# 使用步骤:

###### 1 新建springboot工程，在可选择的组件中勾选cache组件，也可以在Pom文件中导入以下依赖:

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

**2 在启动类上加上注解**

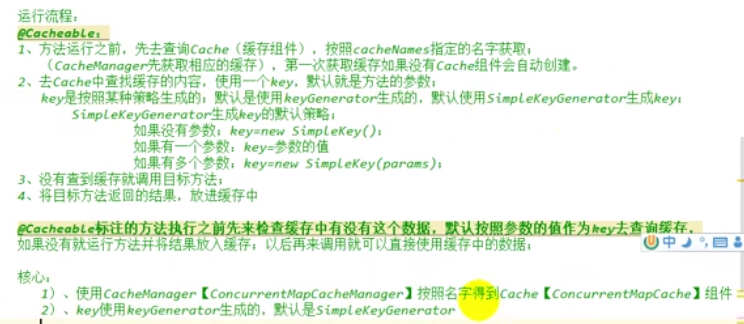
@EnableCaching//开启基于注解的缓存

###### 3 在方法上添加注解

@Cacheable(value = "emp",key = "#id")  
public Employee findEmpById(Integer id) {  
 Employee emp = employeeMapper.getEmpById(id);  
 *log*.warn("{}号员工被查询",id);  
 return emp;  
}

**此时在调用有@Cacheable方法时，返回值会被缓存，第二次调用该方法时会从缓存中取出数据，不会访问数据库**

**4 @Cacheable 具体运行流程**



**5 @Cacheable和@CachePut的区别**

[**执行时机不同：@Cacheable在目标**](mailto:1.执行时机@Cacheable在目标)**方法执行前执行,@CachePut在目标方法执行后执行**

# 2 Linux（ubuntu）安装docker

# ubuntu安装docker

官网文档(如果英文不太好,可以将其翻译成中文看)因为我用的操作系统是ubuntu,所以我就安装ubuntu版本docker  
  
[https://docs.docker.com/install/linux/docker-ce/ubuntu/](https://link.jianshu.com/?t=https%3A%2F%2Fdocs.docker.com%2Finstall%2Flinux%2Fdocker-ce%2Fubuntu%2F)

卸载旧版本  
老版本的Docker被称为docker或docker-engine。如果安装了这些，请将其卸载：

$ sudo apt-get remove docker docker-engine docker.io

## 使用存储库进行安装

* 首次在新的主机上安装Docker CE之前，需要设置Docker存储库。之后，您可以从存储库安装和更新Docker。
* 设置存储库

1.更新apt软件包索引：

$ sudo apt-get update

2.安装软件包以允许apt通过HTTPS使用存储库：

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

3.添加Docker的官方GPG密钥：

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

如果卡住或出现异常:

apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys D8576A8BA88D21E9

-- 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88通过搜索指纹的最后8个字符，确认您现在拥有指纹的密钥 。

$ sudo apt-key fingerprint 0EBFCD88

pub 4096R/0EBFCD88 2017-02-22

Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid Docker Release (CE deb) <docker@docker.com>

sub 4096R/F273FCD8 2017-02-22

4.使用以下命令来设置稳定的存储库。即使您想从边缘或测试存储库安装构建，也总是需要稳定的存储 库。要添加边缘或 测试存储库，请在下面的命令中添加单词或（或两者）后面的单词。edgeteststable

* 注意：下面的lsb\_release -cs子命令返回你的Ubuntu发行版的名字，比如xenial。有时候，像Linux Mint这样的发行版中，可能需要更改$(lsb\_release -cs) 为您的父级Ubuntu发行版。例如，如果你正在使用 Linux Mint Rafaela，你可以使用trusty。
* <b style="color:red">x86\_64 / amd64</b> | armhf | IBM Power（ppc64le）| IBM Z（s390x）

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

# 安装DOCKER CE

1.更新apt软件包索引。

$ sudo apt-get update

2.安装最新版本的Docker CE，或者转到下一步安装特定版本。任何现有的Docker安装都将被替换。

$ sudo apt-get install docker-ce

<b>有多个Docker存储库？</b>

<b>如果您启用了多个Docker存储库，则安装或更新，而不指定版本apt-get install或 apt-get update命令始终安装最高版本，这可能不适合您的稳定性需求。</b>

3.在生产系统上，您应该安装特定版本的Docker CE，而不是始终使用最新版本。此输出被截断。列出可用的版本。

$ apt-cache madison docker-ce

docker-ce | 17.12.0~ce-0~ubuntu | https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages

<b>列表的内容取决于启用了哪个存储库。选择一个特定的版本进行安装。第二列是版本字符串。第三列是存储库名称，它指明了软件包来自哪个存储库，并且通过扩展其稳定性级别。要安装特定版本，请将版本字符串附加到包名称，并用等号（=）将它们分开：</b>

$ sudo apt-get install docker-ce=<VERSION>

（我这里安装的是17.12,也就是最新版本 sudo apt-get install docker-ce=<17.12>）

Docker守护进程自动启动。

4.通过运行hello-world 映像验证是否正确安装了Docker CE 。

$ sudo docker run hello-world

5.运行hello-word的结果

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/engine/userguide/

6.查看当前安装docker 版本

$ docker -v

Docker version 17.12.0-ce, build

# 3 使用docker安装redis

1 百度搜索 docker hub 进入docker的镜像官网，搜索redis,此时可以看到安装redis镜像的命令: docker pull redis ,由于是从国外下载的，速度很慢，所以我们要从国内下载，百度搜索docker 中国，进入<https://www.cnblogs.com/weifeng1463/p/7468391.html> 此时可以看到下载的方式docker pull registry.docker-cn.com/library/ubuntu:16.04

我们改成docker pull registry.docker-cn.com/library/redis 就可以从国内的镜像下载redis

2 docker images 命令可以查看docker中的镜像

root@VM-0-11-ubuntu:/home/ubuntu# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

registry.docker-cn.com/library/redis latest 0f88f9be5839 3 weeks ago 95MB

hello-world

3 启动redis docker run -d -p 6379:6379 --name myredis registry.docker-cn.com/library/redis

此时是创建并启动一个docker容器

使用docker ps 命令可以看到docker中启动的容器

docker ps –a 查看docker中所有的容器

docker start 容器ID 启动容器

4 使用redis desktop manager工具进行连接redis 默认没有密码

附：

docker ps

主要有些参数要说一下

1. 不加参数，表示查看当前正在运行的容器

2. -a，查看所有容器包括停止状态的容器

3. -l，查看最新创建的容器

4. -n=x，查看最后创建的x个容器

列一下docker ps的结果

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

这些分别代表什么

CONTAINER ID：容器ID，唯一标识容器

IMAGE：创建容器时所用的镜像

COMMAND：在容器最后运行的命令

CREATED：容器创建的时间

STATUS：容器的状态（你会看到UPXXX，表示运行状态）

PORTS：对外开放的端口号

NAMES：容器名（也具有唯一性，docker是不允许创建容器名相同的容器的）

启动

我们在用docker ps -a的时候会看见一些容器的状态是CREATED的，这里就需要我们用docker start 容器名或容器ID来启动容器。但是需要注意，用这个命令后，容器进入运行状态，当容器完成任务后，会自己退出，进入停止状态。如果需要再次启动start命令

这里提一个参数，能够让容器在退出后能够自动重启

--restart这个参数会检查容器的退出码，并据此来决定是否要重启容器。

这个例子大家不要轻易尝试，这样会不停的输出hello world，用ctrl+C都没有停掉这个（这里过后好好想想为什么没有停掉？？？？？？？？？）

最后我开了另一个终端输入了

docker stop docker\_restart才终止了容器

终止

在上面也说了一下命令是

docker stop 容器名或容器ID

而这个docker stop命令是给容器进程发送一个SIGTERM信号（可捕捉），默认行为是容器退出。

如果要强行停止一个容器的话，最好用docker kill命令

它是发送一个SIGKILL信号（不可捕捉）

删除

我们不能删除一个正在运行的容器，必须先docker stop或者docker kill后我们才能删除

命令：docker rm 容器名

当然如果我们加-f这个参数我们也是可以删除一个正在运行的容器的

如何一次删除所有容器docker没有给相关命令，但是我们可以这样

-q是只列出容器ID

# 4 springboot中使用redis

##### 引入pom文件,并且添加redis服务器的配置地址

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

# 配置redis数据库的连接地址，端口默认配置的是6379  
spring.redis.url=www.shiyunkai.com

###### 在IDEA中 ctrl+n 搜索redisautoconfiguration 可以看到redis自动装配的源码

@Bean  
@ConditionalOnMissingBean(name = "redisTemplate")  
public RedisTemplate<Object, Object> redisTemplate(  
 RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {  
 RedisTemplate<Object, Object> template = new RedisTemplate<>();  
 template.setConnectionFactory(redisConnectionFactory);  
 return template;  
}  
  
@Bean  
@ConditionalOnMissingBean  
public StringRedisTemplate stringRedisTemplate(  
 RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {  
 StringRedisTemplate template = new StringRedisTemplate();  
 template.setConnectionFactory(redisConnectionFactory);  
 return template;  
}

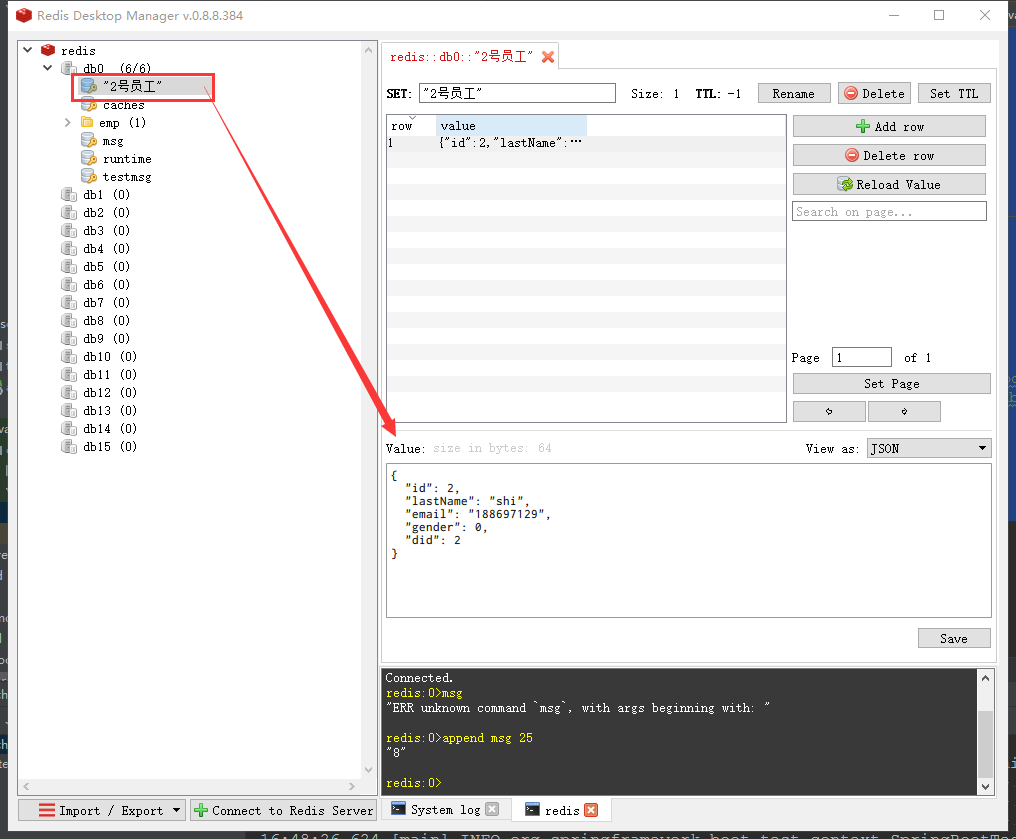
###### 可以看到往容器中添加了两个redis template

其中redisTemplate是操作对象的，stringRedisTemplate是操作字符串的

# redisTemplate的使用:

//redis两个template的注入  
@Autowired  
private RedisTemplate redisTemplate;  
  
@Autowired  
private StringRedisTemplate stringRedisTemplate;  
  
@Autowired  
private RedisTemplate<Object, Employee> myTemplate;  
  
/\* \* @Auther: shiyunkai  
 \* @Description: redis template的使用  
 \* @Param: []  
 \* @Date: 15:47 2019/3/26  
 \* @return: void  
 \*\*/  
@Test  
public void testRedis(){  
 stringRedisTemplate.opsForValue().append("msg","我是dbdbdbdb");  
 stringRedisTemplate.opsForValue().append("testmsg","dbdbdbdb");  
 Employee emp = employeeService.findEmpById(2);  
 //redisTemplate.opsForSet().add("emp1",emp);  
  
 // 测试自己配置的redis template  
 myTemplate.opsForSet().add("2号员工",emp);  
}

###### 运行测试类后可以在 redis desktop manager中查看redis数据库中的数据：



##### 其中myTemple是我们自已配置的redis template,和默认的redis template的区别是自定义了序列化器

@Configuration  
public class MyRedisTemplateConfig {  
  
 @Bean  
 @ConditionalOnMissingBean(name = "myTemplate")  
 public RedisTemplate<Object, Employee> redisTemplate(  
 RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {  
 RedisTemplate<Object, Employee> myTemplate = new RedisTemplate<>();  
 myTemplate.setConnectionFactory(redisConnectionFactory);  
 // 自定义序列化器（可以将java对象转换为json格式的数据存储在redis中）  
 myTemplate.setDefaultSerializer(new Jackson2JsonRedisSerializer<>(Employee.class));  
 return myTemplate;  
 }  
}

# 配置RedisCacheManager和RedisTemplatet

@Configuration  
public class MyRedisTemplateConfig {  
  
 // 配置redis template  
 @Bean  
 @SuppressWarnings("all")  
 public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {  
 RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();  
 template.setConnectionFactory(factory);  
 Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);  
 ObjectMapper om = new ObjectMapper();  
 om.setVisibility(PropertyAccessor.*ALL*, JsonAutoDetect.Visibility.*ANY*);  
 om.enableDefaultTyping(ObjectMapper.DefaultTyping.*NON\_FINAL*);  
 jackson2JsonRedisSerializer.setObjectMapper(om);  
 StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();  
 // key采用String的序列化方式  
 template.setKeySerializer(stringRedisSerializer);  
 // hash的key也采用String的序列化方式  
 template.setHashKeySerializer(stringRedisSerializer);  
 // value序列化方式采用jackson  
 template.setValueSerializer(jackson2JsonRedisSerializer);  
 // hash的value序列化方式采用jackson  
 template.setHashValueSerializer(jackson2JsonRedisSerializer);  
 template.afterPropertiesSet();  
 return template;  
 }  
  
  
 //@Bean  
 //@ConditionalOnMissingBean(name = "myTemplate")  
 //public RedisTemplate<Object, Employee> redisTemplate(  
 // RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {  
 // RedisTemplate<Object, Employee> myTemplate = new RedisTemplate<>();  
 // myTemplate.setConnectionFactory(redisConnectionFactory);  
 // // 自定义序列化器（可以将java对象转换为json格式的数据存储在redis中）  
 // myTemplate.setDefaultSerializer(new Jackson2JsonRedisSerializer<>(Employee.class));  
 // return myTemplate;  
 //}  
  
  
 // 配置redis cache manager  
 @Bean  
 public RedisCacheManager cacheManager(RedisConnectionFactory redisConnectionFactory) {  
 //初始化一个RedisCacheWriter  
 RedisCacheWriter redisCacheWriter = RedisCacheWriter.*nonLockingRedisCacheWriter*(redisConnectionFactory);  
 //设置CacheManager的值序列化方式为json序列化  
 RedisSerializer<Object> jsonSerializer = new GenericJackson2JsonRedisSerializer();  
 RedisSerializationContext.SerializationPair<Object> pair = RedisSerializationContext.SerializationPair.*fromSerializer*(jsonSerializer);  
 RedisCacheConfiguration defaultCacheConfig = RedisCacheConfiguration.*defaultCacheConfig*().serializeValuesWith(pair);  
  
 //设置默认超过时期是1天  
 defaultCacheConfig.entryTtl(Duration.*ofDays*(1));  
 //初始化RedisCacheManager  
 return new RedisCacheManager(redisCacheWriter, defaultCacheConfig);  
 }  
  
  
}