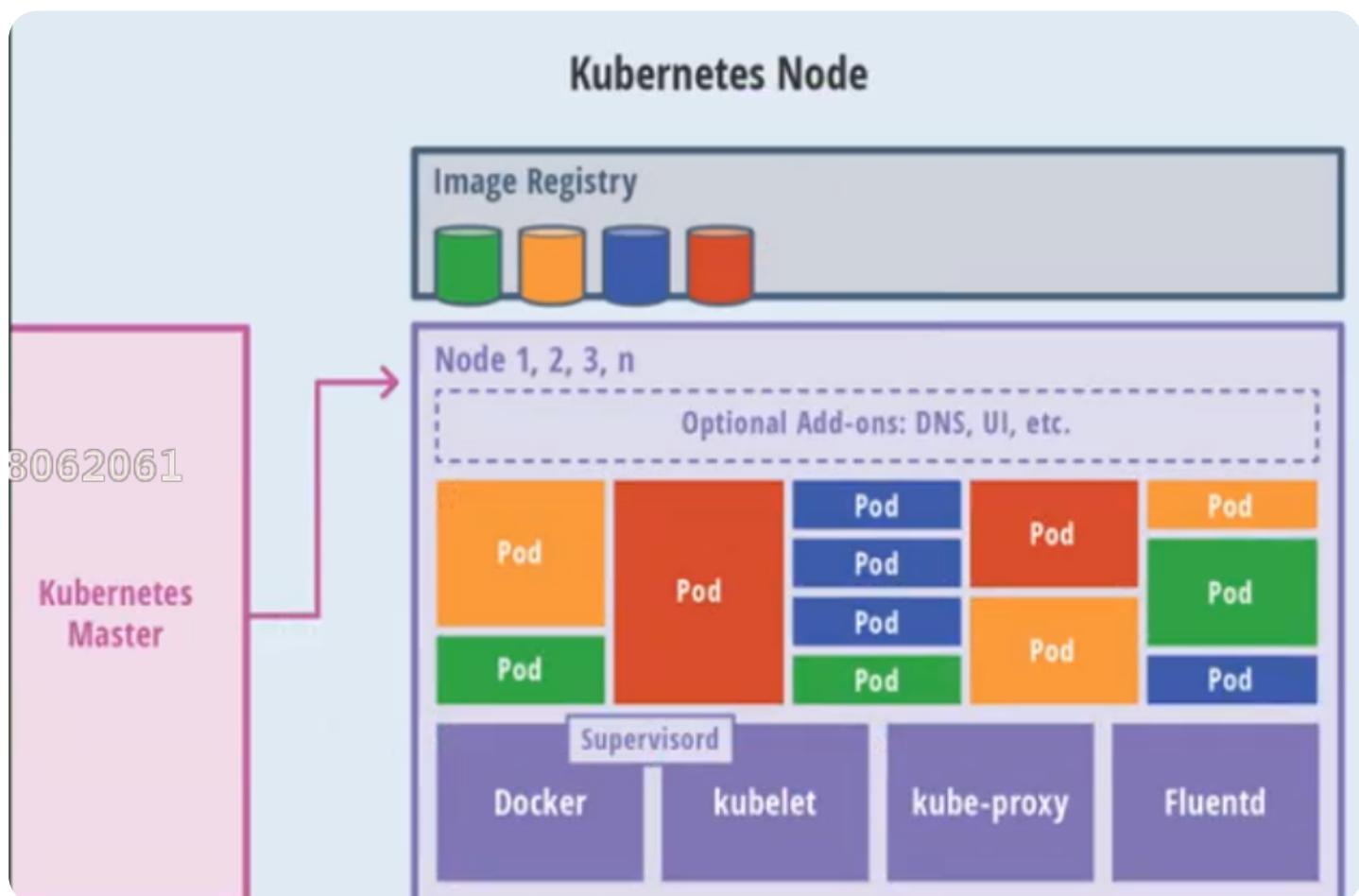
- kube-controller-manager

- 在主节点上运行控制器的组件
- 这些控制器包括：

这些控制器包括：

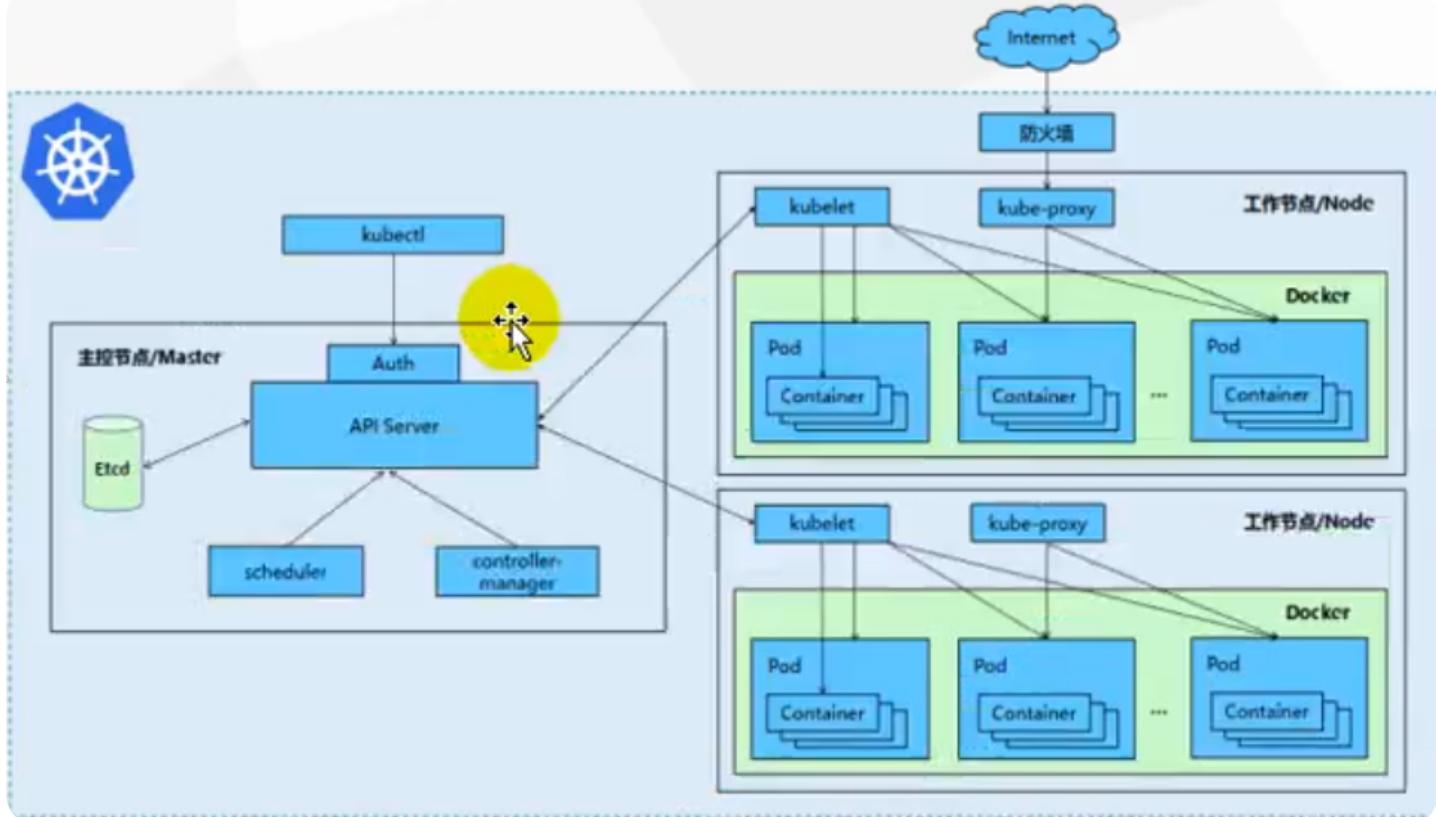
- ◆ 节点控制器（Node Controller）：负责在节点出现故障时进行通知和响应。
- ◆ 副本控制器（Replication Controller）：负责为系统中的每个副本控制器对象维护正确数量的 Pod。
- ◆ 端点控制器（Endpoints Controller）：填充端点（Endpoints）对象（即加入 Service 与 Pod）。
- ◆ 服务帐户和令牌控制器（Service Account & Token Controllers）：为新的命名空间创建默认帐户和 API 访问令牌

(3) Node 节点架构

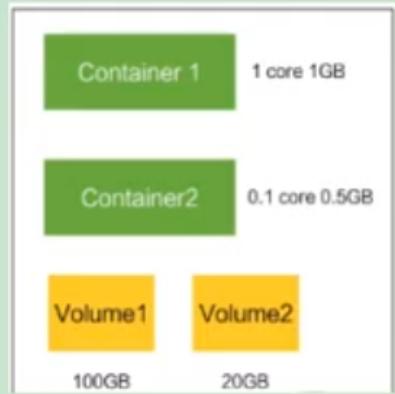


- kubelet
 - 一个在集群中每个节点上运行的代理。它保证容器都运行在 Pod 中。
 - 负责维护容器的生命周期，同时也负责 Volume (CSI) 和网络 (CNI) 的管理；
- kube-proxy
 - 负责为 Service 提供 cluster 内部的服务发现和负载均衡；
- 容器运行环境(Container Runtime)
 - 容器运行环境是负责运行容器的软件。
 - Kubernetes 支持多个容器运行环境: Docker、containerd、cri-o、rktlet 以及任何实现 Kubernetes CRI (容器运行环境接口)。
- fluentd
 - 是一个守护进程，它有助于提供集群层面日志 集群层面的日志

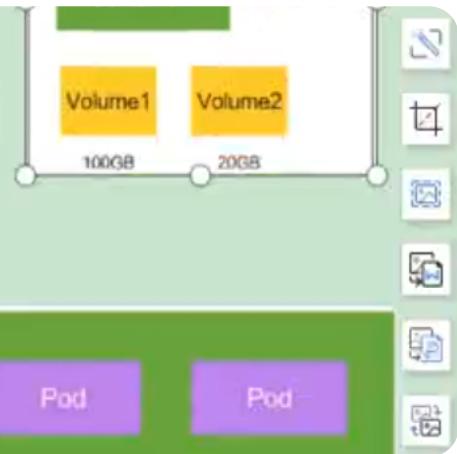
3) 概念



- Container: 容器, 可以是 docker 启动的一个容器
- Pod:
 - k8s 使用 Pod 来组织一组容器
 - 一个 Pod 中的所有容器共享同一网络。
 - Pod 是 k8s 中的最小部署单元
- Volume
 - 声明在 Pod 容器中可访问的文件目录
 - 可以被挂载在 Pod 中一个或多个容器指定路径下
 - 支持多种后端存储抽象(本地存储, 分布式存储, 云存储…)



- Controllers: 更高层次对象，部署和管理 Pod;
 - ReplicaSet: 确保预期的 Pod 副本数量
 - Deployment: 无状态应用部署
 - StatefulSet: 有状态应用部署
 - DaemonSet: 确保所有 Node 都运行一个指定 Pod
 - Job: 一次性任务
 - Cronjob: 定时任务
- Deployment:



- Deployment:
 - 定义一组 Pod 的副本数目、版本等
 - 通过控制器 (Controller) 维持 Pod 数目(自动回
复失败的 Pod)
 - 通过控制器以指定的策略控制版本 (滚动升级, 回滚等)
- Service
 - 定义一组 Pod 的访问策略
 - Pod 的负载均衡, 提供一个或者多个 Pod 的稳定
访问地址

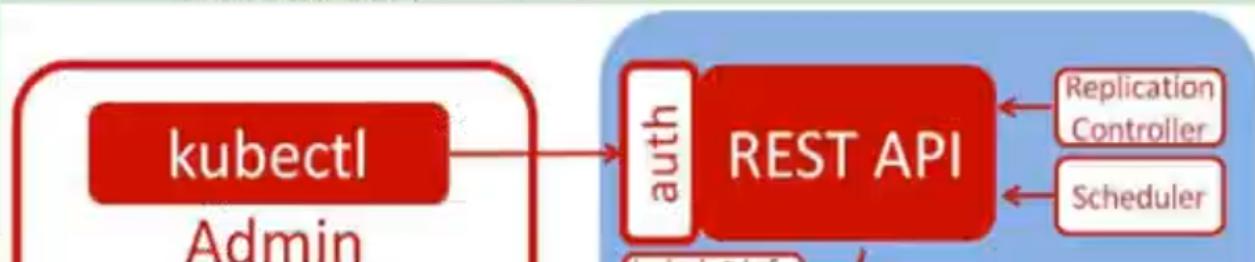


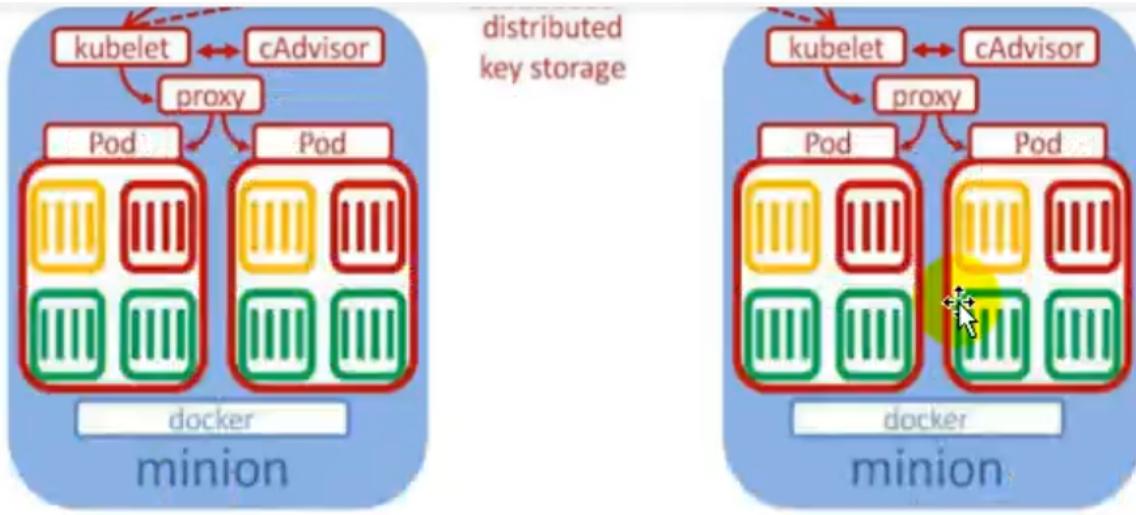
- 支持多种方式 (ClusterIP、NodePort、LoadBalance)
- Label: 标签, 用于对象资源的查询, 筛选

```
apiVersion: v1
kind: Pod
metadata:
  name: apple
  labels:
    color: red
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: banana
  labels:
    color: yellow
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

- Namespace: 命名空间, 逻辑隔离
 - 一个集群内部的逻辑隔离机制 (鉴权, 资源)
 - 每个资源都属于一个 namespace
 - 同一个 namespace 所有资源名不能重复
 - 不同 namespace 可以资源名重复





API:

我们通过 kubernetes 的 API 来操作整个集群。

可以通过 kubectl、ui、curl 最终发送 http+json/yaml 方式的请求给 API Server，然后控制 k8s 集群。k8s 里的所有的资源对象都可以采用 yaml 或 JSON 格式的文件定义或描述

apiVersion

Kind

Metadata

Spec

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

I

4) 快速体验

(1) 安装minikube

<https://github.com/kubernetes/minikube/releases>

下载minikuber-windows-amd64.exe 改名为minikube.exe

打开virtualBox, 打开cmd

运行

minikube start --vm-driver=virtualbox --registry-mirror=<https://registry.docker-cn.com>

等待20分钟即可。

(2) 体验nginx部署升级

1. 提交一个nginx deployment

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml
```

2. 升级 nginx deployment

```
kubectl apply -f https://k8s.io/examples/application/deployment-update.yaml
```

3. 扩容 nginx deployment

2、K8s集群安装

1) kubeadm

kubeadm是官方社区推出的一个用于快速部署kubernetes集群的工具。

这个工具能通过两条指令完成一个kubernetes集群的部署

创建一个master节点



```
$ kubernetes init
```

将一个node节点加入到当前集群中



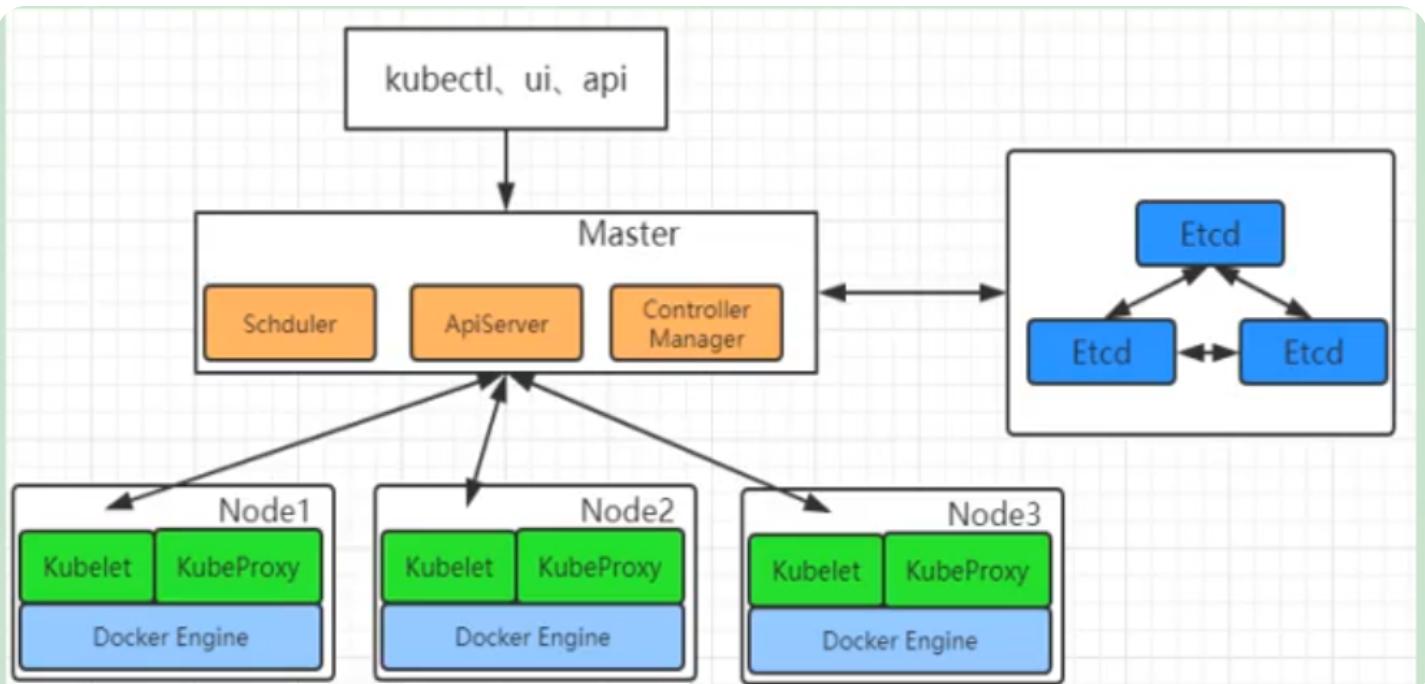
```
$ kubeadm join <Master节点的IP和端口>
```

2) 前置要求

一台或多台机器，操作系统Centos7.x-86_x64
硬件配置：2GB或更多RAM，2个CPU或更多CPU，硬盘30GB或更多
集群中所有的机器之间网络互通
可以访问外网，需要拉取镜像
禁止Swap分区

3) 部署步骤

1. 在所有的节点上安装Docker和kubeadm
2. 不是Kubernetes Master
3. 部署容器网络插件
4. 部署Kubernetes Node，将节点加入Kubernetes集群中
5. 部署DashBoard web页面，可视化查看Kubernetes资源



4) 环境准备

(1) 准备工作

- 我们可以使用vagrant快速创建三个虚拟机。虚拟机启动前先设置virtualbox的主机网络。现在全部统一为192.168.56.1，以后所有虚拟机都是56.x的ip地址。



- 在全局设定中，找到一个空间比较大的磁盘用用来存放镜像。



(2) 启动三个虚拟机

- 使用我们提供的vagrant文件，复制到非中文无空格目录下，运行vagrant up启动三个虚拟机。其实vagrant完全可以一键部署全部K8s集群

<https://github.com/rootsongjc/kubernetes-vagrant-centos-cluster>

<http://github.com/davidkbainbridge/k8s-playground>

下面是vagrantfile，使用它来创建三个虚拟机，分别为k8s-node1，k8s-node2和k8s-node3.



```
Vagrant.configure("2") do |config|
  (1..3).each do |i|
    config.vm.define "k8s-node#{i}" do |node|
      # 设置虚拟机的Box
      node.vm.box = "centos/7"

      # 设置虚拟机的主机名
      node.vm.hostname="k8s-node#{i}"

      # 设置虚拟机的IP
      node.vm.network "private_network", ip: "192.168.56.{99+i}", netmask:
      "255.255.255.0"

      # 设置主机与虚拟机的共享目录
      # node.vm.synced_folder "~/Documents/vagrant/share", "/home/vagrant/share"

      # VirtualBox相关配置
      node.vm.provider "virtualbox" do |v|
        # 设置虚拟机的名称
        v.name = "k8s-node#{i}"
        # 设置虚拟机的内存大小
        v.memory = 4096
        # 设置虚拟机的CPU个数
        v.cpus = 4
      end
    end
  end
end
```

- 进入到三个虚拟机，开启root的密码访问权限



Vagrant ssh xxx进入到系统后

```
su root 密码为vagrant
```

```
vi /etc/ssh/sshd_config
```

修改

```
PermitRootLogin yes  
PasswordAuthentication yes
```

所有的虚拟机设为4核4G

关于在"网络地址转换"的连接方式下，三个节点的eth0，IP地址相同的问题。

问题描述：查看k8s-node1的路由表：



```
[root@k8s-node1 ~]# ip route show  
default via 10.0.2.2 dev eth0 proto dhcp metric 100  
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15 metric 100  
192.168.56.0/24 dev eth1 proto kernel scope link src 192.168.56.100 metric 101  
[root@k8s-node1 ~]
```

能够看到路由表中记录的是，通过端口eth0进行数据包的收发。

分别查看k8s-node1，k8s-node2和k8s-node3的eth0所绑定的IP地址，发现它们都是相同的，全都是10.0.2.15，这些地址是供kubernetes集群通信用的，区别于eth1上的IP地址，是通远程管理使用的。



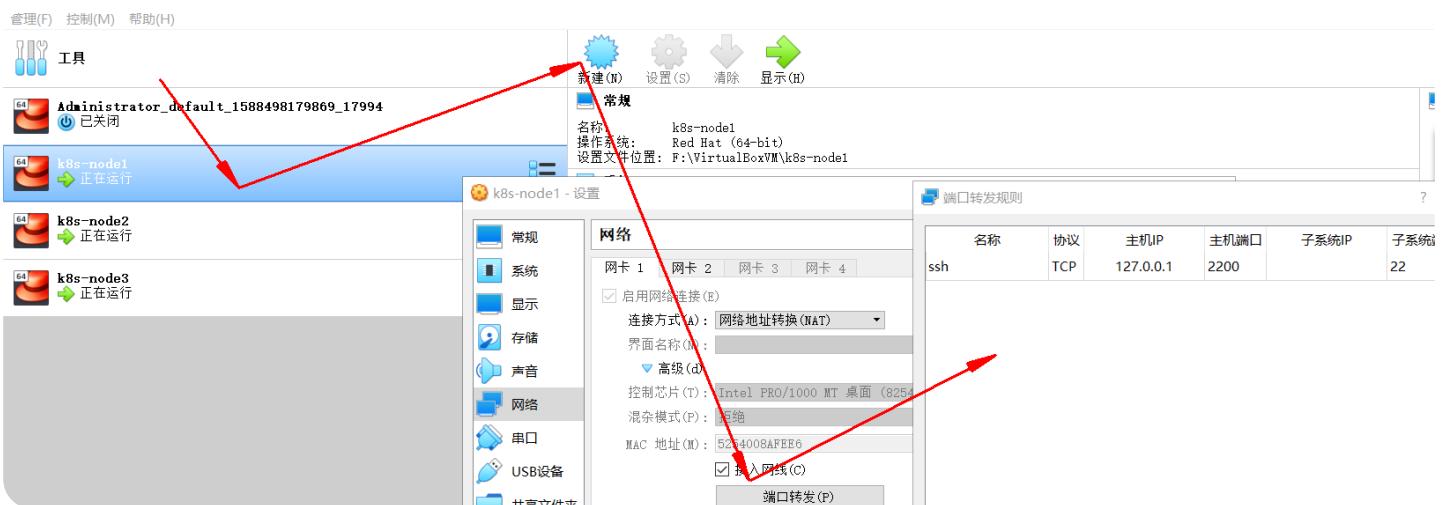
```
[root@k8s-node1 ~]# ip addr  
...  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group  
default qlen 1000  
    link/ether 52:54:00:8a:fe:e6 brd ff:ff:ff:ff:ff:ff  
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
```

```

    valid_lft 84418sec preferred_lft 84418sec
    inet6 fe80::5054:ff:fe8a:fee6/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 08:00:27:a3:ca:c0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.100/24 brd 192.168.56.255 scope global noprefixroute eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fea3:cac0/64 scope link
        valid_lft forever preferred_lft forever
[root@k8s-node1 ~]#

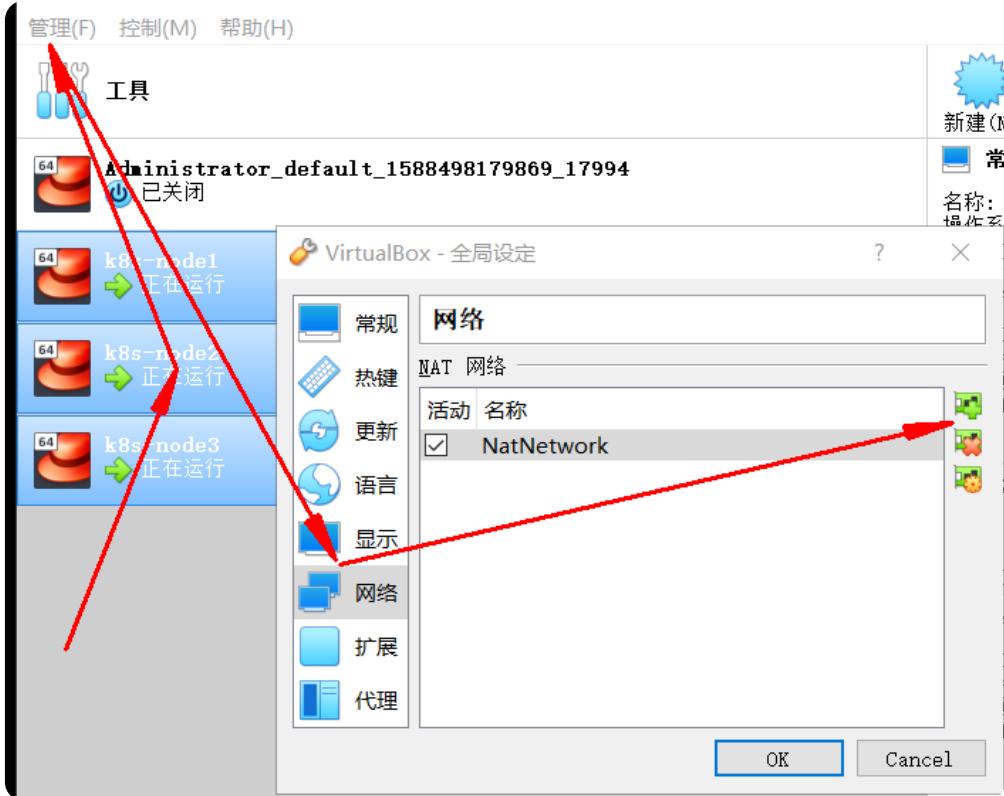
```

原因分析：这是因为它们使用的是端口转发规则，使用同一个地址，通过不同的端口来区分。但是这种端口转发规则在以后的使用中会产生很多不必要的问题，所以需要修改为NAT网络类型。



解决方法：

- 选择三个节点，然后执行“管理”->“全局设定”->“网络”，添加一个NAT网络。



- 分别修改每台设备的网络类型，并刷新重新生成MAC地址。



- 再次查看三个节点的IP

```
✓ root@k8s-node3:~  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:8c:60:61 brd ff:ff:ff:ff:ff:ff  
      inet 10.0.2.5/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0  
        valid_lft 110sec preferred_lft 110sec  
        inet6 fe80::a00:27ff:fe8c:6061/64 scope link  
          valid_lft forever preferred_lft forever  
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:38:9c:8a brd ff:ff:ff:ff:ff:ff  
      inet 192.168.56.102/24 brd 192.168.56.255 scope global noprefixroute eth1  
        valid_lft forever preferred_lft forever
```

```
⚠ root@k8s-node2:~  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:63:69:cf brd ff:ff:ff:ff:ff:ff  
      inet 10.0.2.4/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0  
        valid_lft 1095sec preferred_lft 1095sec  
        inet6 fe80::a00:27ff:fe63:69cf/64 scope link  
          valid_lft forever preferred_lft forever  
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:55:9a:43 brd ff:ff:ff:ff:ff:ff  
      inet 192.168.56.101/24 brd 192.168.56.255 scope global noprefixroute eth1  
        valid_lft forever preferred_lft forever
```

```
⚠ root@k8s-node1:~  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:0a:50:07 brd ff:ff:ff:ff:ff:ff  
      inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0  
        valid_lft 1111sec preferred_lft 1111sec  
        inet6 fe80::a00:27ff:fe0a:5007/64 scope link  
          valid_lft forever preferred_lft forever  
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 08:00:27:a3:ca:c0 brd ff:ff:ff:ff:ff:ff
```

(3) 设置Linux环境（三个节点都执行）

- 关闭防火墙



```
systemctl stop firewalld  
systemctl disable firewalld
```

- 关闭Linux



```
sed -i 's/enforcing/disabled/' /etc/selinux/config  
setenforce 0
```

- 关闭swap

```
swapoff -a #临时关闭  
sed -ri 's/.*swap.*/#&/' /etc/fstab #永久关闭  
free -g #验证, swap必须为0
```

- 添加主机名与IP对应关系：

查看主机名：

```
hostname
```

如果主机名不正确，可以通过“hostnamectl set-hostname <newhostname> :指定新的hostname”命令来进行修改。

```
vi /etc/hosts  
10.0.2.15 k8s-node1  
10.0.2.4 k8s-node2  
10.0.2.5 k8s-node3
```

将桥接的IPV4流量传递到iptables的链：

```
cat > /etc/sysctl.d/k8s.conf <<EOF  
  
net.bridge.bridge-nf-call-ip6tables = 1  
  
net.bridge.bridge-nf-call-iptables = 1  
  
EOF
```

应用规则：



```
sysctl --system
```

疑难问题：遇见提示是只读的文件系统，运行如下命令



```
mount -o remount rw /
```

- date 查看时间（可选）



```
yum -y install ntpupdate  
ntpupdate time.window.com #同步最新时间
```

5) 所有节点安装docker、kubeadm、kubelet、kubectl

Kubernetes默认CRI（容器运行时）为Docker，因此先安装Docker。

(1) 安装Docker

1、卸载之前的docker

```
$ sudo yum remove docker \
    docker-client \
    docker-client-latest \
    docker-common \
    docker-latest \
    docker-latest-logrotate \
    docker-logrotate \
    docker-engine
```

2、安装Docker -CE

```
$ sudo yum install -y yum-utils

$ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

$ sudo yum -y install docker-ce docker-ce-cli containerd.io
```

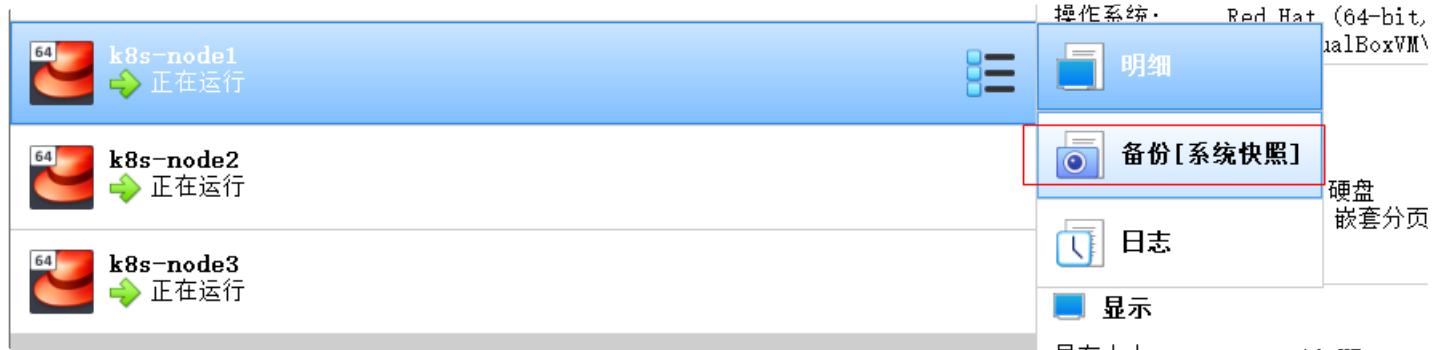
3、配置镜像加速

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": [ "https://ke9h1pt4.mirror.aliyuncs.com" ]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

4、启动Docker && 设置docker开机启动

```
systemctl enable docker
```

基础环境准备好，可以给三个虚拟机备份一下；



(2) 添加阿里与Yum源

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

更多详情见：<https://developer.aliyun.com/mirror/kubernetes>

(3) 安装kubeadm, kubelet和kubectl

```
yum list|grep kube
```

安装



```
yum install -y kubelet-1.17.3 kubeadm-1.17.3 kubectl-1.17.3
```

开机启动



```
systemctl enable kubelet && systemctl start kubelet
```

查看kubelet的状态:



```
systemctl status kubelet
```

查看kubelet版本:



```
[root@k8s-node2 ~]# kubelet --version
Kubernetes v1.17.3
```

6) 部署k8s-master

(1) master节点初始化

在Master节点上，创建并执行master_images.sh



```
#!/bin/bash

images=(  
    kube-apiserver:v1.17.3
```

```

    kube-proxy:v1.17.3
    kube-controller-manager:v1.17.3
    kube-scheduler:v1.17.3
    coredns:1.6.5
    etcd:3.4.3-0
    pause:3.1
)

for imageName in ${images[@]} ; do
    docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/$imageName
#    docker tag registry.cn-hangzhou.aliyuncs.com/google_containers/$imageName
k8s.gcr.io/$imageName
done

```

初始化kubeadm



```
$ kubeadm init \
--apiserver-advertise-address=10.0.2.15 \
--image-repository registry.cn-hangzhou.aliyuncs.com/google_containers \
--kubernetes-version v1.17.3 \
--service-cidr=10.96.0.0/16 \
--pod-network-cidr=10.244.0.0/16
```

注:

- --apiserver-advertise-address=10.0.2.21：这里的IP地址是master主机的地址，为上面的eth0网卡的地址；
-

执行结果:



```
[root@k8s-node1 opt]# kubeadm init \
> --apiserver-advertise-address=10.0.2.15 \
> --image-repository registry.cn-hangzhou.aliyuncs.com/google_containers \
> --kubernetes-version v1.17.3 \
> --service-cidr=10.96.0.0/16 \
> --pod-network-cidr=10.244.0.0/16
W0503 14:07:12.594252 10124 configset.go:202] WARNING: kubeadm cannot validate
component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]
```

```
[init] Using Kubernetes version: v1.17.3
[preflight] Running pre-flight checks
    [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver.
The recommended driver is "systemd". Please follow the guide at
https://kubernetes.io/docs/setup/cri/
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet
connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images
pull'
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [k8s-node1 kubernetes
kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs
[10.96.0.1 10.0.2.15]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [k8s-node1 localhost] and IPs
[10.0.2.15 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [k8s-node1 localhost] and IPs
[10.0.2.15 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
W0503 14:07:30.908642 10124 manifests.go:225] the default kube-apiserver authorization-
mode is "Node,RBAC"; using "Node,RBAC"
[control-plane] Creating static Pod manifest for "kube-scheduler"
W0503 14:07:30.911330 10124 manifests.go:225] the default kube-apiserver authorization-
mode is "Node,RBAC"; using "Node,RBAC"
```

```
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 22.506521 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.18" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node k8s-node1 as control-plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node k8s-node1 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: sg47f3.4asffoi6ijb8ljq
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
#表示kubernetes已经初始化成功了
Your Kubernetes control-plane has initialized successfully!
```

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

```
https://kubernetes.io/docs/concepts/cluster-administration/addons/
```

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 10.0.2.15:6443 --token sg47f3.4asffoi6ijb8ljq \
--discovery-token-ca-cert-hash
sha256:81fccdd29970cbc1b7dc7f171ac0234d53825bdf9b05428fc9e6767436991bfb
[root@k8s-node1 opt]#
```

由于默认拉取镜像地址k8s.cr.io国内无法访问，这里指定阿里云仓库地址。可以手动按照我们的images.sh先拉取镜像。

地址变为：registry.aliyuncs.com/google_containers也可以。

科普：无类别域间路由（Classless Inter-Domain Routing、CIDR）是一个用于给用户分配IP地址以及在互联网上有效第路由IP数据包的对IP地址进行归类的方法。

拉取可能失败，需要下载镜像。

运行完成提前复制：加入集群的令牌。

(2) 测试Kubectl (主节点执行)



```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

详细部署文档：<https://kubernetes.io/docs/concepts/cluster-administration/addons/>



```
$ kubectl get nodes #获取所有节点
```

目前Master状态为notready。等待网络加入完成即可。



```
$ journalctl -u kubelet #查看kubelet日志
```

```
kubeadm join 10.0.2.15:6443 --token sg47f3.4asffoi6ijb8ljq \  
--discovery-token-ca-cert-hash  
sha256:81fccdd29970cbc1b7dc7f171ac0234d53825bdf9b05428fc9e6767436991bfb
```

7) 安装POD网络插件 (CNI)

在master节点上执行按照POD网络插件

```
kubectl apply -f \  
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

以上地址可能被墙，可以直接获取本地已经下载的flannel.yml运行即可，如：

```
[root@k8s-node1 k8s]# kubectl apply -f kube-flannel.yml  
podsecuritypolicy.policy/psp.flannel.unprivileged created  
clusterrole.rbac.authorization.k8s.io/flannel created  
clusterrolebinding.rbac.authorization.k8s.io/flannel created  
serviceaccount/flannel created  
configmap/kube-flannel-cfg created  
daemonset.apps/kube-flannel-ds-amd64 created  
daemonset.apps/kube-flannel-ds-arm64 created  
daemonset.apps/kube-flannel-ds-arm created  
daemonset.apps/kube-flannel-ds-ppc64le created  
daemonset.apps/kube-flannel-ds-s390x created  
[root@k8s-node1 k8s]#
```

同时flannel.yml中指定的images访问不到可以去docker hub找一个wget yml地址

vi 修改yml 所有amd64的地址修改了即可

等待大约3分钟

kubectl get pods -n kube-system 查看指定名称空间的pods

kubectl get pods -all-namespaces 查看所有名称空间的pods

\$ ip link set cni0 down 如果网络出现问题，关闭cni0，重启虚拟机继续测试

执行watch kubectl get pod -n kube-system -o wide 监控pod进度

等待3-10分钟，完全都是running以后继续

查看命名空间：

```
[root@k8s-node1 k8s]# kubectl get ns
NAME        STATUS   AGE
default     Active   30m
kube-node-lease  Active   30m
kube-public    Active   30m
kube-system    Active   30m
[root@k8s-node1 k8s]#
```

```
[root@k8s-node1 k8s]# kubectl get pods --all-namespaces
NAMESPACE      NAME          READY   STATUS    RESTARTS   AGE
kube-system    coredns-546565776c-9sbmk   0/1     Pending   0          31m
kube-system    coredns-546565776c-t68mr   0/1     Pending   0          31m
kube-system    etcd-k8s-node1           1/1     Running   0          31m
kube-system    kube-apiserver-k8s-node1  1/1     Running   0          31m
kube-system    kube-controller-manager-k8s-node1  1/1     Running   0          31m
kube-system    kube-flannel-ds-amd64-6xwth   1/1     Running   0          2m50s
kube-system    kube-proxy-sz2vz          1/1     Running   0          31m
kube-system    kube-scheduler-k8s-node1   1/1     Running   0          31m
[root@k8s-node1 k8s]#
```

查看master上的节点信息：

```
[root@k8s-node1 k8s]# kubectl get nodes
NAME      STATUS   ROLES   AGE   VERSION
k8s-node1  Ready    master   34m   v1.17.3  #status为ready才能够执行下面的命令
[root@k8s-node1 k8s]#
```

最后再次执行，并且分别在“k8s-node2”和“k8s-node3”上也执行这里命令：

```
kubeadm join 10.0.2.15:6443 --token sg47f3.4asffoi6ijb8ljhq \
--discovery-token-ca-cert-hash
sha256:81fccdd29970cbc1b7dc7f171ac0234d53825bdf9b05428fc9e6767436991bfb
```

```
[root@k8s-node1 opt]# kubectl get nodes;
NAME      STATUS    ROLES      AGE      VERSION
k8s-node1  Ready     master     47m     v1.17.3
k8s-node2  NotReady <none>     75s     v1.17.3
k8s-node3  NotReady <none>     76s     v1.17.3
[root@k8s-node1 opt]#
```

监控pod进度

```
watch kubectl get pod -n kube-system -o wide
```

等到所有的status都变为running状态后，再次查看节点信息：

```
[root@k8s-node1 ~]# kubectl get nodes;
NAME      STATUS    ROLES      AGE      VERSION
k8s-node1  Ready     master     3h50m   v1.17.3
k8s-node2  Ready     <none>    3h3m    v1.17.3
k8s-node3  Ready     <none>    3h3m    v1.17.3
[root@k8s-node1 ~]#
```

8) 加入kubernetes的Node节点

在node节点中执行，向集群中添加新的节点，执行在kubeadm init 输出的kubeadm join命令；
确保node节点成功：

token过期怎么办

kubeadm token create --print-join-command

9) 入门操作kubernetes集群

1、在主节点上部署一个tomcat



```
kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8
```

获取所有的资源：



```
[root@k8s-node1 k8s]# kubectl get all
NAME                           READY   STATUS            RESTARTS   AGE
pod/tomcat6-7b84fb5fdc-cfd8g  0/1    ContainerCreating  0          41s

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP  10.96.0.1    <none>        443/TCP   70m

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/tomcat6  0/1     1           0          41s

NAME           DESIRED  CURRENT   READY   AGE
replicaset.apps/tomcat6-7b84fb5fdc  1        1         0       41s
[root@k8s-node1 k8s]#
```

kubectl get pods -o wide 可以获取到tomcat部署信息，能够看到它被部署到了k8s-node2上了

```
[root@k8s-node1 k8s]# kubectl get all -o wide
NAME                                     READY   STATUS    RESTARTS   AGE     IP           NODE
  NOMINATED NODE   READINESS GATES
pod/tomcat6-7b84fb5fdc-cfd8g   1/1     Running   0          114s   10.244.2.2   k8s-node2
<none>           <none>

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/kubernetes   ClusterIP  10.96.0.1   <none>        443/TCP  71m   <none>

NAME           READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES
SELECTOR
deployment.apps/tomcat6   1/1     1           1           114s   tomcat
tomcat:6.0.53-jre8   app=tomcat6

NAME           DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES   SELECTOR
replicaset.apps/tomcat6-7b84fb5fdc   1         1         1       114s   tomcat
tomcat:6.0.53-jre8   app=tomcat6,pod-template-hash=7b84fb5fdc
[root@k8s-node1 k8s]#
```

查看node2节点上，下载了哪些镜像：

```
[root@k8s-node2 opt]# docker images
REPOSITORY                                     TAG
IMAGE ID          CREATED          SIZE
registry.cn-hangzhou.aliyuncs.com/google_containers/kube-proxy   v1.17.3
0d40868643c6   2 weeks ago     117MB
registry.cn-hangzhou.aliyuncs.com/google_containers/pause        3.2
80d28bedfe5d   2 months ago    683kB
quay.io/coreos/flannel                           v0.11.0-amd64
ff281650a721   15 months ago   52.6MB
tomcat                                         6.0.53-jre8
49ab0583115a   2 years ago    290MB
[root@k8s-node2 opt]#
```

查看Node2节点上，正在运行的容器：

```
[root@k8s-node2 opt]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
9194cc4f0b7a        tomcat
"catalina.sh run"   2 minutes ago      Up 2 minutes
k8s_tomcat_tomcat6-7b84fb5fdc-cfd8g_default_0c9ebba2-992d-4c0e-99ef-3c4c3294bc59_0
f44af0c7c345        registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.2
"/pause"            3 minutes ago      Up 3 minutes
k8s_POD_tomcat6-7b84fb5fdc-cfd8g_default_0c9ebba2-992d-4c0e-99ef-3c4c3294bc59_0
ef74c90491e4        ff281650a721
"/opt/bin/flanneld ..." 20 minutes ago     Up 20 minutes
k8s_kube-flannel_kube-flannel-ds-amd64-5xs5j_kube-system_11a94346-316d-470b-9668-
c15ce183abec_0
c8a524e5a193        registry.cn-hangzhou.aliyuncs.com/google_containers/kube-proxy
"/usr/local/bin/kube..." 25 minutes ago     Up 25 minutes
k8s_kube-proxy_kube-proxy-mvlnk_kube-system_519de79a-e8d8-4b1c-a74e-94634cebabcce_0
4590685c519a        registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.2
"/pause"            26 minutes ago      Up 26 minutes
k8s_POD_kube-flannel-ds-amd64-5xs5j_kube-system_11a94346-316d-470b-9668-c15ce183abec_0
54e00af5cde4        registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.2
"/pause"            26 minutes ago      Up 26 minutes
k8s_POD_kube-proxy-mvlnk_kube-system_519de79a-e8d8-4b1c-a74e-94634cebabcce_0
[root@k8s-node2 opt]#
```

在node1上执行:

```
[root@k8s-node1 k8s]# kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
tomcat6-7b84fb5fdc-cfd8g  1/1     Running   0          5m35s

[root@k8s-node1 k8s]# kubectl get pods --all-namespaces
NAMESPACE      NAME                  READY   STATUS    RESTARTS   AGE
default        tomcat6-7b84fb5fdc-cfd8g  1/1     Running   0          163m
kube-system   coredns-546565776c-9sbmk  1/1     Running   0          3h52m
kube-system   coredns-546565776c-t68mr  1/1     Running   0          3h52m
kube-system   etcd-k8s-node1           1/1     Running   0          3h52m
kube-system   kube-apiserver-k8s-node1  1/1     Running   0          3h52m
kube-system   kube-controller-manager-k8s-node1  1/1     Running   0          3h52m
```

```

kube-system   kube-flannel-ds-amd64-5xs5j      1/1    Running   0        3h6m
kube-system   kube-flannel-ds-amd64-6xwth      1/1    Running   0        3h24m
kube-system   kube-flannel-ds-amd64-fvnvx      1/1    Running   0        3h6m
kube-system   kube-proxy-7tkvl                 1/1    Running   0        3h6m
kube-system   kube-proxy-mvlnk                1/1    Running   0        3h6m
kube-system   kube-proxy-sz2vz                 1/1    Running   0        3h52m
kube-system   kube-scheduler-k8s-node1        1/1    Running   0        3h52m
[root@k8s-node1 ~]#

```

从前面看到tomcat部署在Node2上，现在模拟因为各种原因宕机的情况，将node2关闭电源，观察情况。

```

[root@k8s-node1 ~]# kubectl get nodes
NAME     STATUS   ROLES   AGE     VERSION
k8s-node1  Ready    master  4h4m   v1.17.3
k8s-node2  NotReady <none>  3h18m  v1.17.3
k8s-node3  Ready    <none>  3h18m  v1.17.3
[root@k8s-node1 ~]#

```

```

[root@k8s-node1 ~]# kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE     IP          NODE
NOMINATED-NODE   READINESS GATES
tomcat6-7b84fb5fdc-cfd8g   1/1    Running   0          177m   10.244.2.2   k8s-node2
<none>           <none>
[root@k8s-node1 ~]#

```

```

[root@k8s-node1 ~]# kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE     IP          NODE
NOMINATED-NODE   READINESS GATES
tomcat6-5f7ccf4cb9-gjfmq   1/1    Running   0          3m13s  10.244.1.2   k8s-node2
tomcat6-5f7ccf4cb9-qn728   0/1    Terminating   1          13m    <none>      k8s-node3
[root@k8s-node1 ~]#

```

2、暴露nginx访问

在master上执行

```
kubectl expose deployment tomcat6 --port=80 --target-port=8080 --type=NodePort
```

pod的80映射容器的8080；server会带来pod的80

查看服务：

```
[root@k8s-node1 ~]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP      12h
tomcat6    NodePort   10.96.24.191    <none>        80:30526/TCP  49s
[root@k8s-node1 ~]#
```

```
[root@k8s-node1 ~]# kubectl get svc -o wide
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE      SELECTOR
kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP      12h      <none>
tomcat6    NodePort   10.96.24.191    <none>        80:30526/TCP  3m30s   app=tomcat6
[root@k8s-node1 ~]#
```

<http://192.168.56.100:30526/>

← → C ⓘ Not secure | 192.168.56.100:30526



[Administration](#)
[Status](#)
[Tomcat Manager](#)

[Documentation](#)
[Release Notes](#)
[Change Log](#)
[Tomcat Documentation](#)

If you're seeing this page via a web browser, it mea

As you may have guessed by now, this is the default Tomcat home page. It can be found on
\$CATALINA_HOME/webapps/ROOT/index.html

where "\$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing th
installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Pro
and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with ce

```
[root@k8s-node1 ~]# kubectl get all
NAME                               READY   STATUS    RESTARTS   AGE
pod/tomcat6-7b84fb5fdc-qt5jm     1/1    Running   0          13m

NAME             TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP      12h
service/tomcat6   NodePort  10.96.24.191  <none>        80:30526/TCP  9m50s

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/tomcat6  1/1       1           1           11h

NAME           DESIRED  CURRENT  READY   AGE
replicaset.apps/tomcat6-7b84fb5fdc  1         1         1       11h
[root@k8s-node1 ~]#
```

3、动态扩容测试

kubectl get deployment

```
[root@k8s-node1 ~]# kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
tomcat6  2/2     2           2           11h
[root@k8s-node1 ~]#
```

应用升级： kubectl set image (--help查看帮助)

扩容： kubectl scale --replicas=3 deployment tomcat6

```
[root@k8s-node1 ~]# kubectl scale --replicas=3 deployment tomcat6
deployment.apps/tomcat6 scaled
[root@k8s-node1 ~]#

[root@k8s-node1 ~]# kubectl get pods -o wide
NAME                               READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED-NODE   READINESS GATES
```

```

tomcat6-7b84fb5fdc-hdgmc  1/1    Running   0          61s    10.244.2.5    k8s-node2
<none>           <none>
tomcat6-7b84fb5fdc-qt5jm  1/1    Running   0          19m    10.244.1.2    k8s-node3
<none>           <none>
tomcat6-7b84fb5fdc-vlrh6  1/1    Running   0          61s    10.244.2.4    k8s-node2
<none>           <none>

[root@k8s-node1 ~]# kubectl get svc -o wide
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE      SELECTOR
kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP      13h      <none>
tomcat6   NodePort   10.96.24.191  <none>        80:30526/TCP  16m      app=tomcat6
[root@k8s-node1 ~]#

```

扩容了多份，所有无论访问哪个node的指定端口，都可以访问到tomcat6

<http://192.168.56.101:30526/>



TM Apache Tomcat

Administration

Status

Tomcat Manager

If you're seeing this page via a web browser, it means you have successfully reached the default Tomcat home page. It is located at \$CATALINA_HOME/webapps/ROOT/index.html

As you may have guessed by now, this is the default Tomcat home page. It is located at \$CATALINA_HOME/webapps/ROOT/index.html

<http://192.168.56.102:30526/>



TM Apache Tomcat

Administration

Status

Tomcat Manager

If you're seeing this page via a web browser, it means you have successfully reached the default Tomcat home page. It is located at \$CATALINA_HOME/webapps/ROOT/index.html

As you may have guessed by now, this is the default Tomcat home page. It is located at \$CATALINA_HOME/webapps/ROOT/index.html

缩容：kubectl scale --replicas=2 deployment tomcat6



```
[root@k8s-node1 ~]# kubectl scale --replicas=2 deployment tomcat6
deployment.apps/tomcat6 scaled
[root@k8s-node1 ~]# kubectl get pods -o wide
NAME                               READY   STATUS    RESTARTS   AGE     IP           NODE
  NOMINATED-NODE   READINESS GATES
tomcat6-7b84fb5fdc-hdgmc   0/1     Terminating   0          4m47s  <none>       k8s-
node2      <none>            <none>
tomcat6-7b84fb5fdc-qt5jm   1/1     Running     0          22m    10.244.1.2   k8s-
node3      <none>            <none>
tomcat6-7b84fb5fdc-vlrh6   1/1     Running     0          4m47s  10.244.2.4   k8s-
node2      <none>            <none>
[root@k8s-node1 ~]#
```

4、以上操作的yaml获取
参照k8s细节

5、删除

kubectl get all



#查看所有资源

```
[root@k8s-node1 ~]# kubectl get all
NAME                               READY   STATUS    RESTARTS   AGE
pod/tomcat6-7b84fb5fdc-qt5jm   1/1     Running   0          26m
pod/tomcat6-7b84fb5fdc-vlrh6   1/1     Running   0          8m16s

NAME              TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)   AGE
service/kubernetes  ClusterIP  10.96.0.1   <none>        443/TCP   13h
service/tomcat6   NodePort   10.96.24.191 <none>        80:30526/TCP 22m

NAME                               READY   UP-TO-DATE  AVAILABLE   AGE
deployment.apps/tomcat6   2/2     2           2           11h

NAME              DESIRED  CURRENT  READY   AGE
replicaset.apps/tomcat6-7b84fb5fdc  2        2        2       11h
```

```

[root@k8s-node1 ~]# 
#删除deployment.apps/tomcat6
[root@k8s-node1 ~]# kubectl delete deployment.apps/tomcat6
deployment.apps "tomcat6" deleted

#查看剩余的资源
[root@k8s-node1 ~]# kubectl get all
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/kubernetes  ClusterIP  10.96.0.1    <none>        443/TCP     13h
service/tomcat6  NodePort   10.96.24.191  <none>        80:30526/TCP 30m

[root@k8s-node1 ~]#
[root@k8s-node1 ~]#
#删除service/tomcat6
[root@k8s-node1 ~]# kubectl delete service/tomcat6
service "tomcat6" deleted
[root@k8s-node1 ~]# kubectl get all
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/kubernetes  ClusterIP  10.96.0.1    <none>        443/TCP     13h

[root@k8s-node1 ~]#

```

kubectl delete deploye/nginx
 kubectl delete service/nginx-service

3、K8s细节

1、kubectl文档

<https://kubernetes.io/zh/docs/reference/kubectl/overview/>

2、资源类型

<https://kubernetes.io/zh/docs/reference/kubectl/overview/#%e8%b5%84%e6%ba%90%e7%b1%bb%e5%9e%8b>

3、格式化输出

<https://kubernetes.io/zh/docs/reference/kubectl/overview/>

所有 `kubectl` 命令的默认输出格式都是人类可读的纯文本格式。要以特定格式向终端窗口输出详细信息，可以将 `-o` 或 `--output` 参数添加到受支持的 `kubectl` 命令中。

语法



```
kubectl [command] [TYPE] [NAME] -o=<output_format>
```

根据 `kubectl` 操作，支持以下输出格式：

Output format	Description
<code>-o custom-columns=</code>	使用逗号分隔的 <u>自定义列</u> 列表打印表。
<code>-o custom-columns-file=</code>	使用 `` 文件中的 <u>自定义列</u> 模板打印表。
<code>-o json</code>	输出 JSON 格式的 API 对象
<code>-o jsonpath=</code>	打印 <u>jsonpath</u> 表达式定义的字段
<code>-o jsonpath-file=</code>	打印 `` 文件中 <u>jsonpath</u> 表达式定义的字段。
<code>-o name</code>	仅打印资源名称而不打印任何其他内容。
<code>-o wide</code>	以纯文本格式输出，包含任何附加信息。对于 pod 包含节点名。
<code>-o yaml</code>	输出 YAML 格式的 API 对象。

示例

在此示例中，以下命令将单个 pod 的详细信息输出为 YAML 格式的对象：



```
kubectl get pod web-pod-13je7 -o yaml
```

请记住：有关每个命令支持哪种输出格式的详细信息，请参阅 [kubectl](#) 参考文档。

--dry-run:

--dry-run='none': Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

值必须为none, server或client。如果是客户端策略，则只打印该发送对象，但不发送它。如果服务器策略，提交服务器端请求而不持久化资源。

也就是说，通过--dry-run选项，并不会真正的执行这条命令。



```
[root@k8s-node1 ~]# kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8 --dry-run -o yaml
W0504 03:39:08.389369      8107 helpers.go:535] --dry-run is deprecated and can be replaced with --dry-run=client.

apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: tomcat6
    name: tomcat6
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat6
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: tomcat6
    spec:
      containers:
        - image: tomcat:6.0.53-jre8
          name: tomcat
```

```
        resources: {}
status: {}
[root@k8s-node1 ~]#
```

实际上我们也可以将这个yaml输出到文件，然后使用kubectl apply -f来应用它

```
#输出到tomcat6.yaml
[root@k8s-node1 ~]# kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8 --dry-run -o yaml >tomcat6.yaml
W0504 03:46:18.180366    11151 helpers.go:535] --dry-run is deprecated and can be replaced with --dry-run=client.

#修改副本数为3
[root@k8s-node1 ~]# cat tomcat6.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: tomcat6
    name: tomcat6
spec:
  replicas: 3      #修改副本数为3
  selector:
    matchLabels:
      app: tomcat6
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: tomcat6
    spec:
      containers:
        - image: tomcat:6.0.53-jre8
          name: tomcat
          resources: {}
status: {}

#应用tomcat6.yaml
[root@k8s-node1 ~]# kubectl apply -f tomcat6.yaml
deployment.apps/tomcat6 created
[root@k8s-node1 ~]#
```

查看pods:



```
[root@k8s-node1 ~]# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
tomcat6-7b84fb5fdc-5jh6t  1/1     Running   0          8s
tomcat6-7b84fb5fdc-8lhwv  1/1     Running   0          8s
tomcat6-7b84fb5fdc-j4qmh  1/1     Running   0          8s
[root@k8s-node1 ~]#
```

查看某个pod的具体信息:



```
[root@k8s-node1 ~]# kubectl get pods tomcat6-7b84fb5fdc-5jh6t -o yaml
```



```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-05-04T03:50:47Z"
  generateName: tomcat6-7b84fb5fdc-
  labels:
    app: tomcat6
    pod-template-hash: 7b84fb5fdc
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:generateName: {}
      f:labels:
        .: {}
      f:app: {}
      f:pod-template-hash: {}
    f:ownerReferences:
      .: {}
```

```
k:{"uid":"292bfe3b-dd63-442e-95ce-c796ab5bdcc1"}:  
  .: {}  
  f:apiVersion: {}  
  f:blockOwnerDeletion: {}  
  f:controller: {}  
  f:kind: {}  
  f:name: {}  
  f:uid: {}  
  
f:spec:  
  f:containers:  
    k:{"name":"tomcat"}:  
      .: {}  
      f:image: {}  
      f:imagePullPolicy: {}  
      f:name: {}  
      f:resources: {}  
      f:terminationMessagePath: {}  
      f:terminationMessagePolicy: {}  
  
    f:dnsPolicy: {}  
    f:enableServiceLinks: {}  
    f:restartPolicy: {}  
    f:schedulerName: {}  
    f:securityContext: {}  
    f:terminationGracePeriodSeconds: {}  
  
  manager: kube-controller-manager  
  operation: Update  
  time: "2020-05-04T03:50:47Z"  
  
- apiVersion: v1  
  fieldsType: FieldsV1  
  fieldsV1:  
    f:status:  
      f:conditions:  
        k:{"type":"ContainersReady"}:  
          .: {}  
          f:lastProbeTime: {}  
          f:lastTransitionTime: {}  
          f:status: {}  
          f:type: {}  
        k:{"type":"Initialized"}:  
          .: {}  
          f:lastProbeTime: {}  
          f:lastTransitionTime: {}  
          f:status: {}  
          f:type: {}  
        k:{"type":"Ready"}:  
          .: {}
```

```
        f:lastProbeTime: {}
        f:lastTransitionTime: {}
        f:status: {}
        f:type: {}

      f:containerStatuses: {}
      f:hostIP: {}
      f:phase: {}
      f:podIP: {}
      f:podIPs:
        ..: {}

      k:{ "ip": "10.244.2.7"}:
        ..: {}
        f:ip: {}

      f:startTime: {}

    manager: kubelet
    operation: Update
    time: "2020-05-04T03:50:49Z"
  name: tomcat6-7b84fb5fdc-5jh6t
  namespace: default
  ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: true
      controller: true
      kind: ReplicaSet
      name: tomcat6-7b84fb5fdc
      uid: 292bfe3b-dd63-442e-95ce-c796ab5bdcc1
  resourceVersion: "46229"
  selfLink: /api/v1/namespaces/default/pods/tomcat6-7b84fb5fdc-5jh6t
  uid: 2f661212-3b03-47e4-bcb8-79782d5c7578

spec:
  containers:
    - image: tomcat:6.0.53-jre8
      imagePullPolicy: IfNotPresent
      name: tomcat
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
        - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
          name: default-token-bxqtw
          readOnly: true
    dnsPolicy: ClusterFirst
    enableServiceLinks: true
    nodeName: k8s-node2
    priority: 0
    restartPolicy: Always
```

```
schedulerName: default-scheduler
securityContext: {}
serviceAccount: default
serviceAccountName: default
terminationGracePeriodSeconds: 30
tolerations:
- effect: NoExecute
  key: node.kubernetes.io/not-ready
  operator: Exists
  tolerationSeconds: 300
- effect: NoExecute
  key: node.kubernetes.io/unreachable
  operator: Exists
  tolerationSeconds: 300
volumes:
- name: default-token-bxqtw
  secret:
    defaultMode: 420
    secretName: default-token-bxqtw
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-05-04T03:50:47Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2020-05-04T03:50:49Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2020-05-04T03:50:49Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2020-05-04T03:50:47Z"
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID:
    docker://18eb0798384ea44ff68712cda9be94b6fb96265206c554a15cee28c288879304
      image: tomcat:6.0.53-jre8
      imageID: docker-
    pullable://tomcat@sha256:8c643303012290f89c6f6852fa133b7c36ea6fbb8eb8b8c9588a432beb24dc5d
      lastState: {}
      name: tomcat
      ready: true
```

```
restartCount: 0
started: true
state:
  running:
    startedAt: "2020-05-04T03:50:49Z"
hostIP: 10.0.2.4
phase: Running
podIP: 10.244.2.7
podIPs:
- ip: 10.244.2.7
qosClass: BestEffort
startTime: "2020-05-04T03:50:47Z"
```

命令参考

The screenshot shows a web browser displaying the Kubernetes documentation for the 'create' command. The URL in the address bar is <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#create>. The page title is 'create'. On the left, there is a sidebar with a search icon and a list of resource types: 'create', 'clusterrole', 'clusterrolebinding', 'configmap', and 'secret'. The main content area has a large heading 'create' and the sub-instruction 'Create a resource from a file or from stdin.'

service的意义

3、Service 的意义

1、部署一个 nginx

```
kubectl create deployment nginx --image=nginx
```

2、暴露 nginx 访问

```
kubectl expose deployment nginx --port=80 --type=NodePort
```

18668062061

统一应用访问入口；

Service 管理一组 Pod。

防止 Pod 失联（服务发现）、定义一组 Pod 的访问策略

现在 Service 我们使用 NodePort 的方式暴露，这样访问每个节点的端口，都可以访问一个 Pod，如果节点宕机，就会出现问题。

前面我们通过命令行的方式，部署和暴露了tomcat，实际上也可以通过yaml的方式来完成这些操作。



#这些操作实际上是为了获取Deployment的yaml模板

```
[root@k8s-node1 ~]# kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8 --dry-run -o yaml >tomcat6-deployment.yaml
W0504 04:13:28.265432    24263 helpers.go:535] --dry-run is deprecated and can be replaced with --dry-run=client.
[root@k8s-node1 ~]# ls tomcat6-deployment.yaml
tomcat6-deployment.yaml
[root@k8s-node1 ~]#
```

修改“tomcat6-deployment.yaml”内容如下：



```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

labels:
  app: tomcat6
  name: tomcat6
spec:
  replicas: 3
  selector:
    matchLabels:
      app: tomcat6
template:
  metadata:
    labels:
      app: tomcat6
spec:
  containers:
  - image: tomcat:6.0.53-jre8
    name: tomcat

```



#部署

```
[root@k8s-node1 ~]# kubectl apply -f tomcat6-deployment.yaml
deployment.apps/tomcat6 configured
```

#查看资源

```
[root@k8s-node1 ~]# kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/tomcat6-7b84fb5fdc-5jh6t  1/1     Running   0          27m
pod/tomcat6-7b84fb5fdc-8lhwv  1/1     Running   0          27m
pod/tomcat6-7b84fb5fdc-j4qmh  1/1     Running   0          27m

NAME           TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1   <none>       443/TCP   14h

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/tomcat6  3/3     3           3           27m

NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/tomcat6-7b84fb5fdc  3         3         3        27m

[root@k8s-node1 ~]#
```

```
● ○ ●  
kubectl expose deployment tomcat6 --port=80 --target-port=8080 --type=NodePort --dry-run  
-o yaml
```

```
● ○ ●  
apiVersion: v1  
kind: Service  
metadata:  
  creationTimestamp: null  
  labels:  
    app: tomcat6  
    name: tomcat6  
spec:  
  ports:  
  - port: 80  
    protocol: TCP  
    targetPort: 8080  
  selector:  
    app: tomcat6  
  type: NodePort  
status:  
  loadBalancer: {}
```

将这段输出和“tomcat6-deployment.yaml”进行拼接，表示部署完毕并进行暴露服务：

```
● ○ ●  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  labels:  
    app: tomcat6  
    name: tomcat6  
spec:  
  replicas: 3  
  selector:  
    matchLabels:
```

```

app: tomcat6
template:
  metadata:
    labels:
      app: tomcat6
  spec:
    containers:
      - image: tomcat:6.0.53-jre8
        name: tomcat
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: tomcat6
    name: tomcat6
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
  selector:
    app: tomcat6
  type: NodePort

```

部署并暴露服务

[root@k8s-node1 ~]# kubectl apply -f tomcat6-deployment.yaml
 deployment.apps/tomcat6 created
 service/tomcat6 created

查看服务和部署信息

NAME	READY	STATUS	RESTARTS	AGE
pod/tomcat6-7b84fb5fdc-dsqmb	1/1	Running	0	4s
pod/tomcat6-7b84fb5fdc-gbmxc	1/1	Running	0	5s

```

pod/tomcat6-7b84fb5fdc-kjlc6  1/1    Running   0          4s
NAME                  TYPE      CLUSTER-IP        EXTERNAL-IP      PORT(S)       AGE
service/kubernetes   ClusterIP  10.96.0.1        <none>           443/TCP      14h
service/tomcat6      NodePort   10.96.147.210    <none>           80:30172/TCP  4s

NAME                READY   UP-TO-DATE  AVAILABLE  AGE
deployment.apps/tomcat6  3/3     3           3          5s
NAME              DESIRED  CURRENT  READY   AGE
replicaset.apps/tomcat6-7b84fb5fdc  3        3        3      5s
[root@k8s-node1 ~]#

```

访问node1, node1和node3的30172端口:



```

[root@k8s-node1 ~]# curl -I http://192.168.56.{100,101,102}:30172/
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"7454-1491118183000"
Last-Modified: Sun, 02 Apr 2017 07:29:43 GMT
Content-Type: text/html
Content-Length: 7454
Date: Mon, 04 May 2020 04:35:35 GMT

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"7454-1491118183000"
Last-Modified: Sun, 02 Apr 2017 07:29:43 GMT
Content-Type: text/html
Content-Length: 7454
Date: Mon, 04 May 2020 04:35:35 GMT

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"7454-1491118183000"
Last-Modified: Sun, 02 Apr 2017 07:29:43 GMT
Content-Type: text/html
Content-Length: 7454
Date: Mon, 04 May 2020 04:35:35 GMT

```

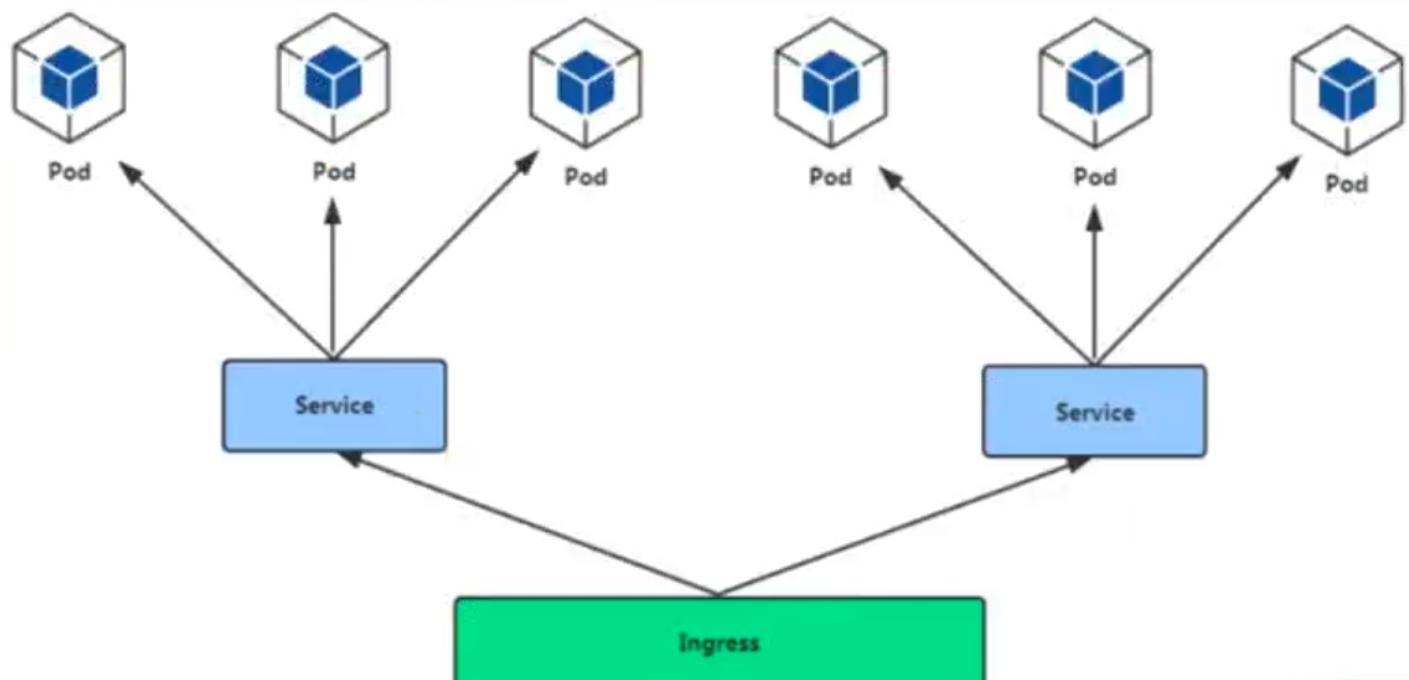
```
[root@k8s-node1 ~]#
```

Ingress

通过Ingress发现pod进行关联。基于域名访问

通过Ingress controller实现POD负载均衡

支持TCP/UDP 4层负载均衡和HTTP 7层负载均衡



步骤：

(1) 部署Ingress controller

执行“k8s/ingress-controller.yaml”

```
[root@k8s-node1 k8s]# kubectl apply -f ingress-controller.yaml
namespace/ingress-nginx created
configmap/nginx-configuration created
configmap/tcp-services created
configmap/udp-services created
serviceaccount/nginx-ingress-serviceaccount created
clusterrole.rbac.authorization.k8s.io/nginx-ingress-clusterrole created
role.rbac.authorization.k8s.io/nginx-ingress-role created
rolebinding.rbac.authorization.k8s.io/nginx-ingress-role-nisa-binding created
clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress-clusterrole-nisa-binding
created
daemonset.apps/nginx-ingress-controller created
service/ingress-nginx created
[root@k8s-node1 k8s]#
```

查看

```
[root@k8s-node1 k8s]# kubectl get pods --all-namespaces
NAMESPACE      NAME           READY   STATUS        RESTARTS
AGE
default        tomcat6-7b84fb5fdc-dsqmb   1/1    Running      0
16m
default        tomcat6-7b84fb5fdc-gbmxc   1/1    Running      0
16m
default        tomcat6-7b84fb5fdc-kjlc6   1/1    Running      0
16m
ingress-nginx  nginx-ingress-controller-9q6cs  0/1    ContainerCreating  0
40s
ingress-nginx  nginx-ingress-controller-qx572   0/1    ContainerCreating  0
40s
kube-system    coredns-546565776c-9sbmk    1/1    Running      1
14h
kube-system    coredns-546565776c-t68mr    1/1    Running      1
14h
kube-system    etcd-k8s-node1            1/1    Running      1
14h
kube-system    kube-apiserver-k8s-node1  1/1    Running      1
14h
```

kube-system	kube-controller-manager-k8s-node1	1/1	Running	1
14h				
kube-system	kube-flannel-ds-amd64-5xs5j	1/1	Running	2
13h				
kube-system	kube-flannel-ds-amd64-6xwth	1/1	Running	2
14h				
kube-system	kube-flannel-ds-amd64-fvnvx	1/1	Running	1
13h				
kube-system	kube-proxy-7tkvl	1/1	Running	1
13h				
kube-system	kube-proxy-mvlnk	1/1	Running	2
13h				
kube-system	kube-proxy-sz2vz	1/1	Running	1
14h				
kube-system	kube-scheduler-k8s-node1	1/1	Running	1
14h				
[root@k8s-node1 k8s]#				

这里master节点负责调度，具体执行交给node2和node3来完成，能够看到它们正在下载镜像

[root@k8s-node1 k8s]# kubectl get pods --all-namespaces	NAME	READY	STATUS	RESTARTS	AGE
1 NAMESPACE					
2 default	tomcat6-7b84fb5fdc-dsqmb	1/1	Running	0	16m
3 default	tomcat6-7b84fb5fdc-gbmxc	1/1	Running	0	16m
4 default	tomcat6-7b84fb5fdc-kjlc6	1/1	Running	0	16m
5 ingress-nginx	nginx-ingress-controller-9q6cs	0/1	ContainerCreating	0	40s
6 ingress-nginx	nginx-ingress-controller-qx572	0/1	ContainerCreating	0	40s
7 kube-system	coredns-546565776c-9sbmk	1/1	Running	1	14h
8 kube-system	coredns-546565776c-t68mr	1/1	Running	1	14h
9 kube-system	etcd-k8s-node1	1/1	Running	1	14h

(2) 创建Ingress规则

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web
spec:
  rules:
  - host: tomcat6.kubernetes.com
    http:
      paths:
        - backend:
            serviceName: tomcat6
            servicePort: 80
```

```
[root@k8s-node1 k8s]# touch ingress-tomcat6.yaml
#将上面的规则，添加到ingress-tomcat6.yaml文件中
[root@k8s-node1 k8s]# vi  ingress-tomcat6.yaml

[root@k8s-node1 k8s]# kubectl apply -f ingress-tomcat6.yaml
ingress.extensions/web created
[root@k8s-node1 k8s]#
```

修改本机的hosts文件，添加如下的域名转换规则：

```
192.168.56.102 tomcat6.kubernetes.com
```

测试：<http://tomcat6.kubernetes.com/>



TM Apache Tomcat

Administration

[Status](#)

[Tomcat Manager](#)

Documentation

[Release Notes](#)

[Change Log](#)

[Tomcat Documentation](#)

If you're seeing this page via a web browser, it mea

As you may have guessed by now, this is the default Tomcat home page. It can be found on

`$CATALINA_HOME/webapps/ROOT/index.html`

where "`$CATALINA_HOME`" is the root of the Tomcat installation directory. If you're seeing th
installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Pro
and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with ce

并且集群中即便有一个节点不可用，也不影响整体的运行。

安装kubernetes可视化界面——DashBoard

1、部署DashBoard



```
$ kubectl apply -f kubernetes-dashboard.yaml
```

文件在“k8s”源码目录提供

2、暴露DashBoard为公共访问

默认DashBoard只能集群内部访问，修改Service为NodePort类型，暴露到外部



```
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
```

```
name: kubernetes-dashboard
namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 3001
  selector:
    k8s-app: kubernetes-dashboard
```

访问地址: <http://NodeIP:30001>

3、创建授权账号



```
$ kubectl create serviceaccount dashboard-admin -n kube-system
```



```
$ kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin
```



```
$ kubectl describe secrets -n kube-system $( kubectl -n kube-system get secret | awk '/dashboard-admin/{print $1}' )
```

使用输出的token登录dashboard



集群

命名空间

节点

持久化存储卷

角色

存储类

命名空间

default

命名空间

名称

标签

 default kube-public kube-system

kubesphere

默认的dashboard没啥用，我们用kubesphere可以打通全部的devops链路，kubesphere集成了很多套件，集群要求比较高

<https://kubesphere.io>

kuboard也很不错，集群要求不高

<https://kuboard.cn/support/>

1、简洁

kubesphere是一款面向云原生设计的开源项目，在目前主流容器调度平台kubernets智商构建的分布式多用户容器管理平台，提供简单易用的操作界面以及向导式操作方式，在降低用户使用容器调度平台学习成本的同时，极大降低开发、测试、运维的日常工作的复杂度。

2、安装前提提交

1、安装helm (master节点执行)

helm是kubernetes的包管理器。包管理器类似于在Ubuntu中使用的apt, centos中的yum或者python中的pip一样,能够快速查找, 下载和安装软件包。Helm有客户端组件helm和服务端组件Tiller组成, 能够将一组K8S资源打包统一管理, 是查找、共享和使用为Kubernetes构建的软件的最佳方式。

1) 安装


```
curl -L https://git.io/get_helm.sh|bash
```

由于被墙的原因, 使用我们给定的get_helm.sh。



```
[root@k8s-node1 k8s]# ll
total 68
-rw-r--r-- 1 root root 7149 Feb 27 01:58 get_helm.sh
-rw-r--r-- 1 root root 6310 Feb 28 05:16 ingress-controller.yaml
-rw-r--r-- 1 root root 209 Feb 28 13:18 ingress-demo.yaml
-rw-r--r-- 1 root root 236 May 4 05:09 ingress-tomcat6.yaml
-rwxr--r-- 1 root root 15016 Feb 26 15:05 kube-flannel.yaml
-rw-r--r-- 1 root root 4737 Feb 26 15:38 kubernetes-dashboard.yaml
-rw-r--r-- 1 root root 3841 Feb 27 01:09 kubesphere-complete-setup.yaml
-rw-r--r-- 1 root root 392 Feb 28 11:33 master_images.sh
-rw-r--r-- 1 root root 283 Feb 28 11:34 node_images.sh
-rw-r--r-- 1 root root 1053 Feb 28 03:53 product.yaml
-rw-r--r-- 1 root root 931 May 3 10:08 Vagrantfile
[root@k8s-node1 k8s]# sh get_helm.sh
Downloading https://get.helm.sh/helm-v2.16.6-linux-amd64.tar.gz
Preparing to install helm and tiller into /usr/local/bin
helm installed into /usr/local/bin/helm
tiller installed into /usr/local/bin/tiller
Run 'helm init' to configure helm.
[root@k8s-node1 k8s]#
```

2) 验证版本



```
helm version
```

3) 创建权限 (master执行)

创建helm-rbac.yaml，写入如下内容



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: kubernetes-dashboard
    namespace: kube-system
```

应用配置：



```
[root@k8s-node1 k8s]# kubectl apply -f helm-rbac.yaml
serviceaccount/tiller created
clusterrolebinding.rbac.authorization.k8s.io/tiller created
[root@k8s-node1 k8s]#
```

2、安装Tiller (Master执行)

1、初始化



```
[root@k8s-node1 k8s]# helm init --service-account=tiller --tiller-image=sapcc/tiller:v2.16.3 --history-max 300
Creating /root/.helm
Creating /root/.helm/repository
Creating /root/.helm/repository/cache
Creating /root/.helm/repository/local
Creating /root/.helm/plugins
Creating /root/.helm/starters
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.
```

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure '[allow unauthenticated users](#)' policy.

To prevent this, run `helm init` with the [--tiller-tls-verify](#) flag.

For more information on securing your installation see:

https://v2.helm.sh/docs/securing_installation/

```
[root@k8s-node1 k8s]#
```

--tiller-image 指定镜像，否则会被墙，等待节点上部署的tiller完成即可。



```
[root@k8s-node1 k8s]# kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
coredns-546565776c-9sbmk          1/1    Running   3          23h
coredns-546565776c-t68mr          1/1    Running   3          23h
etcd-k8s-node1                     1/1    Running   3          23h
kube-apiserver-k8s-node1          1/1    Running   3          23h
kube-controller-manager-k8s-node1  1/1    Running   3          23h
kube-flannel-ds-amd64-5xs5j       1/1    Running   4          22h
kube-flannel-ds-amd64-6xwth       1/1    Running   5          23h
kube-flannel-ds-amd64-fvnvx       1/1    Running   4          22h
```

```
kube-proxy-7tkvl           1/1    Running   3        22h
kube-proxy-mvlnk          1/1    Running   4        22h
kube-proxy-sz2vz          1/1    Running   3        23h
kube-scheduler-k8s-node1  1/1    Running   3        23h
kubernetes-dashboard-975499656-jxczv  0/1    ImagePullBackOff  0        7h45m
tiller-deploy-8cc566858-67bxr  1/1    Running   0        31s
[root@k8s-node1 k8s]#
```

查看集群的所有节点信息：

```
kubectl get node -o wide
```

```
[root@k8s-node1 k8s]# kubectl get node -o wide
NAME      STATUS  ROLES   AGE     VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE
          KERNEL-VERSION             CONTAINER-RUNTIME
k8s-node1  Ready   master   23h    v1.17.3  10.0.2.15    <none>       CentOS Linux 7
(Core)    3.10.0-957.12.2.el7.x86_64  docker://19.3.8
k8s-node2  Ready   <none>   22h    v1.17.3  10.0.2.4    <none>       CentOS Linux 7
(Core)    3.10.0-957.12.2.el7.x86_64  docker://19.3.8
k8s-node3  Ready   <none>   22h    v1.17.3  10.0.2.5    <none>       CentOS Linux 7
(Core)    3.10.0-957.12.2.el7.x86_64  docker://19.3.8
[root@k8s-node1 k8s]#
```

2、测试

```
helm install stable/nginx-ingress --name nginx-ingress
```

最小化安装 KubeSphere

若集群可用 CPU > 1 Core 且可用内存 > 2 G，可以使用以下命令最小化安装 KubeSphere：



```
kubectl apply -f https://raw.githubusercontent.com/kubesphere/ks-installer/master/kubesphere-minimal.yaml
```

提示：若您的服务器提示无法访问 GitHub，可将 [kubesphere-minimal.yaml](#) 或 [kubesphere-complete-setup.yaml](#) 文件保存到本地作为本地的静态文件，再参考上述命令进行安装。

1. 查看滚动刷新的安装日志，请耐心等待安装成功。



```
$ kubectl logs -n kubesphere-system $(kubectl get pod -n kubesphere-system -l app=ks-install -o jsonpath='{.items[0].metadata.name}') -f
```

说明：安装过程中若遇到问题，也可以通过以上日志命令来排查问题。

谷粒商城

接口幂等性



一、什么是幂等性

接口幂等性就是用户对于同一操作发起的一次请求或者多次请求的结果是一致的，不会因为多次点击而产生了副作用；比如说支付场景，用户购买了商品支付扣款成功，但是返回结果的时候网络异常，此时钱已经扣了，用户再次点击按钮，此时会进行第二次扣款，返回结果成功，用户查询余额返发现多扣钱了，流水记录也变成了两条...，这就没有保证接口的幂等性。

二、哪些情况需要防止

用户多次点击按钮

用户页面回退再次提交

微服务互相调用，由于网络问题，导致请求失败。feign 触发重试机制

其他业务情况

三、什么情况下需要幂等

以 SQL 为例，有些操作是天然幂等的。

`SELECT * FROM table WHERE id=?;`，无论执行多少次都不会改变状态，是天然的**幂等**。

`UPDATE tab1 SET col1=1 WHERE col2=2;`，无论执行成功多少次状态都是一致的，也是**幂等**操作。

`delete from user where userid=1;`，多次操作，结果一样，具备**幂等性**

`insert into user(userid,name) values(1,'a');` 如 `userid` 为唯一主键，即重复操作上面的业务，只会插入一条用户数据，具备**幂等性**。

`UPDATE tab1 SET col1=col1+1 WHERE col2=2;`，每次执行的结果都会发生变化，不是**幂等**的。

`insert into user(userid,name) values(1,'a');` 如 `userid` 不是主键，可以重复，那上面业务多次操作，数据都会新增多条，不具备**幂等性**。

四、幂等解决方案

1、token 机制

1、服务端提供了发送 `token` 的接口。我们在分析业务的时候，哪些业务是存在幂等问题的，就必须在执行业务前，先去获取 `token`，服务器会把 `token` 保存到 `redis` 中。

-
- 2、然后调用业务接口请求时，把 token 携带过去，一般放在请求头部。
 - 3、服务器判断 token 是否存在 redis 中，存在表示第一次请求，然后删除 token，继续执行业务。
 - 4、如果判断 token 不存在 redis 中，就表示是重复操作，直接返回重复标记给 client，这样就保证了业务代码，不被重复执行。

危险性：

- 1、先删除 token 还是后删除 token：
 - (1) 先删除可能导致，业务确实没有执行，重试还带上之前 token，由于防重设计导致，请求还是不能执行。
 - (2) 后删除可能导致，业务处理成功，但是服务闪断，出现超时，没有删除 token，别人继续重试，导致业务被执行两边
 - (3) 我们最好设计为先删除 token，如果业务调用失败，就重新获取 token 再次请求。
- 2、Token 获取、比较和删除必须是原子性
 - (1) redis.get(token)、token.equals、redis.del(token)如果这两个操作不是原子，可能导致，高并发下，都 get 到同样的数据，判断都成功，继续业务并发执行
 - (2) 可以在 redis 使用 lua 脚本完成这个操作

```
if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del', KEYS[1]) else return 0 end
```

2、各种锁机制

1、数据库悲观锁

```
select * from xxxx where id = 1 for update;
```

悲观锁使用时一般伴随事务一起使用，数据锁定时间可能会很长，需要根据实际情况选用。另外要注意的是，id 字段一定是主键或者唯一索引，不然可能造成锁表的结果，处理起来会非常麻烦。

2、数据库乐观锁

这种方法适合在更新的场景中，

```
update t_goods set count = count -1 , version = version + 1 where good_id=2 and version = 1
```

根据 version 版本，也就是在操作库存前先获取当前商品的 version 版本号，然后操作的时候带上此 version 号。我们梳理下，我们第一次操作库存时，得到 version 为 1，调用库存服务 version 变成了 2；但返回给订单服务出现了问题，订单服务又一次发起调用库存服务，当订单服务传来的 version 还是 1，再执行上面的 sql 语句时，就不会执行；因为 version 已经变为 2 了，where 条件就不成立。这样就保证了不管调用几次，只会真正的处理一次。

乐观锁主要使用于处理读多写少的问题

3、业务层分布式锁

如果多个机器可能在同一时间同时处理相同的数据，比如多台机器定时任务都拿到了相同数据处理，我们就可以加分布式锁，锁定此数据，处理完成后释放锁。获取到锁的必须先判断这个数据是否被处理过。

3、各种唯一约束

1、数据库唯一约束

插入数据，应该按照唯一索引进行插入，比如订单号，相同的订单就不可能有两条记录插入。我们在数据库层面防止重复。

这个机制是利用了数据库的主键唯一约束的特性，解决了在 `insert` 场景时幂等问题。但主键的要求不是自增的主键，这样就需要业务生成全局唯一的主键。

如果是分库分表场景下，路由规则要保证相同请求下，落地在同一个数据库和同一表中，要不然数据库主键约束就不起效果了，因为是不同的数据库和表主键不相关。

2、redis set 防重

很多数据需要处理，只能被处理一次，比如我们可以计算数据的 MD5 将其放入 redis 的 set，每次处理数据，先看这个 MD5 是否已经存在，存在就不处理。

4、防重表

使用订单号 `orderNo` 做为去重表的唯一索引，把唯一索引插入去重表，再进行业务操作，且他们在同一个事务中。这个保证了重复请求时，因为去重表有唯一约束，导致请求失败，避免了幂等问题。这里要注意的是，去重表和业务表应该在同一库中，这样就保证了在同一个事务，即使业务操作失败了，也会把去重表的数据回滚。这个很好的保证了数据一致性。

之前说的 redis 防重也算

5、全局请求唯一 id

调用接口时，生成一个唯一 id，redis 将数据保存到集合中（去重），存在即处理过。

可以使用 nginx 设置每一个请求的唯一 id；

```
proxy_set_header X-Request-Id $request_id;
```



让天下没有难学的技术



Thymeleaf



Tutorial: Using Thymeleaf

Document version: 20181029 - 29 October 2018

Project version: 3.0.11.RELEASE

Project web site: <https://www.thymeleaf.org>

1 Introducing Thymeleaf

1.1 What is Thymeleaf?

Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text.

The main goal of Thymeleaf is to provide an elegant and highly-maintainable way of creating templates. To achieve this, it builds on the concept of *Natural Templates* to inject its logic into template files in a way that doesn't affect the template from being used as a design prototype. This improves communication of design and bridges the gap between design and development teams.

Thymeleaf has also been designed from the beginning with Web Standards in mind – especially **HTML5** – allowing you to create fully validating templates if that is a need for you.

1.2 What kind of templates can Thymeleaf process?

Out-of-the-box, Thymeleaf allows you to process six kinds of templates, each of which is called a **Template Mode**:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

There are two *markup* template modes (`HTML` and `XML`), three *textual* template modes (`TEXT`, `JAVASCRIPT` and `CSS`) and a *no-op* template mode (`RAW`).

The `HTML` template mode will allow any kind of HTML input, including HTML5, HTML 4 and XHTML. No validation or well-formedness check will be performed, and template code/structure will be respected to the biggest possible extent in output.

The `XML` template mode will allow XML input. In this case, code is expected to be well-formed – no unclosed tags, no unquoted attributes, etc – and the parser will throw exceptions if well-formedness violations are found. Note that no *validation* (against a DTD or XML Schema) will be performed.

The `TEXT` template mode will allow the use of a special syntax for templates of a non-markup nature. Examples of such templates might be text emails or templated documentation. Note that HTML or XML templates can be also processed as `TEXT`, in which case they will not be parsed as markup, and every tag, DOCTYPE, comment, etc, will be treated as mere text.

The `JAVASCRIPT` template mode will allow the processing of JavaScript files in a Thymeleaf application. This means being able to use model data inside JavaScript files in the same way it can be done in HTML files, but with JavaScript-specific integrations such as specialized escaping or *natural scripting*. The `JAVASCRIPT` template mode is considered a *textual* mode and therefore uses the same special syntax as the `TEXT` template mode.

The `CSS` template mode will allow the processing of CSS files involved in a Thymeleaf application. Similar to the `JAVASCRIPT` mode, the `CSS` template mode is also a *textual* mode and uses the special processing syntax from the `TEXT` template mode.

The `RAW` template mode will simply not process templates at all. It is meant to be used for inserting untouched resources (files, URL responses, etc.) into the templates being processed. For example, external, uncontrolled

resources in HTML format could be included into application templates, safely knowing that any Thymeleaf code that these resources might include will not be executed.

1.3 Dialects: The Standard Dialect

Thymeleaf is an extremely extensible template engine (in fact it could be called a *template engine framework*) that allows you to define and customize the way your templates will be processed to a fine level of detail.

An object that applies some logic to a markup artifact (a tag, some text, a comment, or a mere placeholder if templates are not markup) is called a *processor*, and a set of these processors – plus perhaps some extra artifacts – is what a dialect is normally comprised of. Out of the box, Thymeleaf's core library provides a dialect called the **Standard Dialect**, which should be enough for most users.

Note that dialects can actually have no processors and be entirely comprised of other kinds of artifacts, but processors are definitely the most common use case.

This tutorial covers the **Standard Dialect**. Every attribute and syntax feature you will learn about in the following pages is defined by this dialect, even if that isn't explicitly mentioned.

Of course, users can create their own dialects (even extending the Standard one) if they want to define their own processing logic while taking advantage of the library's advanced features. Thymeleaf can also be configured to use several dialects at a time.

The official thymeleaf-spring3 and thymeleaf-spring4 integration packages both define a dialect called the "SpringStandard Dialect", which is mostly the same as the Standard Dialect, but with small adaptations to make better use of some features in the Spring Framework (for example, by using Spring Expression Language or SpringEL instead of OGNL). So if you are a Spring MVC user you are not wasting your time, as almost everything you learn here will be of use in your Spring applications.

Most of the processors of the Standard Dialect are *attribute processors*. This allows browsers to correctly display HTML template files even before being processed because they will simply ignore the additional attributes. For example, while a JSP using tag libraries could include a fragment of code not directly displayable by a browser like:

```
<form:inputText name="userName" value="${user.name}" />
```

...the Thymeleaf Standard Dialect would allow us to achieve the same functionality with:

```
<input type="text" name="userName" value="James Carrot" th:value="${user.name}" />
```

Not only will this be correctly displayed by browsers, but this also allow us to (optionally) specify a value attribute in it ("James Carrot", in this case) that will be displayed when the prototype is statically opened in a browser, and that will be substituted by the value resulting from the evaluation of \${user.name} during processing of the template.

This helps your designer and developer to work on the very same template file and reduce the effort required to transform a static prototype into a working template file. The ability to do this is a feature called *Natural Templating*.

2 The Good Thymes Virtual Grocery

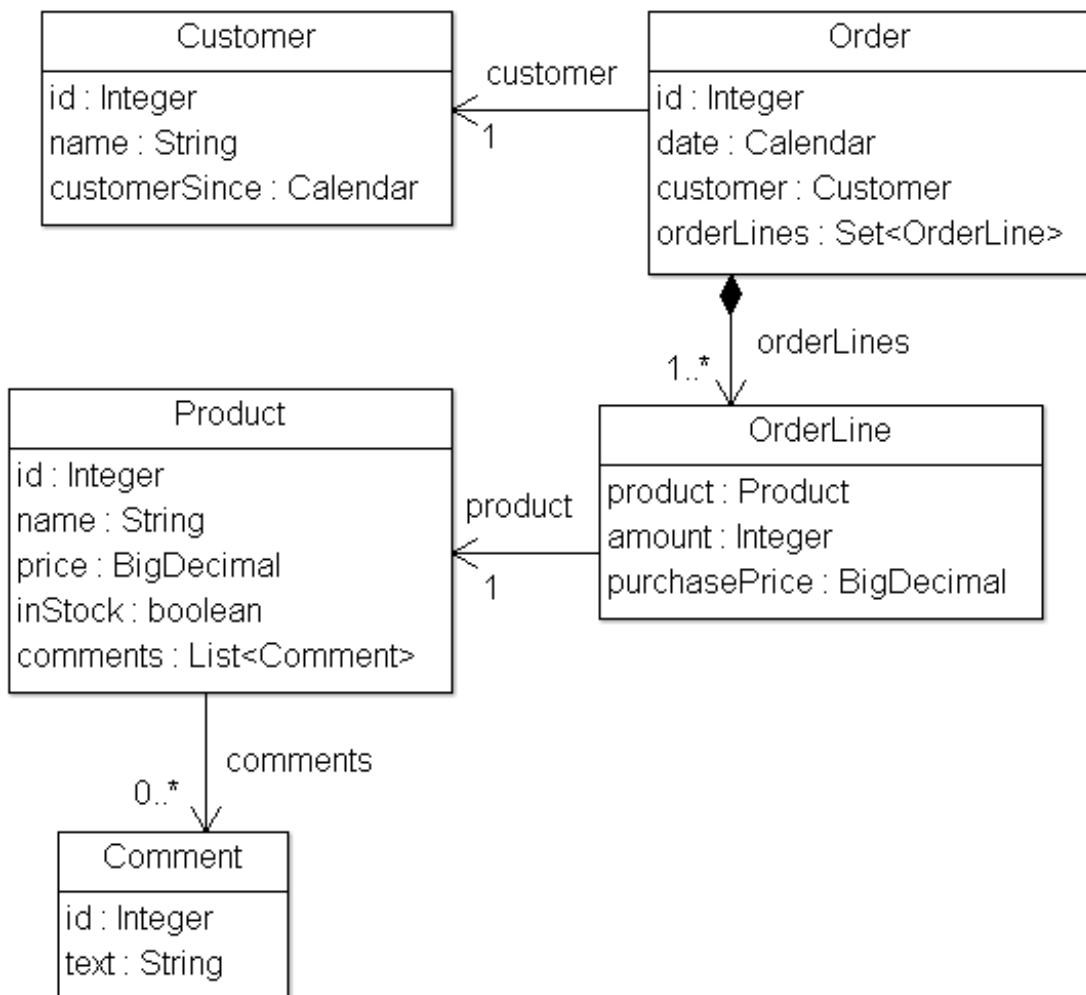
The source code for the examples shown in this, and future chapters of this guide, can be found in the [Good Thymes Virtual Grocery GitHub repository](#).

2.1 A website for a grocery

To better explain the concepts involved in processing templates with Thymeleaf, this tutorial will use a demo application which you can download from the project's web site.

This application is the web site of an imaginary virtual grocery, and will provide us with many scenarios to showcase Thymeleaf's many features.

To start, we need a simple set of model entities for our application: Products which are sold to Customers by creating Orders . We will also be managing Comments about those Products :



Example application model

Our application will also have a very simple service layer, composed by `Service` objects containing methods like:

```
public class ProductService {  
    ...  
  
    public List<Product> findAll() {  
        return ProductRepository.getInstance().findAll();  
    }  
  
    public Product findById(Integer id) {  
        return ProductRepository.getInstance().findById(id);  
    }  
}
```

At the web layer our application will have a filter that will delegate execution to Thymeleaf-enabled commands depending on the request URL:

```

private boolean process(HttpServletRequest request, HttpServletResponse response)
    throws ServletException {

    try {

        // This prevents triggering engine executions for resource URLs
        if (request.getRequestURI().startsWith("/css") ||
            request.getRequestURI().startsWith("/images") ||
            request.getRequestURI().startsWith("/favicon")) {
            return false;
        }

        /*
         * Query controller/URL mapping and obtain the controller
         * that will process the request. If no controller is available,
         * return false and let other filters/servlets process the request.
         */
        IGTVGController controller = this.application.resolveControllerForRequest(request);
        if (controller == null) {
            return false;
        }

        /*
         * Obtain the TemplateEngine instance.
         */
        ITemplateEngine templateEngine = this.application.getTemplateEngine();

        /*
         * Write the response headers
         */
        response.setContentType("text/html;charset=UTF-8");
        response.setHeader("Pragma", "no-cache");
        response.setHeader("Cache-Control", "no-cache");
        response.setDateHeader("Expires", 0);

        /*
         * Execute the controller and process view template,
         * writing the results to the response writer.
         */
        controller.process(
            request, response, this.servletContext, templateEngine);

        return true;
    } catch (Exception e) {
        try {
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        } catch (final IOException ignored) {
            // Just ignore this
        }
        throw new ServletException(e);
    }
}

```

This is our `IGTVGController` interface:

```
public interface IGTVGController {  
  
    public void process(  
        HttpServletRequest request, HttpServletResponse response,  
        ServletContext servletContext, ITemplateEngine templateEngine);  
  
}
```

All we have to do now is create implementations of the `IGTVGController` interface, retrieving data from the services and processing templates using the `ITemplateEngine` object.

In the end, it will look like this:



Example application home page

But first let's see how that template engine is initialized.

2.2 Creating and configuring the Template Engine

The `process(...)` method in our filter contained this line:

```
ITemplateEngine templateEngine = this.application.getTemplateEngine();
```

Which means that the `GTVGApplication` class is in charge of creating and configuring one of the most important objects in a Thymeleaf application: the `TemplateEngine` instance (implementation of the `ITemplateEngine` interface).

Our `org.thymeleaf.TemplateEngine` object is initialized like this:

```

public class GTVGApplication {

    ...
    private final TemplateEngine templateEngine;
    ...

    public GTVGApplication(final ServletContext servletContext) {
        super();

        ServletContextTemplateResolver templateResolver =
            new ServletContextTemplateResolver(servletContext);

        // HTML is the default mode, but we set it anyway for better understanding of code
        templateResolver.setTemplateMode(TemplateMode.HTML);
        // This will convert "home" to "/WEB-INF/templates/home.html"
        templateResolver.setPrefix("/WEB-INF/templates/");
        templateResolver.setSuffix(".html");
        // Template cache TTL=1h. If not set, entries would be cached until expelled
        templateResolver.setCacheTTLMs(Long.valueOf(3600000L));

        // Cache is set to true by default. Set to false if you want templates to
        // be automatically updated when modified.
        templateResolver.setCacheable(true);

        this.templateEngine = new TemplateEngine();
        this.templateEngine.setTemplateResolver(templateResolver);

        ...
    }
}

```

There are many ways of configuring a `TemplateEngine` object, but for now these few lines of code will teach us enough about the steps needed.

The Template Resolver

Let's start with the Template Resolver:

```

ServletContextTemplateResolver templateResolver =
    new ServletContextTemplateResolver(servletContext);

```

Template Resolvers are objects that implement an interface from the Thymeleaf API called `org.thymeleaf.templateresolver.ITemplateResolver`:

```

public interface ITemplateResolver {

    ...

    /*
     * Templates are resolved by their name (or content) and also (optionally) their
     * owner template in case we are trying to resolve a fragment for another template.
     * Will return null if template cannot be handled by this template resolver.
     */
    public TemplateResolution resolveTemplate(
        final IEngineConfiguration configuration,
        final String ownerTemplate, final String template,
        final Map<String, Object> templateResolutionAttributes);
}

```

These objects are in charge of determining how our templates will be accessed, and in this GTVG application, the `org.thymeleaf.templateresolver.ServletContextTemplateResolver` means that we are going to retrieve our template files as resources from the *ServletContext*: an application-wide `javax.servlet.ServletContext` object that exists in every Java web application, and that resolves resources from the web application root.

But that's not all we can say about the template resolver, because we can set some configuration parameters on it. First, the template mode:

```
templateResolver.setTemplateMode(TemplateMode.HTML);
```

HTML is the default template mode for `ServletContextTemplateResolver`, but it is good practice to establish it anyway so that our code documents clearly what is going on.

```
templateResolver.setPrefix("/WEB-INF/templates/");
templateResolver.setSuffix(".html");
```

The *prefix* and *suffix* modify the template names that we will be passing to the engine for obtaining the real resource names to be used.

Using this configuration, the template name "*product/list*" would correspond to:

```
servletContext.getResourceAsStream("/WEB-INF/templates/product/list.html")
```

Optionally, the amount of time that a parsed template can live in the cache is configured at the Template Resolver by means of the `cacheTTLMs` property:

```
templateResolver.setCacheTTLMs(3600000L);
```

A template can still be expelled from cache before that TTL is reached if the max cache size is reached and it is the oldest entry currently cached.

Cache behaviour and sizes can be defined by the user by implementing the `ICacheManager` interface or by modifying the `StandardCacheManager` object to manage the default cache.

There is much more to learn about template resolvers, but for now let's have a look at the creation of our Template Engine object.

The Template Engine

Template Engine objects are implementations of the `org.thymeleaf.ITemplateEngine` interface. One of these implementations is offered by the Thymeleaf core: `org.thymeleaf.TemplateEngine`, and we create an instance of it here:

```
templateEngine = new TemplateEngine();
templateEngine.setTemplateResolver(templateResolver);
```

Rather simple, isn't it? All we need is to create an instance and set the Template Resolver to it.

A template resolver is the only *required* parameter a `TemplateEngine` needs, although there are many others that will be covered later (message resolvers, cache sizes, etc). For now, this is all we need.

Our Template Engine is now ready and we can start creating our pages using Thymeleaf.

3 Using Texts

3.1 A multi-language welcome

Our first task will be to create a home page for our grocery site.

The first version of this page will be extremely simple: just a title and a welcome message. This is our `/WEB-INF/templates/home.html` file:

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

    <head>
        <title>Good Thymes Virtual Grocery</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link rel="stylesheet" type="text/css" media="all"
              href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />
    </head>

    <body>
        <p th:text="#{home.welcome}">Welcome to our grocery store!</p>
    </body>
</html>
```

The first thing you will notice is that this file is HTML5 that can be correctly displayed by any browser because it does not include any non-HTML tags (browsers ignore all attributes they don't understand, like `th:text`).

But you may also notice that this template is not really a *valid* HTML5 document, because these non-standard attributes we are using in the `th:*` form are not allowed by the HTML5 specification. In fact, we are even adding an `xmlns:th` attribute to our `<html>` tag, something absolutely non-HTML5-ish:

```
<html xmlns:th="http://www.thymeleaf.org">
```

...which has no influence at all in template processing, but works as an *incantation* that prevents our IDE from complaining about the lack of a namespace definition for all those `th:*` attributes.

So what if we wanted to make this template **HTML5-valid**? Easy: switch to Thymeleaf's data attribute syntax, using the `data-` prefix for attribute names and hyphen (-) separators instead of semi-colons (:):

```

<!DOCTYPE html>

<html>

<head>
    <title>Good Thymes Virtual Grocery</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" type="text/css" media="all"
          href="../../css/gtvg.css" data-th-href="@{/css/gtvg.css}" />
</head>

<body>

    <p data-th-text="#{home.welcome}">Welcome to our grocery store!</p>

</body>

</html>

```

Custom `data-` prefixed attributes are allowed by the HTML5 specification, so, with this code above, our template would be a *valid HTML5 document*.

Both notations are completely equivalent and interchangeable, but for the sake of simplicity and compactness of the code samples, this tutorial will use the *namespace notation* (`th:*`). Also, the `th:*` notation is more general and allowed in every Thymeleaf template mode (`XML`, `TEXT` ...) whereas the `data-` notation is only allowed in `HTML` mode.

Using `th:text` and externalizing text

Externalizing text is extracting fragments of template code out of template files so that they can be kept in separate files (typically `.properties` files) and that they can be easily replaced with equivalent texts written in other languages (a process called **internationalization** or simply *i18n*). Externalized fragments of text are usually called “*messages*”.

Messages always have a key that identifies them, and Thymeleaf allows you to specify that a text should correspond to a specific message with the `#{...}` syntax:

```
<p th:text="#{home.welcome}">Welcome to our grocery store!</p>
```

What we can see here are in fact two different features of the Thymeleaf Standard Dialect:

- The `th:text` attribute, which evaluates its value expression and sets the result as the body of the host tag, effectively replacing the “Welcome to our grocery store!” text we see in the code.
- The `#{home.welcome}` expression, specified in the *Standard Expression Syntax*, instructing that the text to be used by the `th:text` attribute should be the message with the `home.welcome` key corresponding to whichever locale we are processing the template with.

Now, where is this externalized text?

The location of externalized text in Thymeleaf is fully configurable, and it will depend on the specific `org.thymeleaf.messageresolver.IMessageResolver` implementation being used. Normally, an implementation based on `.properties` files will be used, but we could create our own implementations if we wanted, for example, to obtain messages from a database.

However, we have not specified a message resolver for our template engine during initialization, and that means that our application is using the *Standard Message Resolver*, implemented by `org.thymeleaf.messageresolver.StandardMessageResolver`.

The standard message resolver expects to find messages for `/WEB-INF/templates/home.html` in properties files in the same folder and with the same name as the template, like:

- `/WEB-INF/templates/home_en.properties` for English texts.
- `/WEB-INF/templates/home_es.properties` for Spanish language texts.
- `/WEB-INF/templates/home_pt_BR.properties` for Portuguese (Brazil) language texts.
- `/WEB-INF/templates/home.properties` for default texts (if the locale is not matched).

Let's have a look at our `home_es.properties` file:

```
home.welcome=¡Bienvenido a nuestra tienda de comestibles!
```

This is all we need for making Thymeleaf process our template. Let's create our Home controller then.

Contexts

In order to process our template, we will create a `HomeController` class implementing the `IGTVGController` interface we saw before:

```
public class HomeController implements ITVGController {  
  
    public void process(  
        final HttpServletRequest request, final HttpServletResponse response,  
        final ServletContext servletContext, final ITemplateEngine templateEngine)  
        throws Exception {  
  
        WebContext ctx =  
            new WebContext(request, response, servletContext, request.getLocale());  
  
        templateEngine.process("home", ctx, response.getWriter());  
  
    }  
}
```

The first thing we see is the creation of a *context*. A Thymeleaf context is an object implementing the `org.thymeleaf.context.IContext` interface. Contexts should contain all the data required for an execution of the template engine in a variables map, and also reference the locale that must be used for externalized messages.

```
public interface IContext {  
  
    public Locale getLocale();  
    public boolean containsVariable(final String name);  
    public Set<String> getVariableNames();  
    public Object getVariable(final String name);  
  
}
```

There is a specialized extension of this interface, `org.thymeleaf.context.IWebContext`, meant to be used in ServletAPI-based web applications (like SpringMVC).

```

public interface IWebContext extends IContext {

    public HttpServletRequest getRequest();
    public HttpServletResponse getResponse();
    public HttpSession getSession();
    public ServletContext getServletContext();

}

```

The Thymeleaf core library offers an implementation of each of these interfaces:

- `org.thymeleaf.context.Context` implements `IContext`
- `org.thymeleaf.context.WebContext` implements `IWebContext`

And as you can see in the controller code, `WebContext` is the one we use. In fact we have to, because the use of a `ServletContextTemplateResolver` requires that we use a context implementing `IWebContext`.

```

WebContext ctx = new WebContext(request, response, servletContext, request.getLocale());

```

Only three out of those four constructor arguments are required because the default locale for the system will be used if none is specified (although you should never let this happen in real applications).

There are some specialized expressions that we will be able to use to obtain the request parameters and the request, session and application attributes from the `WebContext` in our templates. For example:

- `${x}` will return a variable `x` stored into the Thymeleaf context or as a *request attribute*.
- `${param.x}` will return a *request parameter* called `x` (which might be multivalued).
- `${session.x}` will return a *session attribute* called `x`.
- `${application.x}` will return a *servlet context attribute* called `x`.

Executing the template engine

With our context object ready, now we can tell the template engine to process the template (by its name) using the context, and passing it a response writer so that the response can be written to it:

```

templateEngine.process("home", ctx, response.getWriter());

```

Let's see the results of this using the Spanish locale:

```

<!DOCTYPE html>

<html>

    <head>
        <title>Good Thymes Virtual Grocery</title>
        <meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>
        <link rel="stylesheet" type="text/css" media="all" href="/gtvg/css/gtvg.css" />
    </head>

    <body>

        <p>¡Bienvenido a nuestra tienda de comestibles!</p>

    </body>

</html>

```

3.2 More on texts and variables

Unescaped Text

The simplest version of our Home page seems to be ready now, but there is something we have not thought about... what if we had a message like this?

```
home.welcome=Welcome to our <b>fantastic</b> grocery store!
```

If we execute this template like before, we will obtain:

```
<p>Welcome to our &lt;b&gt;fantastic&lt;/b&gt; grocery store!</p>
```

Which is not exactly what we expected, because our `` tag has been escaped and therefore it will be displayed in the browser.

This is the default behaviour of the `th:text` attribute. If we want Thymeleaf to respect our HTML tags and not escape them, we will have to use a different attribute: `th:utext` (for “unescaped text”):

```
<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>
```

This will output our message just like we wanted it:

```
<p>Welcome to our <b>fantastic</b> grocery store!</p>
```

Using and displaying variables

Now let's add some more content to our home page. For example, we may want to display the date below our welcome message, like this:

```
Welcome to our fantastic grocery store!
```

```
Today is: 12 july 2010
```

First of all, we will have to modify our controller so that we add that date as a context variable:

```
public void process(
    final HttpServletRequest request, final HttpServletResponse response,
    final ServletContext servletContext, final ITemplateEngine templateEngine)
    throws Exception {

    SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMMM yyyy");
    Calendar cal = Calendar.getInstance();

    WebContext ctx =
        new WebContext(request, response, servletContext, request.getLocale());
    ctx.setVariable("today", dateFormat.format(cal.getTime()));

    templateEngine.process("home", ctx, response.getWriter());
}
```

We have added a `String` variable called `today` to our context, and now we can display it in our template:

```
<body>

    <p th:utext="#{home.welcome}">Welcome to our grocery store!</p>

    <p>Today is: <span th:text="${today}">13 February 2011</span></p>

</body>
```

As you can see, we are still using the `th:text` attribute for the job (and that's correct, because we want to replace the tag's body), but the syntax is a little bit different this time and instead of a `#{...}` expression value, we are using a `${...}` one. This is a **variable expression**, and it contains an expression in a language called *OGNL (Object-Graph Navigation Language)* that will be executed on the context variables map we talked about before.

The `${today}` expression simply means "get the variable called `today`", but these expressions could be more complex (like `${user.name}` for "get the variable called `user`, and call its `getName()` method").

There are quite a lot of possibilities in attribute values: messages, variable expressions... and quite a lot more. The next chapter will show us what all these possibilities are.

4 Standard Expression Syntax

We will take a small break in the development of our grocery virtual store to learn about one of the most important parts of the Thymeleaf Standard Dialect: the Thymeleaf Standard Expression syntax.

We have already seen two types of valid attribute values expressed in this syntax: message and variable expressions:

```
<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>  
<p>Today is: <span th:text="${today}">13 february 2011</span></p>
```

But there are more types of expressions, and more interesting details to learn about the ones we already know. First, let's see a quick summary of the Standard Expression features:

- Simple expressions:
 - Variable Expressions: \${...}
 - Selection Variable Expressions: *{...}
 - Message Expressions: #{...}
 - Link URL Expressions: @{...}
 - Fragment Expressions: ~{...}
- Literals
 - Text literals: 'one text', 'Another one!', ...
 - Number literals: 0, 34, 3.0, 12.3, ...
 - Boolean literals: true, false
 - Null literal: null
 - Literal tokens: one, sometext, main, ...
- Text operations:
 - String concatenation: +
 - Literal substitutions: |The name is \${name}|
- Arithmetic operations:
 - Binary operators: +, -, *, /, %
 - Minus sign (unary operator): -
- Boolean operations:
 - Binary operators: and, or
 - Boolean negation (unary operator): !, not
- Comparisons and equality:
 - Comparators: >, <, >=, <= (gt, lt, ge, le)
 - Equality operators: ==, != (eq, ne)
- Conditional operators:
 - If-then: (if) ? (then)
 - If-then-else: (if) ? (then) : (else)
 - Default: (value) ?: (defaultvalue)
- Special tokens:

- No-Operation: _

All these features can be combined and nested:

```
'User is of type ' + (${user.isAdmin()} ? 'Administrator' : (${user.type} ?: 'Unknown'))
```

4.1 Messages

As we already know, `#{...}` message expressions allow us to link this:

```
<p th:utext="#{home.welcome}">Welcome to our grocery store!</p>
```

...to this:

```
home.welcome=;Bienvenido a nuestra tienda de comestibles!
```

But there's one aspect we still haven't thought of: what happens if the message text is not completely static? What if, for example, our application knew who is the user visiting the site at any moment and we wanted to greet them by name?

```
<p>¡Bienvenido a nuestra tienda de comestibles, John Apricot!</p>
```

This means we would need to add a parameter to our message. Just like this:

```
home.welcome=;Bienvenido a nuestra tienda de comestibles, {0}!
```

Parameters are specified according to the `java.text.MessageFormat` standard syntax, which means you can format to numbers and dates as specified in the API docs for classes in the `java.text.*` package.

In order to specify a value for our parameter, and given an HTTP session attribute called `user`, we could have:

```
<p th:utext="#{home.welcome(${session.user.name})}">
    Welcome to our grocery store, Sebastian Pepper!
</p>
```

Note that the use of `th:utext` here means that the formatted message will not be escaped. This example assumes that `user.name` is already escaped.

Several parameters can be specified, separated by commas.

The message key itself can come from a variable:

```
<p th:utext="#{$welcomeMsgKey}(${session.user.name})">
    Welcome to our grocery store, Sebastian Pepper!
</p>
```

4.2 Variables

We already mentioned that `${...}` expressions are in fact OGNL (Object-Graph Navigation Language) expressions executed on the map of variables contained in the context.

For detailed info about OGNL syntax and features, you should read the [OGNL Language Guide](#)

In Spring MVC-enabled applications OGNL will be replaced with `SpringEL`, but its syntax is very similar to that of OGNL (actually, exactly the same for most common cases).

From OGNL's syntax, we know that the expression in:

```
<p>Today is: <span th:text="${today}">13 february 2011</span>.</p>
```

...is in fact equivalent to this:

```
ctx.getVariable("today");
```

But OGNL allows us to create quite more powerful expressions, and that's how this:

```
<p th:utext="#{home.welcome(${session.user.name})}">  
    Welcome to our grocery store, Sebastian Pepper!  
</p>
```

...obtains the user name by executing:

```
((User) ctx.getVariable("session").get("user")).getName();
```

But getter method navigation is just one of OGNL's features. Let's see some more:

```

/*
 * Access to properties using the point (.). Equivalent to calling property getters.
 */
${person.father.name}

/*
 * Access to properties can also be made by using brackets ([] ) and writing
 * the name of the property as a variable or between single quotes.
 */
${person['father']['name']}

/*
 * If the object is a map, both dot and bracket syntax will be equivalent to
 * executing a call on its get(...) method.
 */
${countriesByCode.ES}
${personsByName['Stephen Zucchini'].age}

/*
 * Indexed access to arrays or collections is also performed with brackets,
 * writing the index without quotes.
 */
${personsArray[0].name}

/*
 * Methods can be called, even with arguments.
 */
${person.createCompleteName()}
${person.createCompleteNameWithSeparator('-')}

```

Expression Basic Objects

When evaluating OGNL expressions on the context variables, some objects are made available to expressions for higher flexibility. These objects will be referenced (per OGNL standard) starting with the # symbol:

- #ctx : the context object.
- #vars: the context variables.
- #locale : the context locale.
- #request : (only in Web Contexts) the HttpServletRequest object.
- #response : (only in Web Contexts) the HttpServletResponse object.
- #session : (only in Web Contexts) the HttpSession object.
- #servletContext : (only in Web Contexts) the ServletContext object.

So we can do this:

```
Established locale country: <span th:text="#{#locale.country}">US</span>.
```

You can read the full reference of these objects in [Appendix A](#).

Expression Utility Objects

Besides these basic objects, Thymeleaf will offer us a set of utility objects that will help us perform common tasks in our expressions.

- #execInfo : information about the template being processed.
- #messages : methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{} syntax.
- #uris : methods for escaping parts of URLs/URIs

- `#conversions` : methods for executing the configured *conversion service* (if any).
- `#dates` : methods for `java.util.Date` objects: formatting, component extraction, etc.
- `#calendars` : analogous to `#dates`, but for `java.util.Calendar` objects.
- `#numbers` : methods for formatting numeric objects.
- `#strings` : methods for `String` objects: contains, startsWith, prepending/appending, etc.
- `#objects` : methods for objects in general.
- `#bools` : methods for boolean evaluation.
- `#arrays` : methods for arrays.
- `#lists` : methods for lists.
- `#sets` : methods for sets.
- `#maps` : methods for maps.
- `#aggregates` : methods for creating aggregates on arrays or collections.
- `#ids` : methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).

You can check what functions are offered by each of these utility objects in the [Appendix B](#).

Reformatting dates in our home page

Now we know about these utility objects, we could use them to change the way in which we show the date in our home page. Instead of doing this in our `HomeController`:

```
SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMMM yyyy");
Calendar cal = Calendar.getInstance();

WebContext ctx = new WebContext(request, servletContext, request.getLocale());
ctx.setVariable("today", dateFormat.format(cal.getTime()));

templateEngine.process("home", ctx, response.getWriter());
```

...we can do just this:

```
WebContext ctx =
    new WebContext(request, response, servletContext, request.getLocale());
ctx.setVariable("today", Calendar.getInstance());

templateEngine.process("home", ctx, response.getWriter());
```

...and then perform date formatting in the view layer itself:

```
<p>
    Today is: <span th:text="#{calendars.format(today, 'dd MMMM yyyy')}>13 May 2011</span>
</p>
```

4.3 Expressions on selections (asterisk syntax)

Not only can variable expressions be written as `${...}`, but also as `*{...}`.

There is an important difference though: the asterisk syntax evaluates expressions on *selected objects* rather than on the whole context. That is, as long as there is no selected object, the dollar and the asterisk syntaxes do exactly the same.

And what is a selected object? The result of an expression using the `th:object` attribute. Let's use one in our user profile (`userprofile.html`) page:

```

<div th:object="${session.user}">
    <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
    <p>Surname: <span th:text="*{lastName}">Pepper</span>.</p>
    <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>

```

Which is exactly equivalent to:

```

<div>
    <p>Name: <span th:text="${session.user.firstName}">Sebastian</span>.</p>
    <p>Surname: <span th:text="${session.user.lastName}">Pepper</span>.</p>
    <p>Nationality: <span th:text="${session.user.nationality}">Saturn</span>.</p>
</div>

```

Of course, dollar and asterisk syntax can be mixed:

```

<div th:object="${session.user}">
    <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
    <p>Surname: <span th:text="${session.user.lastName}">Pepper</span>.</p>
    <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>

```

When an object selection is in place, the selected object will also be available to dollar expressions as the `#object` expression variable:

```

<div th:object="${session.user}">
    <p>Name: <span th:text="${#object.firstName}">Sebastian</span>.</p>
    <p>Surname: <span th:text="${session.user.lastName}">Pepper</span>.</p>
    <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>

```

As said, if no object selection has been performed, dollar and asterisk syntaxes are equivalent.

```

<div>
    <p>Name: <span th:text="*{session.user.name}">Sebastian</span>.</p>
    <p>Surname: <span th:text="*{session.user.surname}">Pepper</span>.</p>
    <p>Nationality: <span th:text="*{session.user.nationality}">Saturn</span>.</p>
</div>

```

4.4 Link URLs

Because of their importance, URLs are first-class citizens in web application templates, and the *Thymeleaf Standard Dialect* has a special syntax for them, the `@` syntax: `@{...}`

There are different types of URLs:

- Absolute URLs: `http://www.thymeleaf.org`
- Relative URLs, which can be:
 - Page-relative: `user/login.html`
 - Context-relative: `/itemdetails?id=3` (context name in server will be added automatically)
 - Server-relative: `~/billing/processInvoice` (allows calling URLs in another context (= application) in the same server.)
 - Protocol-relative URLs: `//code.jquery.com/jquery-2.0.3.min.js`

The real processing of these expressions and their conversion to the URLs that will be output is done by implementations of the `org.thymeleaf.linkbuilder.ILinkBuilder` interface that are registered into the `ITemplateEngine` object being used.

By default, a single implementation of this interface is registered of the class `org.thymeleaf.linkbuilder.StandardLinkBuilder`, which is enough for both offline (non-web) and also web scenarios based on the Servlet API. Other scenarios (like integration with non-ServletAPI web frameworks) might need specific implementations of the link builder interface.

Let's use this new syntax. Meet the `th:href` attribute:

```
<!-- Will produce 'http://localhost:8080/gtvg/order/details?orderId=3' (plus rewriting) -->
<a href="details.html"
    th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/details?orderId=3' (plus rewriting) -->
<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/3/details' (plus rewriting) -->
<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>
```

Some things to note here:

- `th:href` is a modifier attribute: once processed, it will compute the link URL to be used and set that value to the `href` attribute of the `<a>` tag.
- We are allowed to use expressions for URL parameters (as you can see in `orderId=${o.id}`). The required URL-parameter-encoding operations will also be automatically performed.
- If several parameters are needed, these will be separated by commas:
`@{/order/process(execId=${execId},execType='FAST')}`
- Variable templates are also allowed in URL paths: `@{/order/{orderId}/details(orderId=${orderId})}`
- Relative URLs starting with `/` (eg: `/order/details`) will be automatically prefixed by the application context name.
- If cookies are not enabled or this is not yet known, a `";jsessionid=..."` suffix might be added to relative URLs so that the session is preserved. This is called *URL Rewriting* and Thymeleaf allows you to plug in your own rewriting filters by using the `response.encodeURL(...)` mechanism from the Servlet API for every URL.
- The `th:href` attribute allows us to (optionally) have a working static `href` attribute in our template, so that our template links remained navigable by a browser when opened directly for prototyping purposes.

As was the case with the message syntax (`# {...}`), URL bases can also be the result of evaluating another expression:

```
<a th:href="@{$url}(orderId=${o.id})">view</a>
<a th:href="@{'/details/' + ${user.login}}(orderId=${o.id})">view</a>
```

A menu for our home page

Now that we know how to create link URLs, what about adding a small menu in our home page for some of the other pages in the site?

```
<p>Please select an option</p>
<ol>
    <li><a href="product/list.html" th:href="@{/product/list}">Product List</a></li>
    <li><a href="order/list.html" th:href="@{/order/list}">Order List</a></li>
    <li><a href="subscribe.html" th:href="@{/subscribe}">Subscribe to our Newsletter</a></li>
    <li><a href="userprofile.html" th:href="@{/userprofile}">See User Profile</a></li>
</ol>
```

Server root relative URLs

An additional syntax can be used to create server-root-relative (instead of context-root-relative) URLs in order to link to different contexts in the same server. These URLs will be specified like `@{~/path/to/something}`

4.5 Fragments

Fragment expressions are an easy way to represent fragments of markup and move them around templates. This allows us to replicate them, pass them to other templates as arguments, and so on.

The most common use is for fragment insertion using `th:insert` or `th:replace` (more on these in a later section):

```
<div th:insert="~{commons :: main}">...</div>
```

But they can be used anywhere, just as any other variable:

```
<div th:with="frag=~{footer :: #main/text()}">
  <p th:insert="${frag}">
</div>
```

Later in this tutorial there is an entire section devoted to Template Layout, including deeper explanation of fragment expressions.

4.6 Literals

Text literals are just character strings specified between single quotes. They can include any character, but you should escape any single quotes inside them using `\'`.

```
<p>
  Now you are looking at a <span th:text="'"working web application'">template file</span>.
</p>
```

Number literals

Numeric literals are just that: numbers.

```
<p>The year is <span th:text="2013">1492</span>.</p>
<p>In two years, it will be <span th:text="2013 + 2">1494</span>.</p>
```

Boolean literals

The boolean literals are `true` and `false`. For example:

```
<div th:if="${user.isAdmin()} == false"> ...
```

In this example, the `== false` is written outside the braces, and so it is Thymeleaf that takes care of it. If it were written inside the braces, it would be the responsibility of the OGNL/SpringEL engines:

```
<div th:if="${user.isAdmin() == false}"> ...
```

The null literal

The `null` literal can be also used:

```
<div th:if="${variable.something} == null"> ...
```

Literal tokens

Numeric, boolean and null literals are in fact a particular case of *literal tokens*.

These tokens allow a little bit of simplification in Standard Expressions. They work exactly the same as text literals (`'...'`), but they only allow letters (`A-Z` and `a-z`), numbers (`0-9`), brackets (`[` and `]`), dots (`.`), hyphens (`-`) and underscores (`_`). So no whitespaces, no commas, etc.

The nice part? Tokens don't need any quotes surrounding them. So we can do this:

```
<div th:class="content">...</div>
```

instead of:

```
<div th:class="'content'">...</div>
```

4.7 Appending texts

Texts, no matter whether they are literals or the result of evaluating variable or message expressions, can be easily appended using the `+` operator:

```
<span th:text="The name of the user is ' + ${user.name}">
```

4.8 Literal substitutions

Literal substitutions allow for an easy formatting of strings containing values from variables without the need to append literals with `'...' + '...'.`

These substitutions must be surrounded by vertical bars (`|`), like:

```
<span th:text="|Welcome to our application, ${user.name}!|">
```

Which is equivalent to:

```
<span th:text="Welcome to our application, ' + ${user.name} + '!">
```

Literal substitutions can be combined with other types of expressions:

```
<span th:text="${onevar} + ' ' + ${twovar}, ${threevar}">
```

Only variable/message expressions (`${...}`, `*{...}`, `# {...}`) are allowed inside `| ... |` literal substitutions. No other literals (`' ... '`), boolean/numeric tokens, conditional expressions etc. are.

4.9 Arithmetic operations

Some arithmetic operations are also available: `+`, `-`, `*`, `/` and `%`.

```
<div th:with="isEven=${prodStat.count} % 2 == 0">
```

Note that these operators can also be applied inside OGNL variable expressions themselves (and in that case will be executed by OGNL instead of the Thymeleaf Standard Expression engine):

```
<div th:with="isEven=${prodStat.count % 2 == 0}">
```

Note that textual aliases exist for some of these operators: `div (/)`, `mod (%)`.

4.10 Comparators and Equality

Values in expressions can be compared with the `>`, `<`, `>=` and `<=` symbols, and the `==` and `!=` operators can be used to check for equality (or the lack of it). Note that XML establishes that the `<` and `>` symbols should not be used in attribute values, and so they should be substituted by `<` and `>`.

```
<div th:if="${prodStat.count} > 1">
<span th:text="'Execution mode is ' + ( ${execMode} == 'dev')? 'Development' : 'Production'">
```

A simpler alternative may be using textual aliases that exist for some of these operators: `gt (>)`, `lt (<)`, `ge (>=)`, `le (<=)`, `not (!)`. Also `eq (==)`, `neq / ne (!=)`.

4.11 Conditional expressions

Conditional expressions are meant to evaluate only one of two expressions depending on the result of evaluating a condition (which is itself another expression).

Let's have a look at an example fragment (introducing another *attribute modifier*, `th:class`):

```
<tr th:class="${row.even}? 'even' : 'odd'">
...
</tr>
```

All three parts of a conditional expression (`condition`, `then` and `else`) are themselves expressions, which means that they can be variables (`${...}`, `*{...}`), messages (`# {...}`), URLs (`@ {...}`) or literals (`' ... '`).

Conditional expressions can also be nested using parentheses:

```
<tr th:class="${row.even}? (${row.first}? 'first' : 'even') : 'odd'">
...
</tr>
```

Else expressions can also be omitted, in which case a null value is returned if the condition is false:

```
<tr th:class="${row.even}? 'alt'">
...
</tr>
```

4.12 Default expressions (Elvis operator)

A *default expression* is a special kind of conditional value without a *then* part. It is equivalent to the *Elvis operator* present in some languages like Groovy, lets you specify two expressions: the first one is used if it doesn't evaluate to null, but if it does then the second one is used.

Let's see it in action in our user profile page:

```
<div th:object="${session.user}">
...
<p>Age: <span th:text="*[age]?: '(no age specified)'>27</span>.</p>
</div>
```

As you can see, the operator is `?:`, and we use it here to specify a default value for a name (a literal value, in this case) only if the result of evaluating `*{age}` is null. This is therefore equivalent to:

```
<p>Age: <span th:text="*[age != null]? *[age] : '(no age specified)'>27</span>.</p>
```

As with conditional values, they can contain nested expressions between parentheses:

```
<p>
  Name:
  <span th:text="*[firstName]?: (*{admin}? 'Admin' : #{default.username})>Sebastian</span>
</p>
```

4.13 The No-Operation token

The No-Operation token is represented by an underscore symbol (`_`).

The idea behind this token is to specify that the desired result for an expression is to *do nothing*, i.e. do exactly as if the processable attribute (e.g. `th:text`) was not there at all.

Among other possibilities, this allows developers to use prototyping text as default values. For example, instead of:

```
<span th:text="${user.name} ?: 'no user authenticated'>...</span>
```

...we can directly use '*no user authenticated*' as a prototyping text, which results in code that is both more concise and versatile from a design standpoint:

```
<span th:text="${user.name} ?: _>no user authenticated</span>
```

4.14 Data Conversion / Formatting

Thymeleaf defines a *double-brace* syntax for variable (`${...}`) and selection (`*{...}`) expressions that allows us to apply *data conversion* by means of a configured *conversion service*.

It basically goes like this:

```
<td th:text="${user.lastAccessDate}">...</td>
```

Noticed the double brace there?: `${...}` . That instructs Thymeleaf to pass the result of the `user.lastAccessDate` expression to the *conversion service* and asks it to perform a **formatting operation** (a conversion to `String`) before writing the result.

Assuming that `user.lastAccessDate` is of type `java.util.Calendar` , if a *conversion service* (implementation of `IStandardConversionService`) has been registered and contains a valid conversion for `Calendar -> String` , it will be applied.

The default implementation of `IStandardConversionService` (the `StandardConversionService` class) simply executes `.toString()` on any object converted to `String` . For more information on how to register a custom *conversion service* implementation, have a look at the [More on Configuration](#) section.

The official `thymeleaf-spring3` and `thymeleaf-spring4` integration packages transparently integrate Thymeleaf's conversion service mechanism with Spring's own *Conversion Service* infrastructure, so that conversion services and formatters declared in the Spring configuration will be made automatically available to `${...}` and `*{...}` expressions.

4.15 Preprocessing

In addition to all these features for expression processing, Thymeleaf has the feature of *preprocessing* expressions.

Preprocessing is an execution of the expressions done before the normal one that allows for modification of the expression that will eventually be executed.

Preprocessed expressions are exactly like normal ones, but appear surrounded by a double underscore symbol (like `__${expression}__`).

Let's imagine we have an i18n `Messages_fr.properties` entry containing an OGNL expression calling a language-specific static method, like:

```
article.text=@myapp.translator.Translator@translateToFrench({0})
```

...and a `Messages_es.properties` equivalent:

```
article.text=@myapp.translator.Translator@translateToSpanish({0})
```

We can create a fragment of markup that evaluates one expression or the other depending on the locale. For this, we will first select the expression (by preprocessing) and then let Thymeleaf execute it:

```
<p th:text="__#[article.text('textVar')]__">Some text here...</p>
```

Note that the preprocessing step for a French locale will be creating the following equivalent:

```
<p th:text="${@myapp.translator.Translator@translateToFrench(textVar)}">Some text here...</p>
```

The preprocessing String `__` can be escaped in attributes using `__`.

5 Setting Attribute Values

This chapter will explain the way in which we can set (or modify) values of attributes in our markup.

5.1 Setting the value of any attribute

Say our website publishes a newsletter, and we want our users to be able to subscribe to it, so we create a `/WEB-INF/templates/subscribe.html` template with a form:

```
<form action="subscribe.html">
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="Subscribe!" />
  </fieldset>
</form>
```

As with Thymeleaf, this template starts off more like a static prototype than it does a template for a web application. First, the `action` attribute in our form statically links to the template file itself, so that there is no place for useful URL rewriting. Second, the `value` attribute in the submit button makes it display a text in English, but we'd like it to be internationalized.

Enter then the `th:attr` attribute, and its ability to change the value of attributes of the tags it is set in:

```
<form action="subscribe.html" th:attr="action=@{/subscribe}">
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="Subscribe!" th:attr="value=#{subscribe.submit}"/>
  </fieldset>
</form>
```

The concept is quite straightforward: `th:attr` simply takes an expression that assigns a value to an attribute. Having created the corresponding controller and messages files, the result of processing this file will be:

```
<form action="/gtvg/subscribe">
  <fieldset>
    <input type="text" name="email" />
    <input type="submit" value="¡Suscríbete!" />
  </fieldset>
</form>
```

Besides the new attribute values, you can also see that the application context name has been automatically prefixed to the URL base in `/gtvg/subscribe`, as explained in the previous chapter.

But what if we wanted to set more than one attribute at a time? XML rules do not allow you to set an attribute twice in a tag, so `th:attr` will take a comma-separated list of assignments, like:

```

```

Given the required messages files, this will output:

```

```

5.2 Setting value to specific attributes

By now, you might be thinking that something like:

```
<input type="submit" value="Subscribe!" th:attr="value=#{subscribe.submit}"/>
```

...is quite an ugly piece of markup. Specifying an assignment inside an attribute's value can be very practical, but it is not the most elegant way of creating templates if you have to do it all the time.

Thymeleaf agrees with you, and that's why `th:attr` is scarcely used in templates. Normally, you will be using other `th:*` attributes whose task is setting specific tag attributes (and not just any attribute like `th:attr`).

For example, to set the `value` attribute, use `th:value`:

```
<input type="submit" value="Subscribe!" th:value="#{subscribe.submit}"/>
```

This looks much better! Let's try and do the same to the `action` attribute in the `form` tag:

```
<form action="subscribe.html" th:action="@{/subscribe}">
```

And do you remember those `th:href` we put in our `home.html` before? They are exactly this same kind of attributes:

```
<li><a href="product/list.html" th:href="@{/product/list}">Product List</a></li>
```

There are quite a lot of attributes like these, each of them targeting a specific HTML5 attribute:

th:abbr	th:accept	th:accept-charset
th:accesskey	th:action	th:align
th:alt	th:archive	th:audio
th:autocomplete	th:axis	th:background
th:bgcolor	th:border	th:cellpadding
th:cellspacing	th:challenge	th:charset
th:cite	th:class	th:klassid
th:codebase	th:codetype	th:cols
th:colspan	th:compact	th:content
th:contenteditable	th:contextmenu	th:data
th:datetime	th:dir	th:draggable
th:dropzone	th:enctype	th:for
th:form	th:formaction	th:formenctype
th:formmethod	th:formtarget	th:fragment
th:frame	th:frameborder	th:headers
th:height	th:high	th:href
th:hreflang	th:hspace	th:http-equiv
th:icon	th:id	th:inline
th:keytype	th:kind	th:label
th:lang	th:list	th:longdesc
th:low	th:manifest	th:marginheight

th:marginwidth	th:max	th:maxlength
th:media	th:method	th:min
th:name	th:onabort	th:onafterprint
th:onbeforeprint	th:onbeforeunload	th:onblur
th:oncanplay	th:oncanplaythrough	th:onchange
th:onclick	th:oncontextmenu	th:ondblclick
th:ondrag	th:ondragend	th:ondragenter
th:ondragleave	th:ondragover	th:ondragstart
th:ondrop	th:ondurationchange	th:onemptied
th:onended	th:onerror	th:onfocus
th:onformchange	th:onforminput	th:onhashchange
th:oninput	th:oninvalid	th:onkeydown
th:onkeypress	th:onkeyup	th:onload
th:onloadeddata	th:onloadedmetadata	th:onloadstart
th:onmessage	th:onmousedown	th:onmousemove
th:onmouseout	th:onmouseover	th:onmouseup
th:onmousewheel	th:onoffline	th:ononline
th:onpause	th:onplay	th:onplaying
th:onpopstate	th:onprogress	th:onratechange
th:onreadystatechange	th:onredo	th:onreset
th:onresize	th:onscroll	th:onseeked
th:onseeking	th:onselect	th:onshow
th:onstalled	th:onstorage	th:onsubmit
th:onsuspend	th:ontimeupdate	th:onundo
th:onunload	th:onvolumechange	th:onwaiting
th:optimum	th:pattern	th:placeholder
th:poster	th:preload	th:radioigroup
th:rel	th:rev	th:rows
th:rowspan	th:rules	th:sandbox
th:scheme	th:scope	th:scrolling
th:size	th:sizes	th:span
th:spellcheck	th:src	th:srclang
th:standby	th:start	th:step
th:style	th:summary	th:tabindex
th:target	th:title	th:type
th:usemap	th:value	th:valuetype
th:vspace	th:width	th:wrap
th:xmlbase	th:xmllang	th:xmlspace

5.3 Setting more than one value at a time

There are two rather special attributes called `th:alt-title` and `th:lang-xmllang` which can be used for setting two attributes to the same value at the same time. Specifically:

- `th:alt-title` will set `alt` and `title`.
- `th:lang-xmllang` will set `lang` and `xml:lang`.

For our GTVG home page, this will allow us to substitute this:

```

```

...or this, which is equivalent:

```

```

...with this:

```

```

5.4 Appending and prepending

Thymeleaf also offers the `th:attrappend` and `th:attrprepend` attributes, which append (suffix) or prepend (prefix) the result of their evaluation to the existing attribute values.

For example, you might want to store the name of a CSS class to be added (not set, just added) to one of your buttons in a context variable, because the specific CSS class to be used would depend on something that the user did before:

```
<input type="button" value="Do it!" class="btn" th:attrappend="class=' ' + cssStyle" />
```

If you process this template with the `cssStyle` variable set to "warning", you will get:

```
<input type="button" value="Do it!" class="btn warning" />
```

There are also two specific *appending attributes* in the Standard Dialect: the `th:classappend` and `th:styleappend` attributes, which are used for adding a CSS class or a fragment of `style` to an element without overwriting the existing ones:

```
<tr th:each="prod : ${prods}" class="row" th:classappend="${prodStat.odd}? 'odd'">
```

(Don't worry about that `th:each` attribute. It is an *iterating attribute* and we will talk about it later.)

5.5 Fixed-value boolean attributes

HTML has the concept of *boolean attributes*, attributes that have no value and the presence of one means that value is "true". In XHTML, these attributes take just 1 value, which is itself.

For example, `checked`:

```
<input type="checkbox" name="option2" checked /> <!-- HTML -->
<input type="checkbox" name="option1" checked="checked" /> <!-- XHTML -->
```

The Standard Dialect includes attributes that allow you to set these attributes by evaluating a condition, so that if evaluated to true, the attribute will be set to its fixed value, and if evaluated to false, the attribute will not be set:

```
<input type="checkbox" name="active" th:checked="${user.active}" />
```

The following fixed-value boolean attributes exist in the Standard Dialect:

th:async	th:autofocus	th:autoplay
th:checked	th:controls	th:declare
th:default	th:defer	th:disabled
th:formnovalidate	th:hidden	th:ismap
th:loop	th:multiple	th:noplay
th:nowrap	th:open	th:pubdate
th:readonly	th:required	th:reversed
th:scoped	th:seamless	th:selected

5.6 Setting the value of any attribute (default attribute processor)

Thymeleaf offers a *default attribute processor* that allows us to set the value of *any* attribute, even if no specific `th:*` processor has been defined for it at the Standard Dialect.

So something like:

```
<span th:whatever="${user.name}">...</span>
```

Will result in:

```
<span whatever="John Apricot">...</span>
```

5.7 Support for HTML5-friendly attribute and element names

It is also possible to use a completely different syntax to apply processors to your templates in a more HTML5-friendly manner.

```
<table>
  <tr data-th-each="user : ${users}">
    <td data-th-text="${user.login}">...</td>
    <td data-th-text="${user.name}">...</td>
  </tr>
</table>
```

The `data-{prefix}-{name}` syntax is the standard way to write custom attributes in HTML5, without requiring developers to use any namespaced names like `th:*`. Thymeleaf makes this syntax automatically available to all your dialects (not only the Standard ones).

There is also a syntax to specify custom tags: `{prefix}-{name}`, which follows the *W3C Custom Elements specification* (a part of the larger *W3C Web Components spec*). This can be used, for example, for the `th:block` element (or also `th-block`), which will be explained in a later section.

Important: this syntax is an addition to the namespaced `th:*` one, it does not replace it. There is no intention at all to deprecate the namespaced syntax in the future.

6 Iteration

So far we have created a home page, a user profile page and also a page for letting users subscribe to our newsletter... but what about our products? For that, we will need a way to iterate over items in a collection to build out our product page.

6.1 Iteration basics

To display products in our `/WEB-INF/templates/product/list.html` page we will use a table. Each of our products will be displayed in a row (a `<tr>` element), and so for our template we will need to create a *template row* – one that will exemplify how we want each product to be displayed – and then instruct Thymeleaf to repeat it, once for each product.

The Standard Dialect offers us an attribute for exactly that: `th:each`.

Using `th:each`

For our product list page, we will need a controller method that retrieves the list of products from the service layer and adds it to the template context:

```
public void process(
    final HttpServletRequest request, final HttpServletResponse response,
    final ServletContext servletContext, final ITemplateEngine templateEngine)
    throws Exception {

    ProductService productService = new ProductService();
    List<Product> allProducts = productService.findAll();

    WebContext ctx = new WebContext(request, response, servletContext, request.getLocale());
    ctx.setVariable("prods", allProducts);

    templateEngine.process("product/list", ctx, response.getWriter());
}
```

And then we will use `th:each` in our template to iterate over the list of products:

```

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

    <head>
        <title>Good Thymes Virtual Grocery</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link rel="stylesheet" type="text/css" media="all"
              href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />
    </head>

    <body>

        <h1>Product list</h1>

        <table>
            <tr>
                <th>NAME</th>
                <th>PRICE</th>
                <th>IN STOCK</th>
            </tr>
            <tr th:each="prod : ${prods}">
                <td th:text="${prod.name}">Onions</td>
                <td th:text="${prod.price}">2.41</td>
                <td th:text="${prod.inStock} ? ${true} : ${false}">yes</td>
            </tr>
        </table>

        <p>
            <a href="../home.html" th:href="@{}>Return to home</a>
        </p>
    </body>
</html>

```

That `prod : ${prods}` attribute value you see above means “for each element in the result of evaluating `${prods}`, repeat this fragment of template, using the current element in a variable called `prod`”. Let’s give a name each of the things we see:

- We will call `${prods}` the *iterated expression* or *iterated variable*.
- We will call `prod` the *iteration variable* or simply *iter variable*.

Note that the `prod` iter variable is scoped to the `<tr>` element, which means it is available to inner tags like `<td>`.

Iterable values

The `java.util.List` class isn’t the only value that can be used for iteration in Thymeleaf. There is a quite complete set of objects that are considered *iterable* by a `th:each` attribute:

- Any object implementing `java.util.Iterable`
- Any object implementing `java.util Enumeration`.
- Any object implementing `java.util.Iterator`, whose values will be used as they are returned by the iterator, without the need to cache all values in memory.
- Any object implementing `java.util.Map`. When iterating maps, iter variables will be of class `java.util.Map.Entry`.
- Any array.
- Any other object will be treated as if it were a single-valued list containing the object itself.

6.2 Keeping iteration status

When using `th:each`, Thymeleaf offers a mechanism useful for keeping track of the status of your iteration: the `status variable`.

Status variables are defined within a `th:each` attribute and contain the following data:

- The current *iteration index*, starting with 0. This is the `index` property.
- The current *iteration index*, starting with 1. This is the `count` property.
- The total amount of elements in the iterated variable. This is the `size` property.
- The `iter variable` for each iteration. This is the `current` property.
- Whether the current iteration is even or odd. These are the `even/odd` boolean properties.
- Whether the current iteration is the first one. This is the `first` boolean property.
- Whether the current iteration is the last one. This is the `last` boolean property.

Let's see how we could use it with the previous example:

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
  </tr>
  <tr th:each="prod,iterStat : ${prods}" th:class="${iterStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #[true] : #[false]">yes</td>
  </tr>
</table>
```

The status variable (`iterStat` in this example) is defined in the `th:each` attribute by writing its name after the `iter` variable itself, separated by a comma. Just like the `iter` variable, the status variable is also scoped to the fragment of code defined by the tag holding the `th:each` attribute.

Let's have a look at the result of processing our template:

```

<!DOCTYPE html>

<html>

<head>
    <title>Good Thymes Virtual Grocery</title>
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>
    <link rel="stylesheet" type="text/css" media="all" href="/gtvg/css/gtvg.css" />
</head>

<body>

    <h1>Product list</h1>

    <table>
        <tr>
            <th>NAME</th>
            <th>PRICE</th>
            <th>IN STOCK</th>
        </tr>
        <tr class="odd">
            <td>Fresh Sweet Basil</td>
            <td>4.99</td>
            <td>yes</td>
        </tr>
        <tr>
            <td>Italian Tomato</td>
            <td>1.25</td>
            <td>no</td>
        </tr>
        <tr class="odd">
            <td>Yellow Bell Pepper</td>
            <td>2.50</td>
            <td>yes</td>
        </tr>
        <tr>
            <td>Old Cheddar</td>
            <td>18.75</td>
            <td>yes</td>
        </tr>
    </table>

    <p>
        <a href="/gtvg/" shape="rect">Return to home</a>
    </p>
</body>

</html>

```

Note that our iteration status variable has worked perfectly, establishing the `odd` CSS class only to odd rows.

If you don't explicitly set a status variable, Thymeleaf will always create one for you by suffixing `Stat` to the name of the iteration variable:

```

<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #[true] : #[false]">yes</td>
  </tr>
</table>

```

6.3 Optimizing through lazy retrieval of data

Sometimes we might want to optimize the retrieval of collections of data (e.g. from a database) so that these collections are only retrieved if they are really going to be used.

Actually, this is something that can be applied to *any* piece of data, but given the size that in-memory collections might have, retrieving collections that are meant to be iterated is the most common case for this scenario.

In order to support this, Thymeleaf offers a mechanism to *lazily load context variables*. Context variables that implement the `ILazyContextVariable` interface – most probably by extending its `LazyContextVariable` default implementation – will be resolved in the moment of being executed. For example:

```

context.setVariable(
    "users",
    new LazyContextVariable<List<User>>() {
        @Override
        protected List<User> loadValue() {
            return databaseRepository.findAllUsers();
        }
    });

```

This variable can be used without knowledge of its *lazyness*, in code such as:

```

<ul>
  <li th:each="u : ${users}" th:text="${u.name}">user name</li>
</ul>

```

But at the same time, will never be initialized (its `loadValue()` method will never be called) if condition evaluates to false in code such as:

```

<ul th:if="${condition}">
  <li th:each="u : ${users}" th:text="${u.name}">user name</li>
</ul>

```

7 Conditional Evaluation

7.1 Simple conditionals: "if" and "unless"

Sometimes you will need a fragment of your template to only appear in the result if a certain condition is met.

For example, imagine we want to show in our product table a column with the number of comments that exist for each product and, if there are any comments, a link to the comment detail page for that product.

In order to do this, we would use the `th:if` attribute:

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #[true] : #[false]">yes</td>
    <td>
      <span th:text="#{lists.size(prod.comments)}">2</span> comment/s
      <a href="comments.html"
          th:href="@{/product/comments(productId=${prod.id})}"
          th:if="${not #lists.isEmpty(prod.comments)}">view</a>
    </td>
  </tr>
</table>
```

Quite a lot of things to see here, so let's focus on the important line:

```
<a href="comments.html"
    th:href="@{/product/comments(productId=${prod.id})}"
    th:if="${not #lists.isEmpty(prod.comments)}">view</a>
```

This will create a link to the comments page (with URL `/product/comments`) with a `prodId` parameter set to the `id` of the product, but only if the product has any comments.

Let's have a look at the resulting markup:

```

<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr>
    <td>Fresh Sweet Basil</td>
    <td>4.99</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Italian Tomato</td>
    <td>1.25</td>
    <td>no</td>
    <td>
      <span>2</span> comment/s
      <a href="/gtvg/product/comments?prodId=2">view</a>
    </td>
  </tr>
  <tr>
    <td>Yellow Bell Pepper</td>
    <td>2.50</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Old Cheddar</td>
    <td>18.75</td>
    <td>yes</td>
    <td>
      <span>1</span> comment/s
      <a href="/gtvg/product/comments?prodId=4">view</a>
    </td>
  </tr>
</table>

```

Perfect! That's exactly what we wanted.

Note that the `th:if` attribute will not only evaluate *boolean* conditions. Its capabilities go a little beyond that, and it will evaluate the specified expression as `true` following these rules:

- If value is not null:
 - If value is a boolean and is `true`.
 - If value is a number and is non-zero
 - If value is a character and is non-zero
 - If value is a String and is not "false", "off" or "no"
 - If value is not a boolean, a number, a character or a String.
- (If value is null, `th:if` will evaluate to false).

Also, `th:if` has an inverse attribute, `th:unless`, which we could have used in the previous example instead of using a `not` inside the OGNL expression:

```
<a href="comments.html"
    th:href="@{/comments(prodId=${prod.id})}"
    th:unless="#{lists.isEmpty(prod.comments)}">view</a>
```

7.2 Switch statements

There is also a way to display content conditionally using the equivalent of a `switch` structure in Java: the `th:switch` / `th:case` attribute set.

```
<div th:switch="${user.role}">
    <p th:case="'admin'">User is an administrator</p>
    <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

Note that as soon as one `th:case` attribute is evaluated as `true`, every other `th:case` attribute in the same switch context is evaluated as `false`.

The default option is specified as `th:case="*"`:

```
<div th:switch="${user.role}">
    <p th:case="'admin'">User is an administrator</p>
    <p th:case="#{roles.manager}">User is a manager</p>
    <p th:case="*"/>User is some other thing</p>
</div>
```

8 Template Layout

8.1 Including template fragments

Defining and referencing fragments

In our templates, we will often want to include parts from other templates, parts like footers, headers, menus...

In order to do this, Thymeleaf needs us to define these parts, "fragments", for inclusion, which can be done using the `th:fragment` attribute.

Say we want to add a standard copyright footer to all our grocery pages, so we create a `/WEB-INF/templates/footer.html` file containing this code:

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<body>

<div th:fragment="copy">
    &copy; 2011 The Good Thymes Virtual Grocery
</div>

</body>

</html>
```

The code above defines a fragment called `copy` that we can easily include in our home page using one of the `th:insert` or `th:replace` attributes (and also `th:include`, though its use is no longer recommended since Thymeleaf 3.0):

```
<body>

    ...

    <div th:insert="~{footer :: copy}"></div>

</body>
```

Note that `th:insert` expects a *fragment expression* (`~{...}`), which is *an expression that results in a fragment*. In the above example though, which is a non-complex *fragment expression*, the (`~{ , }`) enclosing is completely optional, so the code above would be equivalent to:

```
<body>

    ...

    <div th:insert="footer :: copy"></div>

</body>
```

Fragment specification syntax

The syntax of *fragment expressions* is quite straightforward. There are three different formats:

- "`~{templatename::selector}`" Includes the fragment resulting from applying the specified Markup Selector on the template named `templatename`. Note that `selector` can be a mere fragment name, so you could specify something as simple as `~{templatename::fragmentname}` like in the `~{footer :: copy}` above.

Markup Selector syntax is defined by the underlying AttoParser parsing library, and is similar to XPath expressions or CSS selectors. See [Appendix C](#) for more info.

- "`~{templatename}`" Includes the complete template named `templatename`.

Note that the template name you use in `th:insert / th:replace` tags will have to be resolvable by the Template Resolver currently being used by the Template Engine.

- "`~{::selector}`" or "`~{this::selector}`" Inserts a fragment from the same template, matching `selector`. If not found on the template where the expression appears, the stack of template calls (insertions) is traversed towards the originally processed template (the *root*), until `selector` matches at some level.

Both `templatename` and `selector` in the above examples can be fully-featured expressions (even conditionals!) like:

```
<div th:insert="footer :: (${user.isAdmin}? #{footer.admin} : #{footer.normaluser})"></div>
```

Note again how the surrounding `~{...}` envelope is optional in `th:insert / th:replace`.

Fragments can include any `th:*` attributes. These attributes will be evaluated once the fragment is included into the target template (the one with the `th:insert / th:replace` attribute), and they will be able to reference any context variables defined in this target template.

A big advantage of this approach to fragments is that you can write your fragments in pages that are perfectly displayable by a browser, with a complete and even *valid* markup structure, while still retaining the ability to make Thymeleaf include them into other templates.

Referencing fragments without `th:fragment`

Thanks to the power of Markup Selectors, we can include fragments that do not use any `th:fragment` attributes. It can even be markup code coming from a different application with no knowledge of Thymeleaf at all:

```
...
<div id="copy-section">
    &copy; 2011 The Good Thymes Virtual Grocery
</div>
...
```

We can use the fragment above simply referencing it by its `id` attribute, in a similar way to a CSS selector:

```
<body>  
    ...  
    <div th:insert="~{footer :: #copy-section}"></div>  
</body>
```

Difference between `th:insert` and `th:replace` (and `th:include`)

And what is the difference between `th:insert` and `th:replace` (and `th:include`, not recommended since 3.0)?

- `th:insert` is the simplest: it will simply insert the specified fragment as the body of its host tag.
- `th:replace` actually *replaces* its host tag with the specified fragment.
- `th:include` is similar to `th:insert`, but instead of inserting the fragment it only inserts the *contents* of this fragment.

So an HTML fragment like this:

```
<footer th:fragment="copy">  
    &copy; 2011 The Good Thymes Virtual Grocery  
</footer>
```

...included three times in host `<div>` tags, like this:

```
<body>  
    ...  
    <div th:insert="footer :: copy"></div>  
    <div th:replace="footer :: copy"></div>  
    <div th:include="footer :: copy"></div>  
</body>
```

...will result in:

```

<body>
    ...
    <div>
        <footer>
            © 2011 The Good Thymes Virtual Grocery
        </footer>
    </div>

    <footer>
        © 2011 The Good Thymes Virtual Grocery
    </footer>

    <div>
        © 2011 The Good Thymes Virtual Grocery
    </div>

</body>

```

8.2 Parameterizable fragment signatures

In order to create a more *function-like* mechanism for template fragments, fragments defined with `th:fragment` can specify a set of parameters:

```

<div th:fragment="frag (onevar,twovar)">
    <p th:text="${onevar} + ' - ' + ${twovar}">...</p>
</div>

```

This requires the use of one of these two syntaxes to call the fragment from `th:insert` or `th:replace`:

```

<div th:replace="::frag (${value1},${value2})">...</div>
<div th:replace="::frag (onevar=${value1},twovar=${value2})">...</div>

```

Note that order is not important in the last option:

```

<div th:replace="::frag (twovar=${value2},onevar=${value1})">...</div>

```

Fragment local variables without fragment arguments

Even if fragments are defined without arguments like this:

```

<div th:fragment="frag">
    ...
</div>

```

We could use the second syntax specified above to call them (and only the second one):

```

<div th:replace="::frag (onevar=${value1},twovar=${value2})">

```

This would be equivalent to a combination of `th:replace` and `th:with`:

```
<div th:replace="::frag" th:with="onevar=${value1},twovar=${value2}">
```

Note that this specification of local variables for a fragment – no matter whether it has an argument signature or not – does not cause the context to be emptied prior to its execution. Fragments will still be able to access every context variable being used at the calling template like they currently are.

th:assert for in-template assertions

The `th:assert` attribute can specify a comma-separated list of expressions which should be evaluated and produce true for every evaluation, raising an exception if not.

```
<div th:assert="${onevar},(${twovar} != 43)">...</div>
```

This comes in handy for validating parameters at a fragment signature:

```
<header th:fragment="contentheader(title)" th:assert="${!#strings.isEmpty(title)}">...</header>
```

8.3 Flexible layouts: beyond mere fragment insertion

Thanks to *fragment expressions*, we can specify parameters for fragments that are not texts, numbers, bean objects... but instead fragments of markup.

This allows us to create our fragments in a way such that they can be *enriched* with markup coming from the calling templates, resulting in a very flexible **template layout mechanism**.

Note the use of the `title` and `links` variables in the fragment below:

```
<head th:fragment="common_header(title,links)">

    <title th:replace="${title}">The awesome application</title>

    <!-- Common styles and scripts -->
    <link rel="stylesheet" type="text/css" media="all" th:href="@{/css/awesomeapp.css}">
    <link rel="shortcut icon" th:href="@{/images/favicon.ico}">
    <script type="text/javascript" th:src="@{/sh/scripts/codebase.js}"></script>

    <!--/* Per-page placeholder for additional links */-->
    <th:block th:replace="${links}" />

</head>
```

We can now call this fragment like:

```
...
<head th:replace="base :: common_header(~{::title},~{::link})">

    <title>Awesome - Main</title>

    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
    <link rel="stylesheet" th:href="@{/themes/smoothness/jquery-ui.css}">

</head>
...
```

...and the result will use the actual `<title>` and `<link>` tags from our calling template as the values of the `title` and `links` variables, resulting in our fragment being customized during insertion:

```
...
<head>

  <title>Awesome - Main</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

  <link rel="stylesheet" href="/awe/css/bootstrap.min.css">
  <link rel="stylesheet" href="/awe/themes/smoothness/jquery-ui.css">

</head>
...
```

Using the empty fragment

A special fragment expression, the *empty fragment* (`~{}`), can be used for specifying *no markup*. Using the previous example:

```
<head th:replace="base :: common_header(~{::title},~{})">

  <title>Awesome - Main</title>

</head>
...
```

Note how the second parameter of the fragment (`links`) is set to the *empty fragment* and therefore nothing is written for the `<th:block th:replace="${links}" />` block:

```
...
<head>

  <title>Awesome - Main</title>

  <!-- Common styles and scripts -->
  <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
  <link rel="shortcut icon" href="/awe/images/favicon.ico">
  <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

</head>
...
```

Using the no-operation token

The no-op can be also used as a parameter to a fragment if we just want to let our fragment use its current markup as a default value. Again, using the `common_header` example:

```

...
<head th:replace="base :: common_header(_,~{::link})">

    <title>Awesome - Main</title>

    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
    <link rel="stylesheet" th:href="@{/themes/smoothness/jquery-ui.css}">

</head>
...

```

See how the `title` argument (first argument of the `common_header` fragment) is set to `no-op (_)`, which results in this part of the fragment not being executed at all (`title = no-operation`):

```
<title th:replace="${title}">The awesome application</title>
```

So the result is:

```

...
<head>

    <title>The awesome application</title>

    <!-- Common styles and scripts -->
    <link rel="stylesheet" type="text/css" media="all" href="/awe/css/awesomeapp.css">
    <link rel="shortcut icon" href="/awe/images/favicon.ico">
    <script type="text/javascript" src="/awe/sh/scripts/codebase.js"></script>

    <link rel="stylesheet" href="/awe/css/bootstrap.min.css">
    <link rel="stylesheet" href="/awe/themes/smoothness/jquery-ui.css">

</head>
...

```

Advanced conditional insertion of fragments

The availability of both the `empty fragment` and `no-operation token` allows us to perform conditional insertion of fragments in a very easy and elegant way.

For example, we could do this in order to insert our `common :: adminhead` fragment *only* if the user is an administrator, and insert nothing (empty fragment) if not:

```

...
<div th:insert="${user.isAdmin()} ? ~{common :: adminhead} : ~{}">...</div>
...

```

Also, we can use the `no-operation token` in order to insert a fragment only if the specified condition is met, but leave the markup without modifications if the condition is not met:

```

...
<div th:insert="${user.isAdmin()} ? ~{common :: adminhead} : _">
    Welcome [[${user.name}]], click <a th:href="@{/support}">here</a> for help-desk support.
</div>
...

```

Additionally, if we have configured our template resolvers to *check for existence* of the template resources -- by means

of their `checkExistence` flag — we can use the existence of the fragment itself as the condition in a `default` operation:

```
...
<!-- The body of the <div> will be used if the "common :: salutation" fragment -->
<!-- does not exist (or is empty).-->
<div th:insert="~{common :: salutation} ?: _">
    Welcome [[${user.name}]], click <a th:href="@{/support}">here</a> for help-desk support.
</div>
...
```

8.4 Removing template fragments

Back to the example application, let's revisit the last version of our product list template:

```
<table>
    <tr>
        <th>NAME</th>
        <th>PRICE</th>
        <th>IN STOCK</th>
        <th>COMMENTS</th>
    </tr>
    <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
        <td th:text="${prod.name}">Onions</td>
        <td th:text="${prod.price}">2.41</td>
        <td th:text="${prod.inStock}? #[true] : #[false]">yes</td>
        <td>
            <span th:text="#{lists.size(prod.comments)}>2</span> comment/s
            <a href="comments.html"
                th:href="@{/product/comments(prodId=${prod.id})}"
                th:unless="#{lists.isEmpty(prod.comments)}">view</a>
        </td>
    </tr>
</table>
```

This code is just fine as a template, but as a static page (when directly open by a browser without Thymeleaf processing it) it would not make a nice prototype.

Why? Because, although perfectly displayable by browsers, that table only has a row, and this row has mock data. As a prototype, it simply wouldn't look realistic enough... we should have more than one product, *we need more rows*.

So let's add some:

```

<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes</td>
    <td>
      <span th:text="#{lists.size(prod.comments)}>2</span> comment/s
      <a href="comments.html"
          th:href="@{/product/comments(prodId=${prod.id})}"
          th:unless="#{lists.isEmpty(prod.comments)}">view</a>
    </td>
  </tr>
  <tr class="odd">
    <td>Blue Lettuce</td>
    <td>9.55</td>
    <td>no</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr>
    <td>Mild Cinnamon</td>
    <td>1.99</td>
    <td>yes</td>
    <td>
      <span>3</span> comment/s
      <a href="comments.html">view</a>
    </td>
  </tr>
</table>

```

Ok, now we have three, definitely better for a prototype. But... what will happen when we process it with Thymeleaf?:

```

<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr>
    <td>Fresh Sweet Basil</td>
    <td>4.99</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Italian Tomato</td>
    <td>1.25</td>
    <td>no</td>
    <td>
      <span>2</span> comment/s
      <a href="/gtvg/product/comments?prodId=2">view</a>
    </td>
  </tr>
  <tr>
    <td>Yellow Bell Pepper</td>
    <td>2.50</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Old Cheddar</td>
    <td>18.75</td>
    <td>yes</td>
    <td>
      <span>1</span> comment/s
      <a href="/gtvg/product/comments?prodId=4">view</a>
    </td>
  </tr>
  <tr class="odd">
    <td>Blue Lettuce</td>
    <td>9.55</td>
    <td>no</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr>
    <td>Mild Cinnamon</td>
    <td>1.99</td>
    <td>yes</td>
    <td>
      <span>3</span> comment/s
      <a href="comments.html">view</a>
    </td>
  </tr>
</table>

```

The last two rows are mock rows! Well, of course they are: iteration was only applied to the first row, so there is no reason why Thymeleaf should have removed the other two.

We need a way to remove those two rows during template processing. Let's use the `th:remove` attribute on the second

and third <tr> tags:

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #[true] : #[false]">yes</td>
    <td>
      <span th:text="#{lists.size(prod.comments)}">2</span> comment/s
      <a href="comments.html"
          th:href="@{/product/comments(prodId=${prod.id})}"
          th:unless="#{lists.isEmpty(prod.comments)}">view</a>
    </td>
  </tr>
  <tr class="odd" th:remove="all">
    <td>Blue Lettuce</td>
    <td>9.55</td>
    <td>no</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr th:remove="all">
    <td>Mild Cinnamon</td>
    <td>1.99</td>
    <td>yes</td>
    <td>
      <span>3</span> comment/s
      <a href="comments.html">view</a>
    </td>
  </tr>
</table>
```

Once processed, everything will look again as it should:

```

<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr>
    <td>Fresh Sweet Basil</td>
    <td>4.99</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Italian Tomato</td>
    <td>1.25</td>
    <td>no</td>
    <td>
      <span>2</span> comment/s
      <a href="/gtvg/product/comments?prodId=2">view</a>
    </td>
  </tr>
  <tr>
    <td>Yellow Bell Pepper</td>
    <td>2.50</td>
    <td>yes</td>
    <td>
      <span>0</span> comment/s
    </td>
  </tr>
  <tr class="odd">
    <td>Old Cheddar</td>
    <td>18.75</td>
    <td>yes</td>
    <td>
      <span>1</span> comment/s
      <a href="/gtvg/product/comments?prodId=4">view</a>
    </td>
  </tr>
</table>

```

And what does that `all` value in the attribute, mean? `th:remove` can behave in five different ways, depending on its value:

- `all` : Remove both the containing tag and all its children.
- `body` : Do not remove the containing tag, but remove all its children.
- `tag` : Remove the containing tag, but do not remove its children.
- `all-but-first` : Remove all children of the containing tag except the first one.
- `none` : Do nothing. This value is useful for dynamic evaluation.

What can that `all-but-first` value be useful for? It will let us save some `th:remove="all"` when prototyping:

```

<table>
  <thead>
    <tr>
      <th>NAME</th>
      <th>PRICE</th>
      <th>IN STOCK</th>
      <th>COMMENTS</th>
    </tr>
  </thead>
  <tbody th:remove="all-but-first">
    <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
      <td th:text="${prod.name}">Onions</td>
      <td th:text="${prod.price}">2.41</td>
      <td th:text="${prod.inStock}? ${true} : ${false}">yes</td>
      <td>
        <span th:text="#{lists.size(prod.comments)}">2</span> comment/s
        <a href="comments.html"
          th:href="@{/product/comments(prodId=${prod.id})}"
          th:unless="#{lists.isEmpty(prod.comments)}">view</a>
      </td>
    </tr>
    <tr class="odd">
      <td>Blue Lettuce</td>
      <td>9.55</td>
      <td>no</td>
      <td>
        <span>0</span> comment/s
      </td>
    </tr>
    <tr>
      <td>Mild Cinnamon</td>
      <td>1.99</td>
      <td>yes</td>
      <td>
        <span>3</span> comment/s
        <a href="comments.html">view</a>
      </td>
    </tr>
  </tbody>
</table>

```

The `th:remove` attribute can take any *Thymeleaf Standard Expression*, as long as it returns one of the allowed String values (`all`, `tag`, `body`, `all-but-first` or `none`).

This means removals could be conditional, like:

```
<a href="/something" th:remove="${condition}? tag : none">Link text not to be removed</a>
```

Also note that `th:remove` considers `null` a synonym to `none`, so the following works the same as the example above:

```
<a href="/something" th:remove="${condition}? tag">Link text not to be removed</a>
```

In this case, if `${condition}` is false, `null` will be returned, and thus no removal will be performed.

8.5 Layout Inheritance

To be able to have a single file as layout, fragments can be used. An example of a simple layout having `title` and `content` using `th:fragment` and `th:replace`:

```
<!DOCTYPE html>
<html th:fragment="layout (title, content)" xmlns:th="http://www.thymeleaf.org">
<head>
    <title th:replace="${title}">Layout Title</title>
</head>
<body>
    <h1>Layout H1</h1>
    <div th:replace="${content}">
        <p>Layout content</p>
    </div>
    <footer>
        Layout footer
    </footer>
</body>
</html>
```

This example declares a fragment called `layout` having `title` and `content` as parameters. Both will be replaced on page inheriting it by provided fragment expressions in the example below.

```
<!DOCTYPE html>
<html th:replace=~{layoutFile :: layout(~{::title}, ~{::section})}">
<head>
    <title>Page Title</title>
</head>
<body>
<section>
    <p>Page content</p>
    <div>Included on page</div>
</section>
</body>
</html>
```

In this file, the `html` tag will be replaced by `layout`, but in the layout `title` and `content` will have been replaced by `title` and `section` blocks respectively.

If desired, the layout can be composed by several fragments as `header` and `footer`.

9 Local Variables

Thymeleaf calls *local variables* the variables that are defined for a specific fragment of a template, and are only available for evaluation inside that fragment.

An example we have already seen is the `prod` iter variable in our product list page:

```
<tr th:each="prod : ${prods}">
    ...
</tr>
```

That `prod` variable will be available only within the bounds of the `<tr>` tag. Specifically:

- It will be available for any other `th:*` attributes executing in that tag with less *precedence* than `th:each` (which means they will execute after `th:each`).
- It will be available for any child element of the `<tr>` tag, such as any `<td>` elements.

Thymeleaf offers you a way to declare local variables without iteration, using the `th:with` attribute, and its syntax is like that of attribute value assignments:

```
<div th:with="firstPer=${persons[0]}>
    <p>
        The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.
    </p>
</div>
```

When `th:with` is processed, that `firstPer` variable is created as a local variable and added to the variables map coming from the context, so that it is available for evaluation along with any other variables declared in the context, but only within the bounds of the containing `<div>` tag.

You can define several variables at the same time using the usual multiple assignment syntax:

```
<div th:with="firstPer=${persons[0]},secondPer=${persons[1]}>
    <p>
        The name of the first person is <span th:text="${firstPer.name}">Julius Caesar</span>.
    </p>
    <p>
        But the name of the second person is
        <span th:text="${secondPer.name}">Marcus Antonius</span>.
    </p>
</div>
```

The `th:with` attribute allows reusing variables defined in the same attribute:

```
<div th:with="company=${user.company + ' Co.'},account=${accounts[company]}>...</div>
```

Let's use this in our Grocery's home page! Remember the code we wrote for outputting a formatted date?

```
<p>
    Today is:
    <span th:text="#{calendars.format(today, 'dd MMMM yyyy')}>13 february 2011</span>
</p>
```

Well, what if we wanted that "dd MMMM yyyy" to actually depend on the locale? For example, we might want to add the

following message to our `home_en.properties`:

```
date.format=MMMM dd'', '' yyyy
```

...and an equivalent one to our `home_es.properties`:

```
date.format=dd ''de'' MMMM'', '' yyyy
```

Now, let's use `th:with` to get the localized date format into a variable, and then use it in our `th:text` expression:

```
<p th:with="df=#{date.format}">  
    Today is: <span th:text="${#calendars.format(today,df)}">13 February 2011</span>  
</p>
```

That was clean and easy. In fact, given the fact that `th:with` has a higher precedence than `th:text`, we could have solved this all in the `span` tag:

```
<p>  
    Today is:  
    <span th:with="df=#{date.format}"  
          th:text="${#calendars.format(today,df)}">13 February 2011</span>  
</p>
```

You might be thinking: Precedence? We haven't talked about that yet! Well, don't worry because that is exactly what the next chapter is about.

10 Attribute Precedence

What happens when you write more than one `th:*` attribute in the same tag? For example:

```
<ul>
  <li th:each="item : ${items}" th:text="${item.description}">Item description here...</li>
</ul>
```

We would expect that `th:each` attribute to execute before the `th:text` so that we get the results we want, but given the fact that the HTML/XML standards do not give any kind of meaning to the order in which the attributes in a tag are written, a *precedence* mechanism had to be established in the attributes themselves in order to be sure that this will work as expected.

So, all Thymeleaf attributes define a numeric precedence, which establishes the order in which they are executed in the tag. This order is:

Order	Feature	Attributes
1	Fragment inclusion	<code>th:insert</code> <code>th:replace</code>
2	Fragment iteration	<code>th:each</code>
3	Conditional evaluation	<code>th:if</code> <code>th:unless</code> <code>th:switch</code> <code>th:case</code>
4	Local variable definition	<code>th:object</code> <code>th:with</code>
5	General attribute modification	<code>th:attr</code> <code>th:attrprepend</code> <code>th:attrappend</code>
6	Specific attribute modification	<code>th:value</code> <code>th:href</code> <code>th:src</code> ...
7	Text (tag body modification)	<code>th:text</code> <code>th:utext</code>
8	Fragment specification	<code>th:fragment</code>
9	Fragment removal	<code>th:remove</code>

This precedence mechanism means that the above iteration fragment will give exactly the same results if the attribute position is inverted (although it would be slightly less readable):

```
<ul>
  <li th:text="${item.description}" th:each="item : ${items}">Item description here...</li>
</ul>
```

11 Comments and Blocks

11.1. Standard HTML/XML comments

Standard HTML/XML comments `<!-- ... -->` can be used anywhere in Thymeleaf templates. Anything inside these comments won't be processed by Thymeleaf, and will be copied verbatim to the result:

```
<!-- User info follows -->
<div th:text="${...}">
    ...
</div>
```

11.2. Thymeleaf parser-level comment blocks

Parser-level comment blocks are code that will be simply removed from the template when Thymeleaf parses it. They look like this:

```
<!--/* This code will be removed at Thymeleaf parsing time! */-->
```

Thymeleaf will remove everything between `<!--/*` and `*/-->`, so these comment blocks can also be used for displaying code when a template is statically open, knowing that it will be removed when Thymeleaf processes it:

```
<!--/*-->
<div>
    you can see me only before Thymeleaf processes me!
</div>
<!--*/-->
```

This might come very handy for prototyping tables with a lot of `<tr>`'s, for example:

```
<table>
    <tr th:each="x : ${xs}">
        ...
    </tr>
    <!--/*-->
    <tr>
        ...
    </tr>
    <tr>
        ...
    </tr>
    <!--*/-->
</table>
```

11.3. Thymeleaf prototype-only comment blocks

Thymeleaf allows the definition of special comment blocks marked to be comments when the template is open statically (i.e. as a prototype), but considered normal markup by Thymeleaf when executing the template.

```

<span>hello!</span>
<!--/*
  <div th:text="${...}">
  ...
  </div>
/*-->
<span>goodbye!</span>

```

Thymeleaf's parsing system will simply remove the `<!--/*/` and `/*-->` markers, but not its contents, which will be left therefore uncommented. So when executing the template, Thymeleaf will actually see this:

```

<span>hello!</span>

<div th:text="${...}">
  ...
</div>

<span>goodbye!</span>

```

As with parser-level comment blocks, this feature is dialect-independent.

11.4. Synthetic `th:block` tag

Thymeleaf's only element processor (not an attribute) included in the Standard Dialects is `th:block`.

`th:block` is a mere attribute container that allows template developers to specify whichever attributes they want. Thymeleaf will execute these attributes and then simply make the block, but not its contents, disappear.

So it could be useful, for example, when creating iterated tables that require more than one `<tr>` for each element:

```

<table>
  <th:block th:each="user : ${users}">
    <tr>
      <td th:text="${user.login}">...</td>
      <td th:text="${user.name}">...</td>
    </tr>
    <tr>
      <td colspan="2" th:text="${user.address}">...</td>
    </tr>
  </th:block>
</table>

```

And especially useful when used in combination with prototype-only comment blocks:

```

<table>
  <!--/* <th:block th:each="user : ${users}"> /*-->
    <tr>
      <td th:text="${user.login}">...</td>
      <td th:text="${user.name}">...</td>
    </tr>
    <tr>
      <td colspan="2" th:text="${user.address}">...</td>
    </tr>
  <!--/* </th:block> /*-->
</table>

```

Note how this solution allows templates to be valid HTML (no need to add forbidden `<div>` blocks inside `<table>`), and still works OK when open statically in browsers as prototypes!

12 Inlining

12.1 Expression inlining

Although the Standard Dialect allows us to do almost everything using tag attributes, there are situations in which we could prefer writing expressions directly into our HTML texts. For example, we could prefer writing this:

```
<p>Hello, [[${session.user.name}]]!</p>
```

...instead of this:

```
<p>Hello, <span th:text="${session.user.name}">Sebastian</span>!</p>
```

Expressions between `[[...]]` or `((...))` are considered **inlined expressions** in Thymeleaf, and inside them we can use any kind of expression that would also be valid in a `th:text` or `th:utext` attribute.

Note that, while `[[...]]` corresponds to `th:text` (i.e. result will be *HTML-escaped*), `((...))` corresponds to `th:utext` and will not perform any HTML-escaping. So with a variable such as `msg = 'This is great!'`, given this fragment:

```
<p>The message is "((${msg}))"</p>
```

The result will have those `` tags unescaped, so:

```
<p>The message is "This is <b>great!</b>"</p>
```

Whereas if escaped like:

```
<p>The message is "[[${msg}]]"</p>
```

The result will be HTML-escaped:

```
<p>The message is "This is &lt;b&gt;great!&lt;/b&gt;"</p>
```

Note that **text inlining is active by default** in the body of every tag in our markup -- not the tags themselves --, so there is nothing we need to do to enable it.

Inlining vs natural templates

If you come from other template engines in which this way of outputting text is the norm, you might be asking: *Why aren't we doing this from the beginning? It's less code than all those th:text attributes!*

Well, be careful there, because although you might find inlining quite interesting, you should always remember that inlined expressions will be displayed verbatim in your HTML files when you open them statically, so you probably won't be able to use them as design prototypes anymore!

The difference between how a browser would statically display our fragment of code without using inlining...

```
Hello, Sebastian!
```

...and using it...

```
Hello, [[${session.user.name}]]!
```

...is quite clear in terms of design usefulness.

Disabling inlining

This mechanism can be disabled though, because there might actually be occasions in which we do want to output the `[[...]]` or `[(...)]` sequences without its contents being processed as an expression. For that, we will use `th:inline="none"`:

```
<p th:inline="none">A double array looks like this: [[1, 2, 3], [4, 5]]!</p>
```

This will result in:

```
<p>A double array looks like this: [[1, 2, 3], [4, 5]]!</p>
```

12.2 Text inlining

Text inlining is very similar to the *expression inlining* capability we have just seen, but it actually adds more power. It has to be enabled explicitly with `th:inline="text"`.

Text inlining not only allows us to use the same *inlined expressions* we just saw, but in fact processes *tag bodies* as if they were templates processed in the `TEXT` template mode, which allows us to perform text-based template logic (not only output expressions).

We will see more about this in the next chapter about the *textual template modes*.

12.3 JavaScript inlining

JavaScript inlining allows for a better integration of JavaScript `<script>` blocks in templates being processed in the `HTML` template mode.

As with *text inlining*, this is actually equivalent to processing the scripts contents as if they were templates in the `JAVASCRIPT` template mode, and therefore all the power of the *textual template modes* (see next chapter) will be at hand. However, in this section we will focus on how we can use it for adding the output of our Thymeleaf expressions into our JavaScript blocks.

This mode has to be explicitly enabled using `th:inline="javascript"`:

```
<script th:inline="javascript">
    ...
    var username = [[${session.user.name}]];
    ...
</script>
```

This will result in:

```
<script th:inline="javascript">
...
var username = "Sebastian \"Fruity\" Applejuice";
...
</script>
```

Two important things to note in the code above:

First, that JavaScript inlining will not only output the required text, but also enclose it with quotes and JavaScript-escape its contents, so that the expression results are output as a well-formed JavaScript literal.

Second, that this is happening because we are outputting the `${session.user.name}` expression as **escaped**, i.e. using a double-bracket expression: `[$ ${session.user.name}]]`. If instead we used **unesaped** like:

```
<script th:inline="javascript">
...
var username = [(${session.user.name})];
...
</script>
```

The result would look like:

```
<script th:inline="javascript">
...
var username = Sebastian "Fruity" Applejuice;
...
</script>
```

...which is malformed JavaScript code. But outputting something unescaped might be what we need if we are building parts of our script by means of appending inlined expressions, so it's good to have this tool at hand.

JavaScript natural templates

The mentioned *intelligence* of the JavaScript inlining mechanism goes much further than just applying JavaScript-specific escaping and outputting expression results as valid literals.

For example, we can wrap our (escaped) inlined expressions in JavaScript comments like:

```
<script th:inline="javascript">
...
var username = /*[$ ${session.user.name}]*/ "Gertrud Kiwifruit";
...
</script>
```

And Thymeleaf will ignore everything we have written *after the comment and before the semicolon* (in this case '`Gertrud Kiwifruit'`), so the result of executing this will look exactly like when we were not using the wrapping comments:

```
<script th:inline="javascript">
...
var username = "Sebastian \"Fruity\" Applejuice";
...
</script>
```

But have another careful look at the original template code:

```
<script th:inline="javascript">
    ...
    var username = /*[[${session.user.name}]]*/ "Gertrud Kiwifruit";
    ...
</script>
```

Note how this is **valid JavaScript code**. And it will perfectly execute when you open your template file in a static manner (without executing it at a server).

So what we have here is a way to do **JavaScript natural templates!**

Advanced inlined evaluation and JavaScript serialization

An important thing to note regarding JavaScript inlining is that this expression evaluation is intelligent and not limited to Strings. Thymeleaf will correctly write in JavaScript syntax the following kinds of objects:

- Strings
- Numbers
- Booleans
- Arrays
- Collections
- Maps
- Beans (objects with *getter* and *setter* methods)

For example, if we had the following code:

```
<script th:inline="javascript">
    ...
    var user = /*[[${session.user}]]*/ null;
    ...
</script>
```

That `${session.user}` expression will evaluate to a `User` object, and Thymeleaf will correctly convert it to Javascript syntax:

```
<script th:inline="javascript">
    ...
    var user = {"age":null,"firstName":"John","lastName":"Apricot",
                "name":"John Apricot","nationality":"Antarctica"};
    ...
</script>
```

The way this JavaScript serialization is done is by means of an implementation of the `org.thymeleaf.standard.serializer.IStandardJavaScriptSerializer` interface, which can be configured at the instance of the `StandardDialect` being used at the template engine.

The default implementation of this JS serialization mechanism will look for the [Jackson library](#) in the classpath and, if present, will use it. If not, it will apply a built-in serialization mechanism that covers the needs of most scenarios and produces similar results (but is less flexible).

12.4 CSS inlining

Thymeleaf also allows the use of inlining in CSS `<style>` tags, such as:

```
<style th:inline="css">
  ...
</style>
```

For example, say we have two variables set to two different `String` values:

```
classname = 'main elems'
align = 'center'
```

We could use them just like:

```
<style th:inline="css">
  .[${classname}] {
    text-align: ${align};
  }
</style>
```

And the result would be:

```
<style th:inline="css">
  .main\ elems {
    text-align: center;
  }
</style>
```

Note how CSS inlining also bears some *intelligence*, just like JavaScript's. Specifically, expressions output via *escaped* expressions like `[$(classname)]` will be escaped as **CSS identifiers**. That is why our `classname = 'main elems'` has turned into `main\ elems` in the fragment of code above.

Advanced features: CSS natural templates, etc.

In an equivalent way to what was explained before for JavaScript, CSS inlining also allows for our `<style>` tags to work both statically and dynamically, i.e. as **CSS natural templates** by means of wrapping inlined expressions in comments.

See:

```
<style th:inline="css">
  .main\ elems {
    text-align: /*[${align}]*/ left;
  }
</style>
```

13 Textual template modes

13.1 Textual syntax

Three of the Thymeleaf *template modes* are considered **textual**: `TEXT`, `JAVASCRIPT` and `CSS`. This differentiates them from the markup template modes: `HTML` and `XML`.

The key difference between *textual* template modes and the markup ones is that in a textual template there are no tags into which to insert logic in the form of attributes, so we have to rely on other mechanisms.

The first and most basic of these mechanisms is **inlining**, which we have already detailed in the previous chapter. Inlining syntax is the most simple way to output results of expressions in textual template mode, so this is a perfectly valid template for a text email.

```
Dear [(${name})],  
  
Please find attached the results of the report you requested  
with name "[(${report.name})]".  
  
Sincerely,  
The Reporter.
```

Even without tags, the example above is a complete and valid Thymeleaf template that can be executed in the `TEXT` template mode.

But in order to include more complex logic than mere *output expressions*, we need a new non-tag-based syntax:

```
[# th:each="item : ${items}"]  
  - [(${item})]  
[/]
```

Which is actually the *condensed* version of the more verbose:

```
[#th:block th:each="item : ${items}"]  
  - [#th:block th:utext="${item}" /]  
[/th:block]
```

Note how this new syntax is based on elements (i.e. processable tags) that are declared as `[#element ...]` instead of `<element ...>`. Elements are open like `[#element ...]` and closed like `[/element]`, and standalone tags can be declared by minimizing the open element with a `/` in a way almost equivalent to XML tags: `[#element ... /]`.

The Standard Dialect only contains a processor for one of these elements: the already-known `th:block`, though we could extend this in our dialects and create new elements in the usual way. Also, the `th:block` element (`[#th:block ...] ... [/th:block]`) is allowed to be abbreviated as the empty string (`[# ...] ... [/]`), so the above block is actually equivalent to:

```
[# th:each="item : ${items}"]  
  - [# th:utext="${item}" /]  
[/]
```

And given `[# th:utext="${item}" /]` is equivalent to an *inlined unescaped expression*, we could just use it in order to have less code. Thus we end up with the first fragment of code we saw above:

```
[# th:each="item : ${items}"]
- [(${item})]
[]
```

Note that the *textual syntax requires full element balance (no unclosed tags) and quoted attributes* – it's more XML-style than HTML-style.

Let's have a look at a more complete example of a `TEXT` template, a *plain text* email template:

```
Dear [${customer.name}],
This is the list of our products:
[# th:each="prod : ${products}"]
- [${prod.name}]. Price: [${prod.price}] EUR/kg
[]

Thanks,
The Thymeleaf Shop
```

After executing, the result of this could be something like:

```
Dear Mary Ann Blueberry,
This is the list of our products:
- Apricots. Price: 1.12 EUR/kg
- Bananas. Price: 1.78 EUR/kg
- Apples. Price: 0.85 EUR/kg
- Watermelon. Price: 1.91 EUR/kg

Thanks,
The Thymeleaf Shop
```

And another example in `JAVASCRIPT` template mode, a `greeter.js` file, we process as a textual template and which result we call from our HTML pages. Note this is *not* a `<script>` block in an HTML template, but a `.js` file being processed as a template on its own:

```
var greeter = function() {
    var username = [[${session.user.name}]];
    [# th:each="salut : ${salutations}"]
        alert([[${salut}]] + " " + username);
    []
};

};
```

After executing, the result of this could be something like:

```
var greeter = function() {
    var username = "Bertrand \\"Crunchy\\" Pear";
    alert("Hello" + " " + username);
    alert("Ol\u00e9" + " " + username);
    alert("Hola" + " " + username);
};

};
```

Escaped element attributes

In order to avoid interactions with parts of the template that might be processed in other modes (e.g. `text`-mode inlining inside an `HTML` template), Thymeleaf 3.0 allows the attributes in elements in its *textual syntax* to be escaped. So:

- Attributes in `TEXT` template mode will be *HTML-unescaped*.
- Attributes in `JAVASCRIPT` template mode will be *JavaScript-unescaped*.
- Attributes in `CSS` template mode will be *CSS-unescaped*.

So this would be perfectly OK in a `TEXT`-mode template (note the `>`):

```
[# th:if="${120<user.age}"]
    Congratulations!
[/]
```

Of course that `<` would make no sense in a *real text* template, but it is a good idea if we are processing an `HTML` template with a `th:inline="text"` block containing the code above and we want to make sure our browser doesn't take that `<user.age` for the name of an open tag when statically opening the file as a prototype.

13.2 Extensibility

One of the advantages of this syntax is that it is just as extensible as the *markup* one. Developers can still define their own dialects with custom elements and attributes, apply a prefix to them (optionally), and then use them in textual template modes:

```
[#myorg:dosomething myorg:importantattr="211"]some text[/myorg:dosomething]
```

13.3 Textual prototype-only comment blocks: adding code

The `JAVASCRIPT` and `CSS` template modes (not available for `TEXT`) allow including code between a special comment syntax `/*[+...+]*/` so that Thymeleaf will automatically uncomment such code when processing the template:

```
var x = 23;
/*[+
var msg = "This is a working application";
+]*/
var f = function() {
...
```

Will be executed as:

```
var x = 23;
var msg = "This is a working application";
var f = function() {
...
```

You can include expressions inside these comments, and they will be evaluated:

```
var x = 23;  
/*[+  
var msg = "Hello, " + [[${session.user.name}]];  
+]*/  
var f = function() {  
...  
}
```

13.4 Textual parser-level comment blocks: removing code

In a way similar to that of prototype-only comment blocks, all the three textual template modes (TEXT , JAVASCRIPT and CSS) make it possible to instruct Thymeleaf to remove code between special /*[- */ and /* -]*/ marks, like this:

```
var x = 23;  
/*[- */  
var msg = "This is shown only when executed statically!";  
/* -]*/  
var f = function() {  
...  
}
```

Or this, in TEXT mode:

```
...  
/*[- Note the user is obtained from the session, which must exist -]*/  
Welcome [(${session.user.name})]!  
...
```

13.5 Natural JavaScript and CSS templates

As seen in the previous chapter, JavaScript and CSS inlining offer the possibility to include inlined expressions inside JavaScript/CSS comments, like:

```
...  
var username = /*[[${session.user.name}]]*/ "Sebastian Lychee";  
...
```

...which is valid JavaScript, and once executed could look like:

```
...  
var username = "John Apricot";  
...
```

This same *trick* of enclosing inlined expressions inside comments can in fact be used for the entire textual mode

syntax:

```
/*[# th:if="${user.admin}"]*/
    alert('Welcome admin');
/*[/]*/
```

That alert in the code above will be shown when the template is open statically – because it is 100% valid JavaScript –, and also when the template is run if the user is an admin. It is equivalent to:

```
[# th:if="${user.admin}"]
    alert('Welcome admin');
[]/
```

...which is actually the code to which the initial version is converted during template parsing.

Note however that wrapping elements in comments does not clean the lines they live in (to the right until a ; is found) as inlined output expressions do. That behaviour is reserved for inlined output expressions only.

So Thymeleaf 3.0 allows the development of **complex JavaScript scripts and CSS style sheets in the form of natural templates**, valid both as a *prototype* and as a *working template*.

14 Some more pages for our grocery

Now we know a lot about using Thymeleaf, we can add some new pages to our website for order management.

Note that we will focus on HTML code, but you can have a look at the bundled source code if you want to see the corresponding controllers.

14.1 Order List

Let's start by creating an order list page, /WEB-INF/templates/order/list.html:

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

    <head>
        <title>Good Thymes Virtual Grocery</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link rel="stylesheet" type="text/css" media="all"
              href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />
    </head>

    <body>
        <h1>Order list</h1>

        <table>
            <tr>
                <th>DATE</th>
                <th>CUSTOMER</th>
                <th>TOTAL</th>
                <th></th>
            </tr>
            <tr th:each="o : ${orders}" th:class="${oStat.odd}? 'odd'">
                <td th:text="#{calendars.format(o.date,'dd/MMM/yyyy')}">13 jan 2011</td>
                <td th:text="${o.customer.name}">Frederic Tomato</td>
                <td th:text="#{aggregates.sum(o.orderLines.{purchasePrice * amount})}">23.32</td>
                <td>
                    <a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>
                </td>
            </tr>
        </table>

        <p>
            <a href="../home.html" th:href="@{}>Return to home</a>
        </p>
    </body>
</html>
```

There's nothing here that should surprise us, except for this little bit of OGNL magic:

```
<td th:text="#{aggregates.sum(o.orderLines.{purchasePrice * amount})}">23.32</td>
```

What that does is, for each order line (`OrderLine` object) in the order, multiply its `purchasePrice` and `amount`

properties (by calling the corresponding `getPurchasePrice()` and `getAmount()` methods) and return the result into a list of numbers, later aggregated by the `#aggregates.sum(...)` function in order to obtain the order total price.

You've got to love the power of OGNL.

14.2 Order Details

Now for the order details page, in which we will make a heavy use of asterisk syntax:

```

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

    <head>
        <title>Good Thymes Virtual Grocery</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link rel="stylesheet" type="text/css" media="all"
              href="../../css/gtvg.css" th:href="@{/css/gtvg.css}" />
    </head>

    <body th:object="${order}">

        <h1>Order details</h1>

        <div>
            <p><b>Code:</b> <span th:text="*{id}">99</span></p>
            <p>
                <b>Date:</b>
                <span th:text="#{calendars.format(date, 'dd MMM yyyy')}">13 jan 2011</span>
            </p>
        </div>

        <h2>Customer</h2>

        <div th:object="*{customer}">
            <p><b>Name:</b> <span th:text="*{name}">Frederic Tomato</span></p>
            <p>
                <b>Since:</b>
                <span th:text="#{calendars.format(customerSince, 'dd MMM yyyy')}">1 jan 2011</span>
            </p>
        </div>

        <h2>Products</h2>

        <table>
            <tr>
                <th>PRODUCT</th>
                <th>AMOUNT</th>
                <th>PURCHASE PRICE</th>
            </tr>
            <tr th:each="ol, row : *{orderLines}" th:class="${row.odd} ? 'odd'">
                <td th:text="${ol.product.name}">Strawberries</td>
                <td th:text="${ol.amount}" class="number">3</td>
                <td th:text="${ol.purchasePrice}" class="number">23.32</td>
            </tr>
        </table>

        <div>
            <b>TOTAL:</b>
            <span th:text="#{aggregates.sum(orderLines.{purchasePrice * amount})}">35.23</span>
        </div>

        <p>
            <a href="list.html" th:href="@{/order/list}">Return to order list</a>
        </p>
    </body>
</html>

```

Not much really new here, except for this nested object selection:

```
<body th:object="${order}">  
    ...  
  
    <div th:object="*[customer]">  
        <p><b>Name:</b> <span th:text="*{name}">Frederic Tomato</span></p>  
        ...  
    </div>  
  
    ...  
</body>
```

...which makes that `*{name}` equivalent to:

```
<p><b>Name:</b> <span th:text="${order.customer.name}">Frederic Tomato</span></p>
```

15 More on Configuration

15.1 Template Resolvers

For our Good Thymes Virtual Grocery, we chose an `ITemplateResolver` implementation called `ServletContextTemplateResolver` that allowed us to obtain templates as resources from the Servlet Context.

Besides giving us the ability to create our own template resolver by implementing `ITemplateResolver`, Thymeleaf includes four implementations out of the box:

- `org.thymeleaf.templateresolver.ClassLoaderTemplateResolver`, which resolves templates as classloader resources, like:

```
return Thread.currentThread().getContextClassLoader().getResourceAsStream(template);
```

- `org.thymeleaf.templateresolver.FileTemplateResolver`, which resolves templates as files from the file system, like:

```
return new FileInputStream(new File(template));
```

- `org.thymeleaf.templateresolver.UrlTemplateResolver`, which resolves templates as URLs (even non-local ones), like:

```
return (new URL(template)).openStream();
```

- `org.thymeleaf.templateresolver.StringTemplateResolver`, which resolves templates directly as the `String` being specified as `template` (or *template name*, which in this case is obviously much more than a mere name):

```
return new StringReader(templateName);
```

All of the pre-bundled implementations of `ITemplateResolver` allow the same set of configuration parameters, which include:

- Prefix and suffix (as already seen):

```
templateResolver.setPrefix("/WEB-INF/templates/");
templateResolver.setSuffix(".html");
```

- Template aliases that allow the use of template names that do not directly correspond to file names. If both suffix/prefix and alias exist, alias will be applied before prefix/suffix:

```
templateResolver.addTemplateAlias("adminHome","profiles/admin/home");
templateResolver.setTemplateAliases(aliasesMap);
```

- Encoding to be applied when reading templates:

```
templateResolver.setEncoding("UTF-8");
```

- Template mode to be used:

```
// Default is HTML  
templateResolver.setTemplateMode("XML");
```

- Default mode for template cache, and patterns for defining whether specific templates are cacheable or not:

```
// Default is true  
templateResolver.setCacheable(false);  
templateResolver.getCacheablePatternSpec().addPattern("/users/*");
```

- TTL in milliseconds for parsed template cache entries originated in this template resolver. If not set, the only way to remove an entry from the cache will be to exceed the cache max size (oldest entry will be removed).

```
// Default is no TTL (only cache size exceeded would remove entries)  
templateResolver.setCacheTTLms(60000L);
```

The Thymeleaf + Spring integration packages offer a `SpringResourceTemplateResolver` implementation which uses all the Spring infrastructure for accessing and reading resources in applications, and which is the recommended implementation in Spring-enabled applications.

Chaining Template Resolvers

Also, a Template Engine can specify several template resolvers, in which case an order can be established between them for template resolution so that, if the first one is not able to resolve the template, the second one is asked, and so on:

```
ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();  
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));  
  
ServletContextTemplateResolver servletContextTemplateResolver =  
    new ServletContextTemplateResolver(servletContext);  
servletContextTemplateResolver.setOrder(Integer.valueOf(2));  
  
templateEngine.addTemplateResolver(classLoaderTemplateResolver);  
templateEngine.addTemplateResolver(servletContextTemplateResolver);
```

When several template resolvers are applied, it is recommended to specify patterns for each template resolver so that Thymeleaf can quickly discard those template resolvers that are not meant to resolve the template, enhancing performance. Doing this is not a requirement, but a recommendation:

```
ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();  
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));  
// This classloader will not be even asked for any templates not matching these patterns  
classLoaderTemplateResolver.getResolvablePatternSpec().addPattern("/layout/*.html");  
classLoaderTemplateResolver.getResolvablePatternSpec().addPattern("/menu/*.html");  
  
ServletContextTemplateResolver servletContextTemplateResolver =  
    new ServletContextTemplateResolver(servletContext);  
servletContextTemplateResolver.setOrder(Integer.valueOf(2));
```

If these *resolvable patterns* are not specified, we will be relying on the specific capabilities of each of the `ITemplateResolver` implementations we are using. Note that not all implementations might be able to determine the existence of a template before resolving, and thus could always consider a template as *resolvable* and break the resolution chain (not allowing other resolvers to check for the same template), but then be unable to read the real resource.

All the `ITemplateResolver` implementations that are included with core Thymeleaf include a mechanism that will allow us to make the resolvers *really check* if a resource exists before considering it *resolvable*. It is the `checkExistence` flag, which works like:

```
ClassLoaderTemplateResolver classLoaderTemplateResolver = new ClassLoaderTemplateResolver();
classLoaderTemplateResolver.setOrder(Integer.valueOf(1));
classLoaderTemplateResolver.setCheckExistence(true);
```

This `checkExistence` flag forces the resolver perform a *real check* for resource existence during the resolution phase (and let the following resolver in the chain be called if existence check returns false). While this might sound good in every case, in most cases this will mean a double access to the resource itself (once for checking existence, another time for reading it), and could be a performance issue in some scenarios, e.g. remote URL-based template resources – a potential performance issue that might anyway get largely mitigated by the use of the template cache (in which case templates will only be *resolved* the first time they are accessed).

15.2 Message Resolvers

We did not explicitly specify a Message Resolver implementation for our Grocery application, and as it was explained before, this meant that the implementation being used was an `org.thymeleaf.messageresolver.StandardMessageResolver` object.

`StandardMessageResolver` is the standard implementation of the `IMessageResolver` interface, but we could create our own if we wanted, adapted to the specific needs of our application.

The Thymeleaf + Spring integration packages offer by default an `IMessageResolver` implementation which uses the standard Spring way of retrieving externalized messages, by using `MessageSource` beans declared at the Spring Application Context.

Standard Message Resolver

So how does `StandardMessageResolver` look for the messages requested at a specific template?

If the template name is `home` and it is located in `/WEB-INF/templates/home.html`, and the requested locale is `gl_ES` then this resolver will look for messages in the following files, in this order:

- `/WEB-INF/templates/home_gl_ES.properties`
- `/WEB-INF/templates/home_gl.properties`
- `/WEB-INF/templates/home.properties`

Refer to the JavaDoc documentation of the `StandardMessageResolver` class for more detail on how the complete message resolution mechanism works.

Configuring message resolvers

What if we wanted to add a message resolver (or more) to the Template Engine? Easy:

```
// For setting only one
templateEngine.setMessageResolver(messageResolver);

// For setting more than one
templateEngine.addMessageResolver(messageResolver);
```

And why would we want to have more than one message resolver? For the same reason as template resolvers: message resolvers are ordered and if the first one cannot resolve a specific message, the second one will be asked, then the third, etc.

15.3 Conversion Services

The *conversion service* that enables us to perform data conversion and formatting operations by means of the *double-brace* syntax (`${{{...}}}`) is actually a feature of the Standard Dialect, not of the Thymeleaf Template Engine itself.

As such, the way to configure it is by setting our custom implementation of the `IStandardConversionService` interface directly into the instance of `StandardDialect` that is being configured into the template engine. Like:

```
IStandardConversionService customConversionService = ...  
  
StandardDialect dialect = new StandardDialect();  
dialect.setConversionService(customConversionService);  
  
templateEngine.setDialect(dialect);
```

Note that the `thymeleaf-spring3` and `thymeleaf-spring4` packages contain the `SpringStandardDialect`, and this dialect already comes pre-configured with an implementation of `IStandardConversionService` that integrates Spring's own *Conversion Service* infrastructure into Thymeleaf.

15.4 Logging

Thymeleaf pays quite a lot of attention to logging, and always tries to offer the maximum amount of useful information through its logging interface.

The logging library used is `slf4j`, which in fact acts as a bridge to whichever logging implementation we might want to use in our application (for example, `log4j`).

Thymeleaf classes will log `TRACE`, `DEBUG` and `INFO`-level information, depending on the level of detail we desire, and besides general logging it will use three special loggers associated with the `TemplateEngine` class which we can configure separately for different purposes:

- `org.thymeleaf.TemplateEngine.CONFIG` will output detailed configuration of the library during initialization.
- `org.thymeleaf.TemplateEngine.TIMER` will output information about the amount of time taken to process each template (useful for benchmarking!)
- `org.thymeleaf.TemplateEngine.cache` is the prefix for a set of loggers that output specific information about the caches. Although the names of the cache loggers are configurable by the user and thus could change, by default they are:
 - `org.thymeleaf.TemplateEngine.cache TEMPLATE_CACHE`
 - `org.thymeleaf.TemplateEngine.cache EXPRESSION_CACHE`

An example configuration for Thymeleaf's logging infrastructure, using `log4j`, could be:

```
log4j.logger.org.thymeleaf=DEBUG  
log4j.logger.org.thymeleaf.TemplateEngine.CONFIG=TRACE  
log4j.logger.org.thymeleaf.TemplateEngine.TIMER=TRACE  
log4j.logger.org.thymeleaf.TemplateEngine.cache TEMPLATE_CACHE=TRACE
```

16 Template Cache

Thymeleaf works thanks to a set of parsers – for markup and text – that parse templates into sequences of events (open tag, text, close tag, comment, etc.) and a series of processors – one for each type of behaviour that needs to be applied – that modify the template parsed event sequence in order to create the results we expect by combining the original template with our data.

It also includes – by default – a cache that stores parsed templates; the sequence of events resulting from reading and parsing template files before processing them. This is especially useful when working in a web application, and builds on the following concepts:

- Input/Output is almost always the slowest part of any application. In-memory processing is extremely quick by comparison.
- Cloning an existing in-memory event sequence is always much quicker than reading a template file, parsing it and creating a new event sequence for it.
- Web applications usually have only a few dozen templates.
- Template files are small-to-medium size, and they are not modified while the application is running.

This all leads to the idea that caching the most used templates in a web application is feasible without wasting large amounts of memory, and also that it will save a lot of time that would be spent on input/output operations on a small set of files that, in fact, never change.

And how can we take control of this cache? First, we've learned before that we can enable or disable it at the Template Resolver, even acting only on specific templates:

```
// Default is true
templateResolver.setCacheable(false);
templateResolver.getCacheablePatternSpec().addPattern("/users/*");
```

Also, we could modify its configuration by establishing our own *Cache Manager* object, which could be an instance of the default `StandardCacheManager` implementation:

```
// Default is 200
StandardCacheManager cacheManager = new StandardCacheManager();
cacheManager.setTemplateCacheMaxSize(100);
...
templateEngine.setCacheManager(cacheManager);
```

Refer to the javadoc API of `org.thymeleaf.cache.StandardCacheManager` for more info on configuring the caches.

Entries can be manually removed from the template cache:

```
// Clear the cache completely
templateEngine.clearTemplateCache();

// Clear a specific template from the cache
templateEngine.clearTemplateCacheFor("/users/userList");
```

17 Decoupled Template Logic

17.1 Decoupled logic: The concept

So far we have worked for our Grocery Store with templates done the *usual way*, with logic being inserted into our templates in the form of attributes.

But Thymeleaf also allows us to completely *decouple* the template markup from its logic, allowing the creation of **completely logic-less markup templates** in the `HTML` and `XML` template modes.

The main idea is that template logic will be defined in a separate *logic file* (more exactly a *logic resource*, as it doesn't need to be a *file*). By default, that logic resource will be an additional file living in the same place (e.g. folder) as the template file, with the same name but with `.th.xml` extension:

```
/templates  
+->/home.html  
+->/home.th.xml
```

So the `home.html` file can be completely logic-less. It might look like this:

```
<!DOCTYPE html>  
<html>  
  <body>  
    <table id="usersTable">  
      <tr>  
        <td class="username">Jeremy Grapefruit</td>  
        <td class="usertype">Normal User</td>  
      </tr>  
      <tr>  
        <td class="username">Alice Watermelon</td>  
        <td class="usertype">Administrator</td>  
      </tr>  
    </table>  
  </body>  
</html>
```

Absolutely no Thymeleaf code there. This is a template file that a designer with no Thymeleaf or templating knowledge could have created, edited and/or understood. Or a fragment of HTML provided by some external system with no Thymeleaf hooks at all.

Let's now turn that `home.html` template into a Thymeleaf template by creating our additional `home.th.xml` file like this:

```
<?xml version="1.0"?>  
<thlogic>  
  <attr sel="#usersTable" th:remove="all-but-first">  
    <attr sel="/tr[0]" th:each="user : ${users}">  
      <attr sel="td.username" th:text="${user.name}" />  
      <attr sel="td.usertype" th:text="#{|user.type.${user.type}|}" />  
    </attr>  
  </attr>  
</thlogic>
```

Here we can see a lot of `<attr>` tags inside a `thlogic` block. Those `<attr>` tags perform *attribute injection* on nodes of the original template selected by means of their `sel` attributes, which contain Thymeleaf *markup selectors* (actually

AttoParser markup selectors).

Also note that `<attr>` tags can be nested so that their selectors are *appended*. That `sel="/tr[0]"` above, for example, will be processed as `sel="#usersTable/tr[0]"`. And the selector for the user name `<td>` will be processed as `sel="#usersTable/tr[0]//td.username"`.

So once merged, both files seen above will be the same as:

```
<!DOCTYPE html>
<html>
  <body>
    <table id="usersTable" th:remove="all-but-first">
      <tr th:each="user : ${users}">
        <td class="username" th:text="${user.name}">Jeremy Grapefruit</td>
        <td class="usertype" th:text="#{|user.type.${user.type}|}">Normal User</td>
      </tr>
      <tr>
        <td class="username">Alice Watermelon</td>
        <td class="usertype">Administrator</td>
      </tr>
    </table>
  </body>
</html>
```

This looks more familiar, and is indeed less *verbose* than creating two separate files. But the advantage of *decoupled templates* is that we can give our templates total independence from Thymeleaf, and therefore better maintainability from the design standpoint.

Of course some *contracts* between designers or developers will still be needed – e.g. the fact that the users `<table>` will need an `id="usersTable"` –, but in many scenarios a pure-HTML template will be a much better communication artifact between design and development teams.

17.2 Configuring decoupled templates

Enabling decoupled templates

Decoupled logic will not be expected for every template by default. Instead, the configured template resolvers (implementations of `ITemplateResolver`) will need to specifically mark the templates they resolve as *using decoupled logic*.

Except for `StringTemplateResolver` (which does not allow decoupled logic), all other out-of-the-box implementations of `ITemplateResolver` will provide a flag called `useDecoupledLogic` that will mark all templates resolved by that resolver as potentially having all or part of its logic living in a separate resource:

```
final ServletContextTemplateResolver templateResolver =
  new ServletContextTemplateResolver(servletContext);
...
templateResolver.setUseDecoupledLogic(true);
```

Mixing coupled and decoupled logic

Decoupled template logic, when enabled, is not a requirement. When enabled, it means that the engine will *look for* a resource containing decoupled logic, parsing and merging it with the original template if it exists. No error will be thrown if the decoupled logic resource does not exist.

Also, in the same template we can mix both *coupled* and *decoupled* logic, for example by adding some Thymeleaf

attributes at the original template file but leaving others for the separate decoupled logic file. The most common case for this is using the new (in v3.0) `th:ref` attribute.

17.3 The `th:ref` attribute

`th:ref` is only a marker attribute. It does nothing from the processing standpoint and simply disappears when the template is processed, but its usefulness lies in the fact that it acts as a *markup reference*, i.e. it can be resolved by name from a *markup selector* just like a *tag name* or a *fragment* (`th:fragment`).

So if we have a selector like:

```
<attr sel="whatever" .../>
```

This will match:

- Any `<whatever>` tags.
- Any tags with a `th:fragment="whatever"` attribute.
- Any tags with a `th:ref="whatever"` attribute.

What is the advantage of `th:ref` against, for example, using a pure-HTML `id` attribute? Merely the fact that we might not want to add so many `id` and `class` attributes to our tags to act as *logic anchors*, which might end up *polluting* our output.

And in the same sense, what is the disadvantage of `th:ref`? Well, obviously that we'd be adding a bit of Thymeleaf logic ("logic") to our templates.

Note this applicability of the `th:ref` attribute **does not only apply to decoupled logic template files**: it works the same in other types of scenarios, like in fragment expressions (`~{...}`).

17.4 Performance impact of decoupled templates

The impact is extremely small. When a resolved template is marked to use decoupled logic and it is not cached, the template logic resource will be resolved first, parsed and processed into a sequence of instructions in-memory: basically a list of attributes to be injected to each markup selector.

But this is the only *additional step* required because, after this, the real template will be parsed, and while it is parsed these attributes will be injected *on-the-fly* by the parser itself, thanks to the advanced capabilities for node selection in AttoParser. So parsed nodes will come out of the parser as if they had their injected attributes written in the original template file.

The biggest advantage of this? When a template is configured to be cached, it will be cached already containing the injected attributes. So the overhead of using *decoupled templates* for cacheable templates, once they are cached, will be absolutely *zero*.

17.5 Resolution of decoupled logic

The way Thymeleaf resolves the decoupled logic resources corresponding to each template is configurable by the user. It is determined by an extension point, the

`org.thymeleaf.templateparser.markup.Decoupled.IDecoupledTemplateLogicResolver`, for which a *default implementation* is provided: `StandardDecoupledTemplateLogicResolver`.

What does this standard implementation do?

- First, it applies a `prefix` and a `suffix` to the *base name* of the template resource (obtained by means of its `ITemplateResource#getBaseName()` method). Both prefix and suffix can be configured and, by default, the prefix will be empty and the suffix will be `.th.xml`.
- Second, it asks the template resource to resolve a *relative resource* with the computed name by means of its `ITemplateResource#relative(String relativeLocation)` method.

The specific implementation of `IDecoupledTemplateLogicResolver` to be used can be configured at the `TemplateEngine` easily:

```
final StandardDecoupledTemplateLogicResolver decoupledresolver =
    new StandardDecoupledTemplateLogicResolver();
decoupledresolver.setPrefix("../viewlogic/");
...
templateEngine.setDecoupledTemplateLogicResolver(decoupledresolver);
```

18 Appendix A: Expression Basic Objects

Some objects and variable maps are always available to be invoked. Let's see them:

Base objects

- **#ctx**: the context object. An implementation of `org.thymeleaf.context.IContext` or `org.thymeleaf.context.IWebContext` depending on our environment (standalone or web).

Note `#vars` and `#root` are synomyns for the same object, but using `#ctx` is recommended.

```
/*
 * =====
 * See javadoc API for class org.thymeleaf.context.IContext
 * =====
 */

${#ctx.locale}
${#ctx.variableNames}

/*
 * =====
 * See javadoc API for class org.thymeleaf.context.IWebContext
 * =====
 */

${#ctx.request}
${#ctx.response}
${#ctx.session}
${#ctx.servletContext}
```

- **#locale**: direct access to the `java.util.Locale` associated with current request.

```
${#locale}
```

Web context namespaces for request/session attributes, etc.

When using Thymeleaf in a web environment, we can use a series of shortcuts for accessing request parameters, session attributes and application attributes:

Note these are not *context objects*, but maps added to the context as variables, so we access them without `#`. In some way, they act as *namespaces*.

- **param**: for retrieving request parameters. `${param.foo}` is a `String[]` with the values of the `foo` request parameter, so `${param.foo[0]}` will normally be used for getting the first value.

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.context.WebRequestParamsVariablesMap
 * =====
 */

${param.foo}          // Retrieves a String[] with the values of request parameter 'foo'
${param.size()}
${param.isEmpty()}
${param.containsKey('foo')}
...

```

- **session** : for retrieving session attributes.

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.context.WebSessionVariablesMap
 * =====
 */

${session.foo}        // Retrieves the session attribute 'foo'
${session.size()}
${session.isEmpty()}
${session.containsKey('foo')}
...

```

- **application** : for retrieving application/servlet context attributes.

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.context.WebServletContextVariablesMap
 * =====
 */

${application.foo}    // Retrieves the ServletContext attribute 'foo'
${application.size()}
${application.isEmpty()}
${application.containsKey('foo')}
...

```

Note there is **no need to specify a namespace for accessing request attributes** (as opposed to *request parameters*) because all request attributes are automatically added to the context as variables in the context root:

```

${myRequestAttribute}

```

Web context objects

Inside a web environment there is also direct access to the following objects (note these are objects, not maps/namespaces):

- **#request** : direct access to the `javax.servlet.http.HttpServletRequest` object associated with the current request.

```

${#request.getAttribute('foo')}
${#request.getParameter('foo')}
${#request.getContextPath()}
${#request.getRequestName()}
...

```

- **#session** : direct access to the `javax.servlet.http.HttpSession` object associated with the current request.

```
 ${#session.getAttribute('foo')}\n${#session.id}\n${#session.lastAccessedTime}\n...
```

- **#servletContext** : direct access to the `javax.servlet.ServletContext` object associated with the current request.

```
 ${#servletContext.getAttribute('foo')}\n${#servletContext.contextPath}\n...
```

19 Appendix B: Expression Utility Objects

Execution Info

- **#execInfo** : expression object providing useful information about the template being processed inside Thymeleaf Standard Expressions.

```
/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.ExecutionInfo
 * =====
 */

/*
 * Return the name and mode of the 'leaf' template. This means the template
 * from where the events being processed were parsed. So if this piece of
 * code is not in the root template "A" but on a fragment being inserted
 * into "A" from another template called "B", this will return "B" as a
 * name, and B's mode as template mode.
 */
${#execInfo.templateName}
${#execInfo.templateMode}

/*
 * Return the name and mode of the 'root' template. This means the template
 * that the template engine was originally asked to process. So if this
 * piece of code is not in the root template "A" but on a fragment being
 * inserted into "A" from another template called "B", this will still
 * return "A" and A's template mode.
 */
${#execInfo.processedTemplateName}
${#execInfo.processedTemplateMode}

/*
 * Return the stacks (actually, List<String> or List<TemplateMode>) of
 * templates being processed. The first element will be the
 * 'processedTemplate' (the root one), the last one will be the 'leaf'
 * template, and in the middle all the fragments inserted in nested
 * manner to reach the leaf from the root will appear.
 */
${#execInfo.templateNames}
${#execInfo.templateModes}

/*
 * Return the stack of templates being processed similarly (and in the
 * same order) to 'templateNames' and 'templateModes', but returning
 * a List<TemplateData> with the full template metadata.
 */
${#execInfo.templateStack}
```

Messages

- **#messages** : utility methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using `#{...}` syntax.

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Messages
 * =====
 */

/*
 * Obtain externalized messages. Can receive a single key, a key plus arguments,
 * or an array/list/set of keys (in which case it will return an array/list/set of
 * externalized messages).
 * If a message is not found, a default message (like '??msgKey??') is returned.
 */
${#messages.msg('msgKey')}
${#messages.msg('msgKey', param1)}
${#messages.msg('msgKey', param1, param2)}
${#messages.msg('msgKey', param1, param2, param3)}
${#messages.msgWithParams('msgKey', new Object[] {param1, param2, param3, param4})}
${#messages.arrayMsg(messageKeyArray)}
${#messages.listMsg(messageKeyList)}
${#messages.setMsg(messageKeySet)}

/*
 * Obtain externalized messages or null. Null is returned instead of a default
 * message if a message for the specified key is not found.
 */
${#messages.msgOrNull('msgKey')}
${#messages.msgOrNull('msgKey', param1)}
${#messages.msgOrNull('msgKey', param1, param2)}
${#messages.msgOrNull('msgKey', param1, param2, param3)}
${#messages.msgOrNullWithParams('msgKey', new Object[] {param1, param2, param3, param4})}
${#messages.arrayMsgOrNull(messageKeyArray)}
${#messages.listMsgOrNull(messageKeyList)}
${#messages.setMsgOrNull(messageKeySet)}

```

URIs/URLs

- **#uris** : utility object for performing URI/URL operations (esp. escaping/unescaping) inside Thymeleaf Standard Expressions.

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Uris
 * =====
 */

/*
 * Escape/Unescape as a URI/URL path
 */
${#uris.escapePath(uri)}
${#uris.escapePath(uri, encoding)}
${#uris.unescapePath(uri)}
${#uris.unescapePath(uri, encoding)}

/*
 * Escape/Unescape as a URI/URL path segment (between '/' symbols)
 */
${#uris.escapePathSegment(uri)}
${#uris.escapePathSegment(uri, encoding)}
${#uris.unescapePathSegment(uri)}
${#uris.unescapePathSegment(uri, encoding)}

/*
 * Escape/Unescape as a Fragment Identifier (#frag)
 */
${#uris.escapeFragmentId(uri)}
${#uris.escapeFragmentId(uri, encoding)}
${#uris.unescapeFragmentId(uri)}
${#uris.unescapeFragmentId(uri, encoding)}

/*
 * Escape/Unescape as a Query Parameter (?var=value)
 */
${#uris.escapeQueryParam(uri)}
${#uris.escapeQueryParam(uri, encoding)}
${#uris.unescapeQueryParam(uri)}
${#uris.unescapeQueryParam(uri, encoding)}

```

Conversions

- **#conversions** : utility object that allows the execution of the *Conversion Service* at any point of a template:

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Conversions
 * =====
 */

/*
 * Execute the desired conversion of the 'object' value into the
 * specified class.
 */
${#conversions.convert(object, 'java.util.TimeZone')}
${#conversions.convert(object, targetClass)}

```

Dates

- **#dates** : utility methods for `java.util.Date` objects:

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Dates
 * =====
 */

/*
 * Format date with the standard locale format
 * Also works with arrays, lists or sets
 */
${#dates.format(date)}
${#dates.arrayFormat(datesArray)}
${#dates.listFormat(datesList)}
${#dates.setFormat(datesSet)}

/*
 * Format date with the ISO8601 format
 * Also works with arrays, lists or sets
 */
${#dates.formatISO(date)}
${#dates.arrayFormatISO(datesArray)}
${#dates.listFormatISO(datesList)}
${#dates.setFormatISO(datesSet)}

/*
 * Format date with the specified pattern
 * Also works with arrays, lists or sets
 */
${#dates.format(date, 'dd/MMM/yyyy HH:mm')}
${#dates.arrayFormat(datesArray, 'dd/MMM/yyyy HH:mm')}
${#dates.listFormat(datesList, 'dd/MMM/yyyy HH:mm')}
${#dates.setFormat(datesSet, 'dd/MMM/yyyy HH:mm')}

/*
 * Obtain date properties
 * Also works with arrays, lists or sets
 */
${#dates.day(date)}                                // also arrayDay(...), listDay(...), etc.
${#dates.month(date)}                             // also arrayMonth(...), listMonth(...), etc.
${#dates.monthName(date)}                          // also arrayMonthName(...), listMonthName(...), etc.
${#dates.monthNameShort(date)}                     // also arrayMonthNameShort(...), listMonthNameShort(...), etc.
${#dates.year(date)}                               // also arrayYear(...), listYear(...), etc.
${#dates.dayOfWeek(date)}                         // also arrayDayOfWeek(...), listDayOfWeek(...), etc.
${#dates.dayOfWeekName(date)}                      // also arrayDayOfWeekName(...), listDayOfWeekName(...), etc.
${#dates.dayOfWeekNameShort(date)}                 // also arrayDayOfWeekNameShort(...), listDayOfWeekNameShort(...), etc.
${#dates.hour(date)}                               // also arrayHour(...), listHour(...), etc.
${#dates.minute(date)}                            // also arrayMinute(...), listMinute(...), etc.
${#dates.second(date)}                            // also arraySecond(...), listSecond(...), etc.
${#dates.millisecond(date)}                      // also arrayMillisecond(...), listMillisecond(...), etc.

/*
 * Create date (java.util.Date) objects from its components
 */
${#dates.create(year,month,day)}
${#dates.create(year,month,day,hour,minute)}
${#dates.create(year,month,day,hour,minute,second)}
${#dates.create(year,month,day,hour,minute,second,millisecond)}

/*
 * Create a date (java.util.Date) object for the current date and time
 */
${#dates.createNow()}

${#dates.createNowForTimeZone()}

/*
 * Create a date (java.util.Date) object for the current date (time set to 00:00)
*/

```

```
 */
${#dates.createToday()}

${#dates.createTodayForTimeZone()}
```

Calendars

- **#calendars** : analogous to **#dates** , but for `java.util.Calendar` objects:

```
/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Calendars
 * =====
 */

/*
 * Format calendar with the standard locale format
 * Also works with arrays, lists or sets
 */
${#calendars.format(cal)}
${#calendars.arrayFormat(calArray)}
${#calendars.listFormat(calList)}
${#calendars.setFormat(calSet)}

/*
 * Format calendar with the ISO8601 format
 * Also works with arrays, lists or sets
 */
${#calendars.formatISO(cal)}
${#calendars.arrayFormatISO(calArray)}
${#calendars.listFormatISO(calList)}
${#calendars.setFormatISO(calSet)}

/*
 * Format calendar with the specified pattern
 * Also works with arrays, lists or sets
 */
${#calendars.format(cal, 'dd/MMM/yyyy HH:mm')}
${#calendars.arrayFormat(calArray, 'dd/MMM/yyyy HH:mm')}
${#calendars.listFormat(calList, 'dd/MMM/yyyy HH:mm')}
${#calendars.setFormat(calSet, 'dd/MMM/yyyy HH:mm')}

/*
 * Obtain calendar properties
 * Also works with arrays, lists or sets
 */
${#calendars.day(date)}           // also arrayDay(...), listDay(...), etc.
${#calendars.month(date)}         // also arrayMonth(...), listMonth(...), etc.
${#calendars.monthName(date)}     // also arrayMonthName(...), listMonthName(...), etc.
${#calendars.monthNameShort(date)} // also arrayMonthNameShort(...), listMonthNameShort(...), etc.
${#calendars.year(date)}          // also arrayYear(...), listYear(...), etc.
${#calendars.dayOfWeek(date)}     // also arrayDayOfWeek(...), listDayOfWeek(...), etc.
${#calendars.dayOfWeekName(date)}  // also arrayDayOfWeekName(...), listDayOfWeekName(...), etc.
${#calendars.dayOfWeekNameShort(date)} // also arrayDayOfWeekNameShort(...), listDayOfWeekNameShort(...), etc.
${#calendars.hour(date)}          // also arrayHour(...), listHour(...), etc.
${#calendars.minute(date)}        // also arrayMinute(...), listMinute(...), etc.
${#calendars.second(date)}        // also arraySecond(...), listSecond(...), etc.
${#calendars.millisecond(date)}   // also arrayMillisecond(...), listMillisecond(...), etc.

/*
 * Create calendar (java.util.Calendar) objects from its components
 */
${#calendars.createCalendar(
```

```

${#calendars.create(year,month,day)}
${#calendars.create(year,month,day,hour,minute)}
${#calendars.create(year,month,day,hour,minute,second)}
${#calendars.create(year,month,day,hour,minute,second,millisecond)}

${#calendars.createForTimeZone(year,month,day,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,second,timeZone)}
${#calendars.createForTimeZone(year,month,day,hour,minute,second,millisecond,timeZone)}

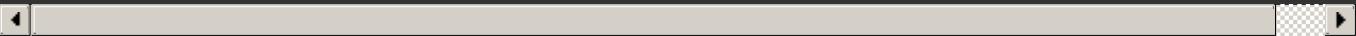
/*
 * Create a calendar (java.util.Calendar) object for the current date and time
 */
${[#calendars.createNow()]}

${[#calendars.createNowForTimeZone()]}

/*
 * Create a calendar (java.util.Calendar) object for the current date (time set to 00:00)
 */
${[#calendars.createToday()]}

${[#calendars.createTodayForTimeZone()]}

```



Numbers

- **#numbers** : utility methods for number objects:

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Numbers
 * =====
 */

/*
 * =====
 * Formatting integer numbers
 * =====
 */

/*
 * Set minimum integer digits.
 * Also works with arrays, lists or sets
 */
${[#numbers.formatInteger(num,3)]}
${[#numbers.arrayFormatInteger(numArray,3)]}
${[#numbers.listFormatInteger(numList,3)]}
${[#numbers.setFormatInteger(numSet,3)]}

/*
 * Set minimum integer digits and thousands separator:
 * 'POINT', 'COMMA', 'WHITE SPACE', 'NONE' or 'DEFAULT' (by locale).
 * Also works with arrays, lists or sets
 */
${[#numbers.formatInteger(num,3,'POINT')]}
${[#numbers.arrayFormatInteger(numArray,3,'POINT')]}
${[#numbers.listFormatInteger(numList,3,'POINT')]}
${[#numbers.setFormatInteger(numSet,3,'POINT')}]

/*
 * =====

```

```

* Formatting decimal numbers
* =====
*/
/*
 * Set minimum integer digits and (exact) decimal digits.
 * Also works with arrays, lists or sets
 */
${#numbers.formatDecimal(num,3,2)}
${#numbers.arrayFormatDecimal(numArray,3,2)}
${#numbers.listFormatDecimal(numList,3,2)}
${#numbers.setFormatDecimal(numSet,3,2)}

/*
 * Set minimum integer digits and (exact) decimal digits, and also decimal separator.
 * Also works with arrays, lists or sets
 */
${#numbers.formatDecimal(num,3,2,'COMMA')}
${#numbers.arrayFormatDecimal(numArray,3,2,'COMMA')}
${#numbers.listFormatDecimal(numList,3,2,'COMMA')}
${#numbers.setFormatDecimal(numSet,3,2,'COMMA')}

/*
 * Set minimum integer digits and (exact) decimal digits, and also thousands and
 * decimal separator.
 * Also works with arrays, lists or sets
 */
${#numbers.formatDecimal(num,3,'POINT',2,'COMMA')}
${#numbers.arrayFormatDecimal(numArray,3,'POINT',2,'COMMA')}
${#numbers.listFormatDecimal(numList,3,'POINT',2,'COMMA')}
${#numbers.setFormatDecimal(numSet,3,'POINT',2,'COMMA')}

/*
 * =====
 * Formatting currencies
 * =====
*/
${#numbers.formatCurrency(num)}
${#numbers.arrayFormatCurrency(numArray)}
${#numbers.listFormatCurrency(numList)}
${#numbers.setFormatCurrency(numSet)}

/*
 * =====
 * Formatting percentages
 * =====
*/
${#numbers.formatPercent(num)}
${#numbers.arrayFormatPercent(numArray)}
${#numbers.listFormatPercent(numList)}
${#numbers.setFormatPercent(numSet)}

/*
 * Set minimum integer digits and (exact) decimal digits.
 */
${#numbers.formatPercent(num, 3, 2)}
${#numbers.arrayFormatPercent(numArray, 3, 2)}
${#numbers.listFormatPercent(numList, 3, 2)}
${#numbers.setFormatPercent(numSet, 3, 2)}

/*
 * =====

```

```

-----  

* Utility methods  

* =====  

*/  
  

/*  

 * Create a sequence (array) of integer numbers going  

 * from x to y  

 */  

${#numbers.sequence(from,to)}  

${#numbers.sequence(from,to,step)}

```

Strings

- **#strings** : utility methods for `String` objects:

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Strings
 * =====
 */  
  

/*
 * Null-safe toString()
 */
${#strings.toString(obj)}                                // also array*, list* and set*  
  

/*
 * Check whether a String is empty (or null). Performs a trim() operation before check
 * Also works with arrays, lists or sets
 */
${#strings.isEmpty(name)}
${#strings.arrayIsEmpty(nameArr)}
${#strings.listIsEmpty(nameList)}
${#strings.setIsEmpty(nameSet)}  
  

/*
 * Perform an 'isEmpty()' check on a string and return it if false, defaulting to
 * another specified string if true.
 * Also works with arrays, lists or sets
 */
${#strings.defaultString(text,default)}
${#strings.arrayDefaultString(textArr,default)}
${#strings.listDefaultString(textList,default)}
${#strings.setDefaultString(textSet,default)}  
  

/*
 * Check whether a fragment is contained in a String
 * Also works with arrays, lists or sets
 */
${#strings.contains(name,'ez')}                         // also array*, list* and set*
${#strings.containsIgnoreCase(name,'ez')}                // also array*, list* and set*  
  

/*
 * Check whether a String starts or ends with a fragment
 * Also works with arrays, lists or sets
 */
${#strings.startsWith(name,'Don')}                      // also array*, list* and set*
${#strings.endsWith(name,endingFragment)}                // also array*, list* and set*  
  

/*
 * Substring-related operations
 * Also works with arrays, lists or sets
*/

```

```

${#strings.indexOf(name,frag)}                                // also array*, list* and set*
${#strings.substring(name,3,5)}                            // also array*, list* and set*
${#strings.substringAfter(name,prefix)}                     // also array*, list* and set*
${#strings.substringBefore(name,suffix)}                    // also array*, list* and set*
${#strings.replace(name,'las','ler')}                      // also array*, list* and set*

/*
 * Append and prepend
 * Also works with arrays, lists or sets
 */
${#strings.prepend(str,prefix)}                           // also array*, list* and set*
${#strings.append(str,suffix)}                           // also array*, list* and set*

/*
 * Change case
 * Also works with arrays, lists or sets
 */
${#strings.toUpperCase(name)}                           // also array*, list* and set*
${#strings.toLowerCase(name)}                           // also array*, list* and set*

/*
 * Split and join
 */
${#strings.arrayJoin(namesArray,',')}                   // returns String[]
${#strings.listJoin(namesList,',')}                     // returns List<String>
${#strings.setJoin(namesSet,',')}                      // returns Set<String>

${#strings.arraySplit(namesStr,',')}                  // returns String[]
${#strings.listSplit(namesStr,',')}                   // returns List<String>
${#strings.setSplit(namesStr,',')}                     // returns Set<String>

/*
 * Trim
 * Also works with arrays, lists or sets
 */
${#strings.trim(str)}                                 // also array*, list* and set*

/*
 * Compute length
 * Also works with arrays, lists or sets
 */
${#strings.length(str)}                             // also array*, list* and set*

/*
 * Abbreviate text making it have a maximum size of n. If text is bigger, it
 * will be clipped and finished in "..."
 * Also works with arrays, lists or sets
 */
${#strings.abbreviate(str,10)}                        // also array*, list* and set*

/*
 * Convert the first character to upper-case (and vice-versa)
 */
${#strings.capitalize(str)}                          // also array*, list* and set*
${#strings.unCapitalize(str)}                        // also array*, list* and set*

/*
 * Convert the first character of every word to upper-case
 */
${#strings.capitalizeWords(str)}                     // also array*, list* and set*
${#strings.capitalizeWords(str,delimiters)}          // also array*, list* and set*

/*
 * Escape the string
 */
${#strings.escapeXml(str)}                          // also array*, list* and set*
${#strings.escapeJava(str)}                         // also array*, list* and set*
${#strings.escapeJavaScript(str)}                   // also array*, list* and set*

```

```

${#strings.unescapeJava(str)}                                // also array*, list* and set*
${#strings.unescapeJavaScript(str)}                         // also array*, list* and set*

/*
 * Null-safe comparison and concatenation
 */
${#strings.equals(first, second)}
${#strings.equalsIgnoreCase(first, second)}
${#strings.concat(values...)}
${#strings.concatReplaceNulls(nullValue, values...)}

/*
 * Random
 */
${#strings.randomAlphanumeric(count)}

```

Objects

- **#objects** : utility methods for objects in general

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Objects
 * =====
 */

/*
 * Return obj if it is not null, and default otherwise
 * Also works with arrays, lists or sets
 */
${#objects.nullSafe(obj,default)}
${#objects.arrayNullSafe(objArray,default)}
${#objects.listNullSafe(objList,default)}
${#objects.setNullSafe(objSet,default)}

```

Booleans

- **#bools** : utility methods for boolean evaluation

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Bools
 * =====
 */

/*
 * Evaluate a condition in the same way that it would be evaluated in a th:if tag
 * (see conditional evaluation chapter afterwards).
 * Also works with arrays, lists or sets
 */
${#bools.isTrue(obj)}
${#bools.arrayIsTrue(objArray)}
${#bools.listIsTrue(objList)}
${#bools.setIsTrue(objSet)}

/*
 * Evaluate with negation
 * Also works with arrays, lists or sets
 */
${#bools.isFalse(cond)}
${#bools.arrayIsFalse(condArray)}
${#bools.listIsFalse(condList)}
${#bools.setIsFalse(condSet)}

/*
 * Evaluate and apply AND operator
 * Receive an array, a list or a set as parameter
 */
${#bools.arrayAnd(condArray)}
${#bools.listAnd(condList)}
${#bools.setAnd(condSet)}

/*
 * Evaluate and apply OR operator
 * Receive an array, a list or a set as parameter
 */
${#bools.arrayOr(condArray)}
${#bools.listOr(condList)}
${#bools.setOr(condSet)}

```

Arrays

- **#arrays** : utility methods for arrays

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Arrays
 * =====
 */

/*
 * Converts to array, trying to infer array component class.
 * Note that if resulting array is empty, or if the elements
 * of the target object are not all of the same class,
 * this method will return Object[].
 */
${#arrays.toArray(object)}

/*
 * Convert to arrays of the specified component class.
 */
${#arrays.toStringArray(object)}
${#arrays.toIntegerArray(object)}
${#arrays.toLongArray(object)}
${#arrays.toDoubleArray(object)}
${#arrays.toFloatArray(object)}
${#arrays.toBooleanArray(object)}

/*
 * Compute length
 */
${#arrays.length(array)}

/*
 * Check whether array is empty
 */
${#arrays.isEmpty(array)}

/*
 * Check if element or elements are contained in array
 */
${#arrays.contains(array, element)}
${#arrays.containsAll(array, elements)}

```

Lists

- **#lists : utility methods for lists**

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Lists
 * =====
 */

/*
 * Converts to list
 */
${#lists.toList(object)}

/*
 * Compute size
 */
${#lists.size(list)}

/*
 * Check whether list is empty
 */
${#lists.isEmpty(list)}

/*
 * Check if element or elements are contained in list
 */
${#lists.contains(list, element)}
${#lists.containsAll(list, elements)}

/*
 * Sort a copy of the given list. The members of the list must implement
 * comparable or you must define a comparator.
 */
${#lists.sort(list)}
${#lists.sort(list, comparator)}

```

Sets

- **#sets** : utility methods for sets

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Sets
 * =====
 */

/*
 * Converts to set
 */
${#sets.toSet(object)}

/*
 * Compute size
 */
${#sets.size(set)}

/*
 * Check whether set is empty
 */
${#sets.isEmpty(set)}

/*
 * Check if element or elements are contained in set
 */
${#sets.contains(set, element)}
${#sets.containsAll(set, elements)}

```

Maps

- **#maps** : utility methods for maps

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Maps
 * =====
 */

/*
 * Compute size
 */
${#maps.size(map)}

/*
 * Check whether map is empty
 */
${#maps.isEmpty(map)}

/*
 * Check if key/s or value/s are contained in maps
 */
${#maps.containsKey(map, key)}
${#maps.containsAllKeys(map, keys)}
${#maps.containsValue(map, value)}
${#maps.containsAllValues(map, value)}

```

Aggregates

- **#aggregates** : utility methods for creating aggregates on arrays or collections

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Aggregates
 * =====
 */

/*
 * Compute sum. Returns null if array or collection is empty
 */
${#aggregates.sum(array)}
${#aggregates.sum(collection)}

/*
 * Compute average. Returns null if array or collection is empty
 */
${#aggregates.avg(array)}
${#aggregates.avg(collection)}

```

IDs

- **#ids** : utility methods for dealing with `id` attributes that might be repeated (for example, as a result of an iteration).

```

/*
 * =====
 * See javadoc API for class org.thymeleaf.expression.Ids
 * =====
 */

/*
 * Normally used in th:id attributes, for appending a counter to the id attribute value
 * so that it remains unique even when involved in an iteration process.
 */
${#ids.seq('someId')}

/*
 * Normally used in th:for attributes in <label> tags, so that these labels can refer to Ids
 * generated by means of the #ids.seq(...) function.
 *
 * Depending on whether the <label> goes before or after the element with the #ids.seq(...)
 * function, the "next" (label goes before "seq") or the "prev" function (label goes after
 * "seq") function should be called.
 */
${#ids.next('someId')}
${#ids.prev('someId')}

```

20 Appendix C: Markup Selector Syntax

Thymeleaf's Markup Selectors are directly borrowed from Thymeleaf's parsing library: [AttoParser](#).

The syntax for this selectors has large similarities with that of selectors in XPath, CSS and jQuery, which makes them easy to use for most users. You can have a look at the complete syntax reference at the [AttoParser documentation](#).

For example, the following selector will select every `<div>` with the class `content`, in every position inside the markup (note this is not as concise as it could be, read on to know why):

```
<div th:insert="mytemplate :: //div[@class='content']">...</div>
```

The basic syntax includes:

- `/x` means direct children of the current node with name `x`.
- `//x` means children of the current node with name `x`, at any depth.
- `x[@z="v"]` means elements with name `x` and an attribute called `z` with value "`v`".
- `x[@z1="v1" and @z2="v2"]` means elements with name `x` and attributes `z1` and `z2` with values "`v1`" and "`v2`", respectively.
- `x[i]` means element with name `x` positioned in number `i` among its siblings.
- `x[@z="v"][i]` means elements with name `x`, attribute `z` with value "`v`" and positioned in number `i` among its siblings that also match this condition.

But more concise syntax can also be used:

- `x` is exactly equivalent to `//x` (search an element with name or reference `x` at any depth level, a *reference* being a `th:ref` or a `th:fragment` attribute).
- Selectors are also allowed without element name/reference, as long as they include a specification of arguments. So `[@class='oneclass']` is a valid selector that looks for any elements (tags) with a class attribute with value "`oneclass`".

Advanced attribute selection features:

- Besides `=` (equal), other comparison operators are also valid: `!=` (not equal), `^=` (starts with) and `$=` (ends with). For example: `x[@class^='section']` means elements with name `x` and a value for attribute `class` that starts with `section`.
- Attributes can be specified both starting with `@` (XPath-style) and without (jQuery-style). So `x[z='v']` is equivalent to `x[@z='v']`.
- Multiple-attribute modifiers can be joined both with `and` (XPath-style) and also by chaining multiple modifiers (jQuery-style). So `x[@z1='v1' and @z2='v2']` is actually equivalent to `x[@z1='v1'][@z2='v2']` (and also to `x[z1='v1'][z2='v2']`).

Direct *jQuery-like* selectors:

- `x.oneclass` is equivalent to `x[class='oneclass']`.
- `.oneclass` is equivalent to `[class='oneclass']`.
- `x#oneid` is equivalent to `x[id='oneid']`.
- `#oneid` is equivalent to `[id='oneid']`.

- `x%oneref` means `<x>` tags that have a `th:ref="oneref"` or `th:fragment="oneref"` attribute.
- `%oneref` means any tags that have a `th:ref="oneref"` or `th:fragment="oneref"` attribute. Note this is actually equivalent to simply `oneref` because references can be used instead of element names.
- Direct selectors and attribute selectors can be mixed: `a.external[@href^='https']`.

So the above Markup Selector expression:

```
<div th:insert="mytemplate :: //div[@class='content']">...</div>
```

Could be written as:

```
<div th:insert="mytemplate :: div.content">...</div>
```

Examining a different example, this:

```
<div th:replace="mytemplate :: myfrag">...</div>
```

Will look for a `th:fragment="myfrag"` fragment signature (or `th:ref` references). But would also look for tags with name `myfrag` if they existed (which they don't, in HTML). Note the difference with:

```
<div th:replace="mytemplate :: .myfrag">...</div>
```

...which will actually look for any elements with `class="myfrag"`, without caring about `th:fragment` signatures (or `th:ref` references).

Multivalued class matching

Markup Selectors understand the class attribute to be **multivalued**, and therefore allow the application of selectors on this attribute even if the element has several class values.

For example, `div.two` will match `<div class="one two three" />`

		kafka	RocketMQ	RabbitMQ
定位	设计定位	系统间的数据管道，实时数据处理；例如：常规的消息系统、网站活性跟踪、监控数据、日志收集、处理等	非日志的可靠消息传输。 例如：订单、交易、充值、核计算、消息推送，链式处理，binglog分发等	可靠消息传输，和RocketMQ类似。
	成熟度	日志领域成熟	成熟	成熟
	所属社区 / 公司	Apache	Alibaba开发，已加入到Apache下	Mozilla Public License
	社区活跃度	高	中	高
基础对比	文档完备性	高	高	高
	开发语言	Scala	Java	Erlang
	支持协议	一套自行设计的基于TCP的二进制协议	自己定义的一套 (社区提供JMS-不成熟)	AMQP
	客户端语言	C/C++、Python、Go、Erlang、.NET、Ruby、Node.js PHP等	Java	Java、C、C++、Python、PHP、Perl 等
	持久化方式	磁盘文件	磁盘文件	内存、文件
	部署方式	单机 / 集群	单机 / 集群	单机 / 集群
	集群管理	zookeeper	name server	
	选主方式	从ISR中自动选举一个leader	不支持自动选举。通过设定brokername，brokerid实现。brokername相同，brokerid=0时为master，其他为slave	最早加入集群的broker
可用性、可靠性比较	可用性	非常高 分布式、主从	非常高 分布式、主从	低 主从，采用镜像模式实现。数据量大时可能产生性能瓶颈
	主从切换	自动切换 允许N-1个失败；master失败后自动从slaves中选择一个主；	不支持自动切换 master失败后不能向master发送信息，consumers会30s（默认）可以感知此事件。此后从slaves中选择一个成为master。如果master无法恢复，将导致部分数据丢失	自动切换 自动从加入集群的slave会成为master，因为新加入的大致30s（默认）可以感知此事件。此后从slaves中选择一个成为master。如果master无法恢复，将导致部分数据丢失
	数据可靠性	很好 支持producer单条发送、同步刷盘、同步复制，但这个场景下性能明显下降。	很好 producer单条发送，broker端支持同步刷盘、异步刷盘、同步双写、异步复制。	好 producer支持同步 / 异步ack，支持队列数据持久化，镜像模式下支持主从同步
	消息写入性能	非常好 每条10个字节测试，百万条/s	很好 每条10个字节测试，单机单broker约7w/s，单机多broker约12w/s	RAM为RocketMQ的1/2，Disk的性能约RAM性能的1/3
	性能的稳定性	队列/分区多样性不稳定，明显下降。 消息堆积时性能稳定	队列较多、消息堆积时性能稳定	消息堆积时，性能不稳定，明显下降
	单机支持的队列数	单机超过64个队列/分区，Load会发生明显的飙升现象，队列越多，load越高，发送消息响应时间变长	单机支持最高5万个队列，Load不会发生明显变化	依赖于内存
	堆积能力	非常好 消息存储在log中，每个分区一个log文件	非常好 所有消息存储在同一个commit log中	一般 生产者、消费者正常时，性能表现稳定；消费者不消费时，性能不稳定
	复制备份	消息先写入leader的log，followers从leader中pull消息。之后再ack leader，然后写入log中。 ISB中被与leader同步的列表，落后太多的follower被剔除掉	同步双写 同步复制：slave启动线程从master中拉数据被刷掉	普通模式下不复制。 镜像模式下：消息先到master，然后写到slave上。如果在slave之前的消息不会被复制到新的slave上。
	消息投递实时性	毫秒级 具体由consumer轮询间隔时间决定	毫秒级 支持pull、push两种模式，延时通常在毫秒级	毫秒级
功能对比	顺序消费	支持顺序消费 但是每一台Broker宕机后，就会产生消息乱序	支持顺序消费 但是在顺序消息场景下，消费失败时消费队列将会暂停	支持顺序消费 但是如果一个消费失败，此消息的顺序会被打乱
	定时消息	不支持	开源版本仅支持定时Level	不支持
	事务消息	不支持	支持	不支持
	Broker端消息过滤	支持	通过tag过滤，类似于子topic	不支持
	消息查询	不支持	根据MessageId查询 支持根据MessageKey查询消息	不支持
	消费失败重试	不支持失败重试 offset存储在consumer中，无法保证。 0.8.2版本后将offset存储在zk中	支持失败重试 offset存储在broker中	支持失败重试
	消息重新消费	支持通过修改offset来重新消费	支持按照时间来重新消息	
	发送端负载均衡	可自由指定	可自由指定	需要单独loadbalancer支持
	消费并行度	消费并行度和分区数一致	顺序消费：消费并行度和分区数一致 乱序消费：消费服务器的消费线程数之和	镜像模式下其实也是从master消费
	消费方式	consumer pull	consumer pull / broker push	broker push
运维	批量发送	支持 默认producer缓存、压缩，然后批量发送	不支持	不支持
	消息清理	指定文件保存时间，过期删除	指定文件保存时间，过期删除	可用内存少于40%（默认），触发gc，gc时找到相关的两个文件，合并right文件到left。
	访问权限控制	无	无	类似数据库一样，需要配置用户名密码
	系统维护	Scala语言开发，维护成本高	Java语言开发，维护成本低	Erlang语言开发，维护成本高
	部署依赖	zookeeper	nameserver	Erlang环境
	管理后台	官网不提供，第三方开源管理工具可供使用；不用重新开发	官方提供，rocketmq-console	官方提供rabbitmqadmin
管理后台功能	Kafka Web Console	Brokers列表、Kafka 集群中 Topic列表，及对应的Partition、LogSize等信息；Topic对应的Consumer Groups、Offset、Lag等信息；Groups列表，及对应的LeaderOffsetMonitor	Cluster、Topic、Connection、NameServ、Message Broker、Offset、Consumer	overview、connections、channels、exchanges、queues、admin
	Kafka Manager	管理各个节点的状态，查看集群状态(topic、partition、offset、lag等信息)；展示topic和consumer之间的关系；图形化展示consumer的offset、lag等信息		
总结	优点	1. 在高吞吐、低延迟、高可用、集群热扩展、集群热迁移等方面表现良好。 2. producer端提供缓存、压缩功能，可节省性能，提高效率。 3. 提供多样消费能力。 4. 提供多种客户端语言。 5. 生态丰富，在大数据处理方面有大量配套的设施。	1. 相比于RabbitMQ，使用人数较少，生态不够完善。 2. api、系统设计都更加适合在业务处理的场景，支持多种消费方式。 3. 支持broker消息过滤。 4. 支持事务。 5. 提供多样化的消费能力：consumer可以水平扩展，消费能力很强。 6. 生态丰富，在大数据处理方面，单机处理消息上百亿，历届大规模的数据处理经验，比较接地气。	1. 在高吞吐量、高可用、集群热扩展、集群热迁移、性能上占有一定优势。 2. 支持多种客户端语言，支持soap协议。 3. 由于erlang语言的特性，性能也比较好，使用RAM模式时，性能很好。 4. 管理界面较丰富，在互联网公司也有较大规模的应用。 5. 官方提供50万台左右，单机处理消息上百亿，历届大规模的数据处理经验，比较接地气。
	缺点	1. 消费者数目受到分区数目的限制。 2. 单机topic过多时，性能会明显降低。 3. 不支持事务。	1. 相比于RabbitMQ，使用人数较少，生态不够完善。 2. 不支持事务，消息实时能力有限。 3. 支持的语言较少。	1. erlang语言难度较大，集群不支持动态扩展。 2. 消息堆积时，性能会明显降低。 3. 官方只支持java。