单元测试

通用规则

面向接口进行单元测试。

只对api层测试。

覆盖率需要达到要求。

对于业务代码处理的逻辑，需要准备测试用例。测试用例可以与测试人员一起准备，或者由测试人员准备好，开发人员使用测试用例进行单元测试。

单元测试中需要对依赖的外部服务接口Mock，这样能保证单元测试的稳定性。并且在单元测试环境中不会连接注册服务器，因此无法调用feign的服务接口。

单元测试采用docker镜像的方式。本地需要安装docker服务（win7、win10等系统的安装参见附录）。

代码中需要的redis，kafka，es，数据库都采用docker镜像的方式，在单元测试启动的时候启动，单元测试停止时关闭。

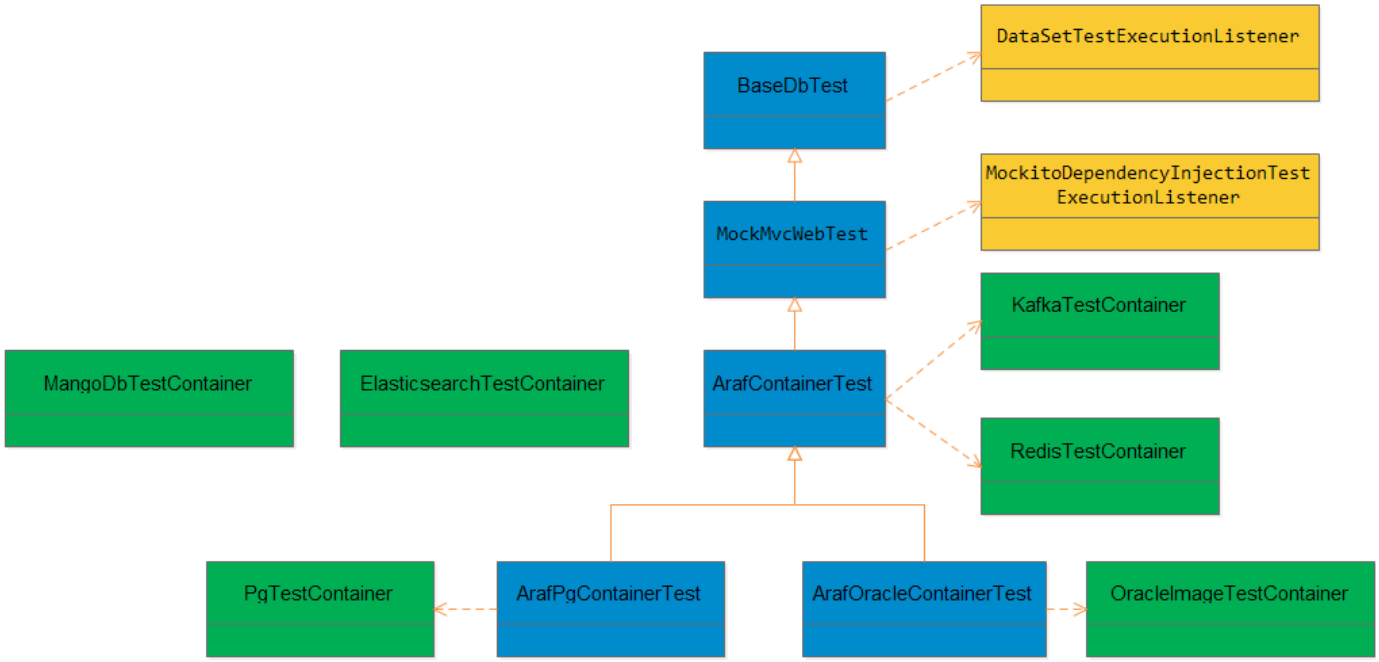
针对数据库数据，每个测试方法前会准备测试数据，方法执行完后，删除数据。保证测试的独立性。

单元测试基类提供了postgre与oracle的支持，对不同的开发项目，继承不同的基类，ArafOracleContainerTest与ArafPgContainerTest

对于操作HDFS文件，框架提供了本地模拟的hdfs接口，不影响测试代码的执行。

某些场景需要在单元测试环境中连外部数据库等，继承MockMvcWebTest即可。

单元测试基类



通用的设置：

@Commit 单元测试中默认数据是提交的。

MockWebMvcTest：默认在单元测试启动的时候，会准备一个虚拟的web环境。测试代码中，可以访问web中的url（api）

Container结尾的类都是工具类，提供docker初始化环境。Test结尾的类，是单元测试的基类。

准备数据，准备Mock对象。

单元测试代码样例

准备schema.sql数据

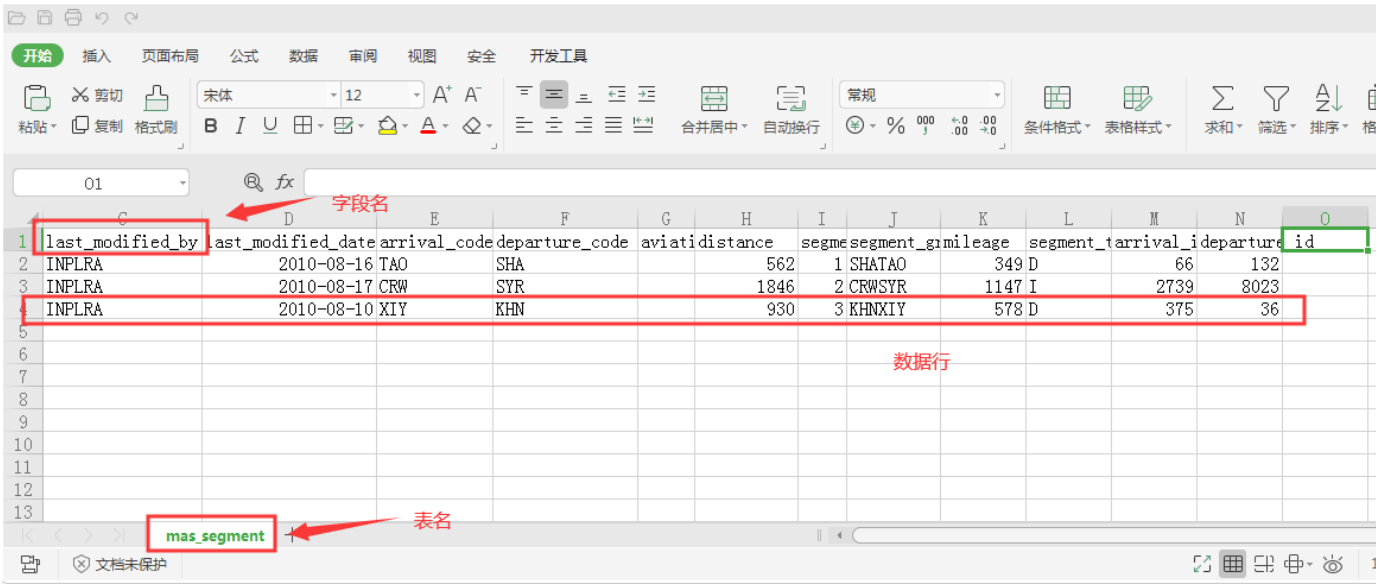
在test/resources目录下的schema.sql文件，单元测试启动后，会根据这个文件中的建表语句创建数据库表。

```
CREATE TABLE mas_currency_rate(
  curr_rate_id character varying(50) NOT NULL,
    created_by character varying(50),
  created_date timestamp(3),
  last_modified_by character varying(50),
  last_modified_date timestamp(3),
  average_rate_usd numeric(15,7),
  currency_code character varying(3),
  quote_price numeric(15,7),
  rate_5_eur numeric(15,7),
  rate_5_gbp numeric(15,7),
  rate_5_self numeric(15,7),
  rate_5_usd numeric(15,7),
  roe numeric(15,7),
  month character varying(6),
  CONSTRAINT ref_currency_rate_pkey PRIMARY KEY (curr_rate_id)
);
```

建表语句只需要基本的create即可，包含主键

准备数据文件

在单元测试test/resouces/dataset目录下准备xls格式的数据文件。其中格式如下：



创建测试类文件

测试类需要加上@SpringBootTest(classes = { AppServer.class })

使用excel表格准备测试数据，并在测试方法上加@ITestDataSet(locations = { "/dataSet/mas_country.xls" }) 指定数据文件的位置。

```
@SpringBootTest(classes = { AppServer.class })
public class MasCountryApiTest extends ArafPgContainerTest {

    @Test
    @ITestDataSet(locations = { "/dataSet/mas_country.xls" })
    public void testFindOne() throws Exception {
        MvcResult result = mockMvc
                .perform(MockMvcRequestBuilders.get("/masCountry/id/{id}",
"158")
                        .contentType(MediaType.APPLICATION_JSON))
                .andDo(MockMvcResultHandlers.print()).andReturn();
        assertThat("", result.getResponse().getStatus(), equalTo
(HttpServletResponse.SC_OK));
    }

    @Test
    @ITestDataSet(locations = { "/dataSet/mas_country.xls" })
    public void testFindPage() throws Exception {
        MvcResult result = mockMvc
                .perform(MockMvcRequestBuilders.get("/masCountrys")
                        .contentType(MediaType.APPLICATION_JSON))
                .andDo(MockMvcResultHandlers.print()).andReturn();
        assertThat("", result.getResponse().getStatus(), equalTo
(HttpServletResponse.SC_OK));
    }

    @Test
    @ITestDataSet(locations = { "/dataSet/mas_country.xls" })
    public void testFindPage2() throws Exception {
        MasCountryVO vo = new MasCountryVO();
        vo.setCountryCode("US");
        MvcResult result = mockMvc
                .perform(MockMvcRequestBuilders.post("/masCountrys")
                        .contentType(MediaType.APPLICATION_JSON).content
(JacksonHelper.entityToJson(new BaseQueryVO<MasCountryVO>(vo))))
                .andDo(MockMvcResultHandlers.print()).andReturn();
        assertThat("", result.getResponse().getStatus(), equalTo
(HttpServletResponse.SC_OK));
    }
}
```

单元测试结果必须写断言。一般应该写业务含义的断言。例子中只是代表执行成功，不代表业务正确。

样例如下：

```
assertThat("origin_effect_date", s.getOriginEffectDate(), equalTo
(DateUtils.parseDate("2019-07-01", "yyyy-MM-dd")));
```

hdfs测试需要配置本地环境

1.添加环境变量

HADOOP_HOME=D:\Program Files (x86)\hadoop

2.添加path

%HADOOP_HOME%\bin

hadoop附件下载：hadoop.rar

3.yml配置启动本地模式：

4.配置完成后，ide重启

hdfs 本地模式

```
#
  hdfs:
    mode: local
    defaultFS: hdfs://127.0.0.1:0
    username: default
```

Mock的写法

在单元测试环境中不会连接注册服务器，因此无法调用feign的服务接口。需要调用外部服务时，可以使用mock方法。

官方MockBean用法

参见代码样例：

Mocktio

```
@SpringBootTest(classes = AppServer.class)
public class GridViewProfileControllerTest extends MockMvcWebTest {

    @Autowired
    @InjectMocks
    private GridViewProfileService gridViewProfileService;

    @MockBean
    private GridViewProfileDao gridViewProfileDao;


        @SpyBean
    private GridViewProfileDao gridViewProfileDao;


        @Test
    @ITestDataSet(locations = { "/dataSet/unittest_all.xls" }, fileType =
"xls", dsNames = {
        "dataSource" })
    public void testSaveOrUpdate() throws Exception {
        GridViewProfile p = new GridViewProfile();
        p.setId("mocked");
        Mockito.when(gridViewProfileDao.findByProfile(Mockito.anyString(),
Mockito.anyString(),
                Mockito.anyString(), Mockito.anyString())).thenReturn(p);
```

在单元测试代码中注入需要进行mock的类，以及使用mock类的service。在测试方法中使用Mockito的静态方法对mock类的方法进行替换。

如果只需要mock其中的一些方法，不是mock整个类，则使用@SpyBean

升级修改

1、需要mock的对象，加上注释@MockBean，不使用@Mock；

2、使用@MockBean的对象，不能使用@Autowired

使用Mock注解

Mocktio

```
@SpringBootTest(classes = AppServer.class)
public class GridViewProfileControllerTest extends MockMvcWebTest {

    @Autowired
    private GridViewProfileService gridViewProfileService; Mockautowired

        @Autowired
    @Mock
    private GridViewProfileDao gridViewProfileDao;  Mock

        @Test
    @ITestDataSet(locations = { "/dataSet/unittest_all.xls" }, fileType =
"xls", dsNames = {
        "dataSource" })
    public void testSaveOrUpdate() throws Exception {
        GridViewProfile p = new GridViewProfile();
        p.setId("mocked");
        Mockito.when(gridViewProfileDao.findByProfile(Mockito.anyString(),
Mockito.anyString(),
                Mockito.anyString(), Mockito.anyString())).thenReturn(p);
```

MockBean的注意事项

> MockBean的使用中，存在一个重大的问题。每一个使用了MockBean的类，spring都会创建一个独立的applicationContext，导致spring会初始化多次，同样导致了数据源会初始化多次，最终导致内存溢出，或者数据库连接不足，单元测试无法结束。底层的原因是，spring为了实现MockBean注释的类，在多个测试类之间独立，从而初始化了多次context。
>
> 为了解决这个问题，有两种解决方法：
>
> 1、架构中保留了原有@Mock的使用方式，用自己的方法进行Mock类的注入，不影响已有的context，不会初始化多次Context。但是对于使用了RefreshScope的类支持不好；
>
> 2、将所有的MockBean，SpyBean都放在一个统一的基类中，所有单元测试的类统一继承这个基类。参考下面的例子。

建立统一的单元测试基类。

```
/**
 * mock
 *
 * @version OPRA v1.0
 * @author Yu Tao, 2020122
 */
public class BaseTest extends ArafPgContainerTest {

    @MockBean
    protected MasAirlineApi airlineApi;

    @SpyBean
    protected HdfsService hdfsService;

    @MockBean
    protected OprJobInstanceApi oprJobInstanceApi;

    @MockBean
    protected InputFileHepler inputFileHepler;

    @MockBean
    protected MasAirlineCache masAirlineCache;

    @MockBean
    protected MasAirportCache masAirportCache;

    @MockBean
    protected MasCityApi masCityApi;

    @MockBean
    protected AttachmentManageApi attatchApi;

    @MockBean
    protected SysWorkflowNowApi sysWorkflowNowApi;

}
```

单元测试类：

```
@SpringBootTest(classes = { AppServer.class })
@Slf4j
public class DccJobTest extends BaseTest {

    @Autowired
    private MasCctGlobalService masCctGlobalService;

    @Autowired
    @InjectMocks
    private DccJob dccJob;

    @Autowired @InjectMocks
```

```
    private InputFileService inputFileService;

    @Test
    @ITestDataSet(locations = { "/dataset/general/mas_cct_global.xls" })
    public void testJob() throws Exception {
        Mockito.when(oprJobInstanceApi.findOne(Mockito.anyString())).
thenReturn(null);
        Mockito.doNothing().when(hdfsService).moveFile(Mockito.any(Path.
class), Mockito.any(Path.class));
        String fileName = "dataset/inputfile/dcc/2020DCH.xls";
        String filePath = Thread.currentThread().getContextClassLoader().
getResource(fileName).getFile();
        //target
        filePath = ArafStringUtils.substring(filePath, 0, ArafStringUtils.
lastIndexOf(filePath, "/"))+"/2020DCH.xls";
        Map<String, String> map = new HashMap<>();
        map.put(AbstractFileImportJob.FILE_PATH, filePath);
        map.put("inputType", InputType.M.toString());
        dccJob.run(JobContext.buildContext(InputFileJobName.DCC_INPUT_JOB,
map));

        assertMasRoes();

    }


.........



@Slf4j
@SpringBootTest(classes = { AppServer.class })
public class MasAirlineApiTest extends BaseTest {

    @Autowired
    private MasAirlineService masAirlineService;

    @Test
    @ITestDataSet(locations = { "/dataset/general/mas_airline.xls",
        "/dataset/general/mas_currency.xls", "/dataset/general/mas_city.
xls" })
    public void testFindOne() throws Exception {
        MvcResult result = mockMvc
                .perform(MockMvcRequestBuilders.get("/masAirline/id/{id}",
"1")
                        .contentType(MediaType.APPLICATION_JSON))
                .andDo(MockMvcResultHandlers.print()).andReturn();
        assertThat("", result.getResponse().getStatus(), equalTo
(HttpServletResponse.SC_OK));
    }
```

这种方法有一个问题，就是没有使用Mock的类，也要继承公共的BaseTest基类，可能导致设计上不好看。但是不影响使用。

多数据源的单元测试配置

某些情况下，我们工程中有多数据源的设计。单元测试中也是可以实现的。需要自定义一个单元测试基类，引入多个数据源的镜像设置即可。

单元测试环境的配置文件

```
datasource:
  #
  prpdata:
     name: prpdata
     type: com.alibaba.druid.pool.DruidDataSource
     driver-class-name: oracle.jdbc.OracleDriver
     initialSize: 5
     minIdle: 5
     maxActive: 20
     maxWait: 60000
     timeBetweenEvictionRunsMillis: 60000
     minEvictableIdleTimeMillis: 300000
     testWhileIdle: true
     testOnBorrow: false
     testOnReturn: false
     poolPreparedStatements: true
     maxPoolPreparedStatementPerConnectionSize: 20
     filters: stat,wall
     connectionProperties: druid.stat.mergeSql=true;druid.stat.
slowSqlMillis=5000
  #
  ipraprot:
     name: ipraprot
     type: com.alibaba.druid.pool.DruidDataSource
     driver-class-name: oracle.jdbc.OracleDriver
     initialSize: 5
     minIdle: 5
     maxActive: 20
     maxWait: 60000
     timeBetweenEvictionRunsMillis: 60000
     minEvictableIdleTimeMillis: 300000
     testWhileIdle: true
     testOnBorrow: false
     testOnReturn: false
     poolPreparedStatements: true
     maxPoolPreparedStatementPerConnectionSize: 20
     filters: stat,wall
     connectionProperties: druid.stat.mergeSql=true;druid.stat.
slowSqlMillis=5000
```

这里配置多个数据源，不需要配置url，user，pwd等。

自定义单元测试基类

```
public class PrpTest extends ArafContainerTest {

    @ClassRule
    public static OracleImageContainer oracle1 = new OracleImageContainer(
            "wnameless/oracle-xe-11g-r2", "integration-tests-db", 1521,
"schema1.sql")
                    .setJdbcUrl("datasource.prpdata.url")
                    .setJdbcUserName("datasource.prpdata.username")
                    .setJdbcPassword("datasource.prpdata.password");

    @ClassRule
    public static OracleImageContainer oracle2 = new OracleImageContainer(
            "wnameless/oracle-xe-11g-r2", "integration-tests-db", 1531,
"schema2.sql")
                    .setJdbcUrl("datasource.ipraprot.url")
                    .setJdbcUserName("datasource.ipraprot.username")
                    .setJdbcPassword("datasource.ipraprot.password");

}
```

代码中引入两个oracle镜像，启动两个数据库

测试方法

```
@Test
    @ITestDataSet(locations = { "/dataset/prpdata/PRP_RESULT.xls" },
dsNames = {"prpdataDataSource" })
    public void testfindResultVOs() throws Exception {
        PrpResultVO vo = new PrpResultVO();
        vo.setHostAirline("1");
        vo.setPrefix("1");
        vo.setTicketno("1");
        vo.setCouponno(1);
        MvcResult result = mockMvc
                .perform(MockMvcRequestBuilders.get("/prpResult
/findresult")
                        .contentType(MediaType.APPLICATION_JSON))
                .andDo(MockMvcResultHandlers.print()).andReturn();
        assertThat("", result.getResponse().getStatus(), equalTo
(HttpServletResponse.SC_OK));

    }
```

这里注意，准备数据时，需要指定数据源的名称。

与sonar集成

DevOps#SONAR发布

附录：

1.对于Windows 10以下的用户，推荐使用 Docker Toolbox

直接下载

客户端安装

https://github.com/docker/toolbox/releases/download/v19.03.1/DockerToolbox-19.03.1.exe

已安装

1.针对已经安装客户端，由于空间不足导致需要重新新建虚机，可以直接在Oracle VM VirtualBox 直接删除虚机，然后按照下面的步骤重新安装

新安装

```
    boot2docker.iso
```

1.安装客户端DockerToolbox-19.03.1.exe（已安装则直接跳过）

2.指定docker machine的存储位置

```
WindowsMACHINE_STORAGE_PATH F:\vm\docker

D:\Program Files\Docker Toolbox\boot2docker.isocacheF:\vm\docker\cache
```

3.修改D:\Program Files\Docker Toolbox\start.sh脚本

```
"${DOCKER_MACHINE}" create

--engine-registry-mirror
--virtualbox-disk-size M


  "${DOCKER_MACHINE}" create '--engine-registry-mirror=https://8080ka5p.
mirror.aliyuncs.com' -d virtualbox --virtualbox-disk-size "100000"
$PROXY_ENV "${VM}"
```

4.运行Docker QuickStart安装即可

5.建议虚机分配至少2cpu、2G内存，在Oracle VM VirtualBox关闭虚机后调整

验证命令： docker ps

```
docker@default:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
docker@default:~$
```

docker-machine ssh default

当类似

docker search mysql

该命令无法执行使，需要调整一下dns

sudo vi /etc/resolv.conf

nameserver 8.8.8.8

常用镜像(本地导入)-推荐

1.10.1.21.24机器上有导出好的镜像
    用户名密码appadm/App1adm2
2.镜像地址

/data/limingyi_oracle-xe-18c-prebuilt:latest.tar

/data/docker-images-bak/images/export 下所有镜像

```
confluentinc_cp-kafka:5.3.1.tar  quay.io_testcontainers_ryuk:0.2.3.tar
hello-world:latest.tar           redis:5.0.5.tar
postgres:11.1.tar                wnameless_oracle-xe-11g-r2:latest.tar
[appadm@PAXC-P-HDFS-14 export]$
```

3.镜像比较大，差不多总共20g，个人的docker记得留出足够的空间
    删除镜像命令
    docker rmi 镜像名
    导入镜像命令
    docker load -i 镜像名limingyi_oracle-xe-18c-prebuilt:latest.tar
   例如：docker load -i limingyi_oracle-xe-18c-prebuilt:latest.tar

   注意：当导入oracle18c后，执行如下命令：

   docker tag ebc535b76e06 limingyi/oracle-xe-18c-prebuilt:latest

常用镜像（自行下载）

常用docker images 拉取

docker pull confluentinc/cp-kafka:5.3.1

docker pull quay.io/testcontainers/ryuk:0.2.3

docker pull alpine:3.5

docker pull postgres:11.1

docker pull redis:5.0.5

docker pull limingyi/oracle-xe-18c-prebuilt

镜像加速

在国内使用Docker Hub的话就特别慢，为此，我们可以给Docker配置国内的加速地址。如阿里云镜像地址(**建议使用**)或其他加速地址

```
docker-machine ssh default
sudo sed -i "s|EXTRA_ARGS='|EXTRA_ARGS='--registry-mirror=https://registry.
docker-cn.com |g" /var/lib/boot2docker/profile
exit
docker-machine restart default
```

oracle的镜像包比较大，第一次运行单元测试需要较长时间等待下载：

```
docker@default:~$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
wnameless/oracle-xe-11g-r2    latest              0d19fd2e072e        4 weeks ago         2.1GB
redis                         5.0.5               63130206b0fa        6 weeks ago         98.2MB
confluentinc/cp-kafka         5.3.1               be6286eb3417        7 weeks ago         590MB
postgres                      9.6.12              ed51199dbcc6        5 months ago        230MB
postgres                      11.1                5a02f920193b        8 months ago        312MB
quay.io/testcontainers/ryuk   0.2.3               64849fd2d464        8 months ago        10.7MB
alpine                        3.5                 f80194ae2e0c        9 months ago        4MB
```

由于上面进行过Docker文件地址的修改，可以从其他同事直接拷备作为自己的Docker镜像

2.对于Windows 10 Home的用户

安装要求

- Windows 10 Home machines must meet the following requirements to install Docker Desktop:
    - Install Windows 10, version 1903 or higher.
    - Enable the WSL 2 feature on Windows. For detailed instructions, refer to the Microsoft documentation.
    - The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 Home:
        - 64 bit processor with Second Level Address Translation (SLAT)
        - 4GB system RAM
        - BIOS-level hardware virtualization support must be enabled in the BIOS settings. For more information, see Virtualization.
    - Download and install the Linux kernel update package.

客户端安装

https://docs.docker.com/docker-for-windows/install-windows-home/

镜像加速

在系统右下角托盘图标内右键菜单选择 Settings，打开配置窗口后左侧导航菜单选择 Docker Daemon。编辑窗口内的JSON串，填写下方加速器地址：

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
```

3.对于Windows 10 专业版以上的用户

安装要求

- Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later).
- Hyper-V and Containers Windows features must be enabled.

客户端安装

https://docs.docker.com/docker-for-windows/install/

镜像加速

在系统右下角托盘图标内右键菜单选择 Settings，打开配置窗口后左侧导航菜单选择 Docker Daemon。编辑窗口内的JSON串，填写下方加速器地址：

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
```

4.注意

Docker for Windows 和 Docker Toolbox互不兼容，如果同时安装两者的话，需要使用hyperv的参数启动。

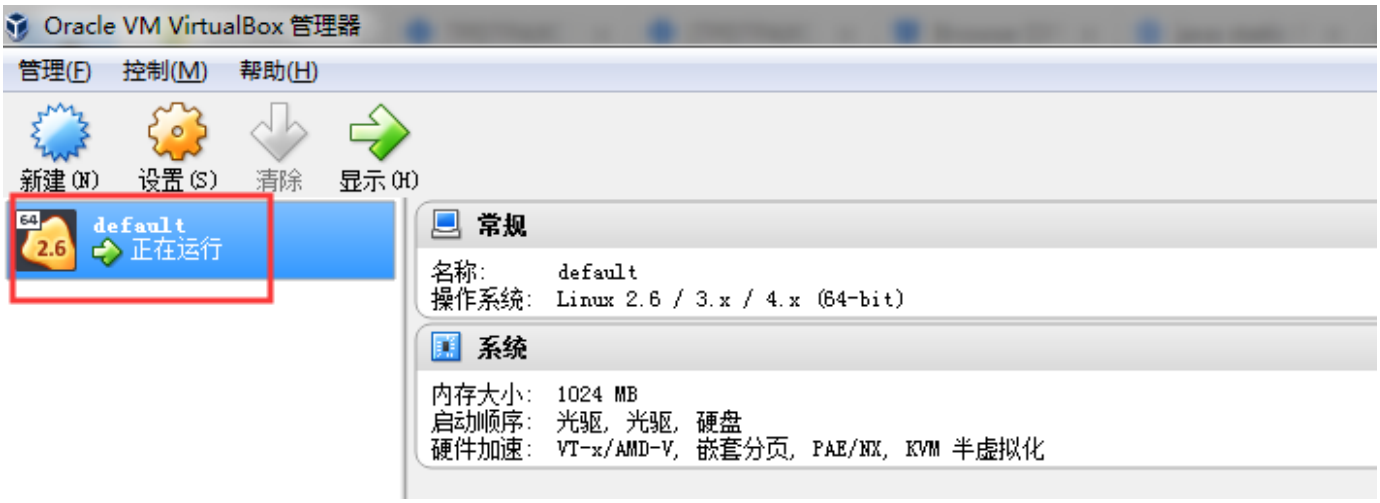
```
docker-machine create --engine-registry-mirror=https://8080ka5p.mirror.
aliyuncs.com -d hyperv default
```

Docker for Windows 有两种运行模式，一种运行Windows相关容器，一种运行传统的Linux容器。同一时间只能选择一种模式运行。

运行中出现下面的错误：

```
Caused by: com.github.dockerjava.api.exception.
InternalServerErrorException: {"message":"driver failed programming
external connectivity on endpoint objective_wright
(709b2f91ae761c8cfb7c2c1caa99360265da62eb7c20f82de40d4038545c6dfb): Bind
for 0.0.0.0:6379 failed: port is already allocated"}
```

打开oracle虚拟机，重启default虚拟机



5. 参考文档

Docker 命令参考文档

Dockerfile 镜像构建参考文档