文章
Peng Qiao · 一月 14, 2021 阅读大约需2 分钟

Open Exchange

Dockerfile 和它的朋友们或者如何在 InterSystems IRIS 上运行和合作 ObjectScript 项目

你好，开发者！

你们中的许多人在 Open Exchange 和 Github 上发布了 InterSystems ObjectScript 库。

但对于开发者来说，如何简化项目的使用和协作呢？

在本文中，我想介绍一种简单方法，只需一组标准文件复制到你的仓库中，就可以启动任何 ObjectScript 项目和对其做出贡献。

我们开始吧！

TLDR - 将以下文件从 该仓库 复制到你的仓库：

Dockerfile

docker-compose.yml

Installer.cls

iris.script

settings.json

.dockerignore
.gitattributes
.gitignore

你已经知道启动你的项目和协作的标准方式。以下将详细说明这样做的步骤和原因。

注意：在本文中，我们将考虑可在 InterSystems IRIS 2019.1 及更新版本上运行的项目。

选择 InterSystems IRIS 项目的启动环境

通常，我们鼓励开发者尝试项目/库，并确保这是快速安全的练习。

在我看来，快速安全地启动任何新项目的理想方式是 Docker 容器，它可以保护开发者启动、导入、编译和计算任何内容对于主机来说都是安全的，并且不会破坏任何系统或代码。如果出了问题，只需止并删除容器即可。
如果应用程序占用大量磁盘空间，使用容器将其擦除，空间就回来了。
如果某个应用程序破坏了数据库配置，只需删除配置被破坏的容器。简单又安全。

Docker 容器提供安全和标准化。

运行通用 InterSystems IRIS Docker 容器的最简单方法是运行 IRIS 社区版映像：

1. 安装 Docker desktop

2. 在操系统终端中运行以命令：

```
docker run --rm -p 52773:52773 --init --name my-iris store/intersystems/iris-community:2020.1.0.199.0
```

3. 然后在主机浏览器上打开管理门户：

http://localhost:52773/csp/sys/UtilHome.csp

4. 或者打开终端启动 IRIS：

   docker exec -it my-iris iris session IRIS

5. 不需要IRIS 容器时，将其停止：

   docker stop my-iris

好！我们在一个 docker 容器中运行 IRIS。但是你的开发者将你的代码安装到 IRIS 中，可能还要进行一些设置。这就是我们下面要讨论的。

导入 ObjectScript 文件

最简单的 InterSystems ObjectScript 项目可以包含一组 ObjectScript 文件，例如类、例程、宏和global。请查看有关命名和建议的文件夹结构的文章。

问题是如何将所有这些代码导入 IRIS 容器？

此时我们可以借助 Dockerfile 来获取通用 IRIS 容器，并将某个仓库中的所有代码导入 IRIS，然后根据需要对 IRIS 进行一些设置。我们需要在仓库中添加一个 Dockerfile。

我们来看一下 ObjectScript 模板仓库中的 Dockerfile：

```
ARG IMAGE=store/intersystems/irishealth:2019.3.0.308.0-community
ARG IMAGE=store/intersystems/iris-community:2019.3.0.309.0
ARG IMAGE=store/intersystems/iris-community:2019.4.0.379.0
ARG IMAGE=store/intersystems/iris-community:2020.1.0.199.0
FROM $IMAGE

USER root

WORKDIR /opt/irisapp
RUN chown ${ISC_PACKAGE_MGRUSER}:${ISC_PACKAGE_IRISGROUP} /opt/irisapp

USER irisowner

COPY  Installer.cls .
COPY  src src
COPY iris.script /tmp/iris.script # run iris and initial
```

```
RUN iris start IRIS \
    && iris session IRIS < /tmp/iris.script
```

前几个 ARG 行设置 $IMAGE 变量，随后在 FROM 中使用该变量。这适合在不同的 IRIS 版本中测试/运行代码，只需在 FROM 前面的最后一行中更改 $IMAGE 变量即可切换版本。

这里我们采用：

```
ARG IMAGE=store/intersystems/iris-community:2020.1.0.199.0

FROM $IMAGE
```

这意味着我们将使用 IRIS 2020 社区版 build 199。

我们想要导入仓库中的代码，这意味着我们需要将仓库中的文件复制到 docker 容器。下面几行完成操作：

```
USER root

WORKDIR /opt/irisapp
RUN chown ${ISC_PACKAGE_MGRUSER}:${ISC_PACKAGE_IRISGROUP} /opt/irisapp

USER irisowner

COPY  Installer.cls .
COPY  src src
```

`USER root` - 这里我们将用户切换为 root，以在 docker 中创建文件夹和复制文件。

`WORKDIR  /opt/irisapp` - 我们在此行中设置将文件复制到的工作目录。

`RUN chown ${ISC_PACKAGE_MGRUSER}:${ISC_PACKAGE_IRISGROUP} /opt/irisapp` - 这里我们为运行 IRIS 的 irisowner 用户和组授权限。

`USER irisowner` - 将用户从 root 切换到 irisowner

`COPY Installer.cls .` - 将 Installer.cls 复制到工作目录的根目录。不要漏了那个点儿。

`COPY src src` - 将仓库的 src 文件夹中的源文件复制到 docker 上的工作目录的 src 文件夹中。

在下一个块中，我们运行初始脚本，其中将调用安装程序和 ObjectScript 代码：

```
COPY iris.script /tmp/iris.script # run iris and initial
RUN iris start IRIS \
    && iris session IRIS < /tmp/iris.script
```

`COPY iris.script /` - 我们将 iris.script 复制到根目录。它包含我们要调用以设置容器的 ObjectScript。

`RUN iris start IRIS\` - 启动 IRIS

`&& iris session IRIS < /tmp/iris.script` - 启动 IRIS 终端并在其中输入初始 ObjectScript。

很好！我们有 Dockerfile，它将文件导入 docker。但还有两个文件：installer.cls 和 iris.script。我们来检查一下。

Installer.cls

```
Class App.Installer
{

XData setup
{
<Manifest>
  <Default Name="SourceDir" Value="#{$system.Process.CurrentDirectory()}src"/>
  <Default Name="Namespace" Value="IRISAPP"/>
  <Default Name="app" Value="irisapp" />

  <Namespace Name="${Namespace}" Code="${Namespace}" Data="${Namespace}" Create="yes"
 Ensemble="no">

    <Configuration>
      <Database Name="${Namespace}" Dir="/opt/${app}/data" Create="yes" Resource="%DB
_${Namespace}"/>

      <Import File="${SourceDir}" Flags="ck" Recurse="1"/>
    </Configuration>
    <CSPApplication Url="/csp/${app}" Directory="${cspdir}${app}"  ServeFiles="1" Rec
urse="1" MatchRoles=":%DB_${Namespace}" AuthenticationMethods="32"

    />
  </Namespace>

</Manifest>
}

ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 3, pInstaller As %Installer.In
staller, pLogger As %Installer.AbstractLogger) As %Status [ CodeMode = objectgenerato
r, Internal ]
{
  #; Let XGL document generate code for this method.
  Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "setup")
}

}
```

坦白说，我们不需要Installer.cls 来可以导入文件。这只需一行能完成
但除了导入代码之外，我们经常还需要设置 CSP 应用，引入安全设置，创建数据库和命名空间。

在此 Installer.cls 中，我们创建了名为 IRISAPP 的新数据库和命名空间，并为该命名空间创建了默认的 /csp/irisapp 应用程序。

所有这些都在元素中 执行：

```
<Namespace Name="${Namespace}" Code="${Namespace}" Data="${Namespace}" Create="yes" E
nsemble="no">

    <Configuration>
      <Database Name="${Namespace}" Dir="/opt/${app}/data" Create="yes" Resource="%DB
_${Namespace}"/>

      <Import File="${SourceDir}" Flags="ck" Recurse="1"/>
    </Configuration>
```

```
    <CSPApplication Url="/csp/${app}" Directory="${cspdir}${app}"  ServeFiles="1" Rec
urse="1" MatchRoles=":%DB_${Namespace}" AuthenticationMethods="32"

    />
  </Namespace>
```

我们用 Import 标签导入 SourceDir 中的所有文件：

```
<Import File="${SourceDir}" Flags="ck" Recurse="1"/>
```

这里的 SourceDir 是一个变量，设置为当前目录/src 文件夹：

```
<Default Name="SourceDir" Value="#{$system.Process.CurrentDirectory()}src"/>
```

有了带这些设置的 Installer.cls，我们可以自信地创建一个干净的新数据库 IRISAPP，我们将从 src 文件夹导入任意 ObjectScript 代码到该数据库中。

[iris.script](#)

欢迎在这里提供任何所需的初始 ObjectScript 设置代码来启动 IRIS 容器。

例如，我们加载并运行 installer.cls，然后让 UserPasswords 永远有效，以避免第一次启动时出现密码更改请求，因为开发不需要这个提示。

```
; run installer to create namespace
do $SYSTEM.OBJ.Load("/opt/irisapp/Installer.cls", "ck")
set sc = ##class(App.Installer).setup()  zn "%SYS"
Do ##class(Security.Users).UnExpireUserPasswords("*") ; call your initial methods her
e
halt
```

[docker-compose.yml](#)

为什么需要docker-compose.yml，不能只用 Dockerfile 构建和运行映像吗？可以的。但 docker-compose.yml 能让生活更轻松。

通常，docker-compose.yml 用于启动连接到一个网络的个 docker 映像。

当启动一个 docker 映像需要处理的个参数时，也可以使用 docker-compose.yml 来简化过程。你可以使用它向 docker 传递参数，例如端口映射、卷、VSCode 连接参数。

```
version: '3.6'
services:
  iris:
    build:
      context: .
      dockerfile: Dockerfile
    restart: always
    ports:
      - 51773
      - 52773
      - 53773
    volumes:
      - ~/iris.key:/usr/irissys/mgr/iris.key
```

```
      - ./:/irisdev/app
```

这里我们声明服务 iris，它使用 docker 文件 Dockerfile，并开放 IRIS 的以下端口：51773、52773、53773。
此服务还映射两个卷：将主机主目录中的 iris.key 映射到期望的 IRIS 文件夹，以及将源代码的根文件夹映射到 /irisdev/app 文件夹。

Docker-compose 提供了更短的统一命令来构建和运行映像，无论你在 docker compose 中设置了什么参数。

总之，构建和启动镜像的命令是：

```
$ docker-compose up -d
```

要打开 IRIS 终端：

```
$ docker-compose exec iris iris session iris

Node: 05a09e256d6b, Instance: IRIS

USER>
```

此外，docker-compose.yml 有助于设置 VSCode ObjectScript 插件的连接。

.vscode/settings.json

与 ObjectScript 加载项连接设置有关的部分如下：

```
{
    "objectscript.conn" :{
      "ns": "IRISAPP",
      "active": true,
      "docker-compose": {
        "service": "iris",
        "internalPort": 52773
      }
    }

}
```

在这里，我们到的设置与 VSCode ObjectScript 插件的默认设置不同。

我们想要连接到 IRISAPP 命名空间，我们用 Installer.cls 创建的：

```
"ns": "IRISAPP",
```

一个 docker-compose 设置指示，docker-compose 文件在服务“iris”内，VSCode 将连接到 52773 映射到的端口：

```
"docker-compose": {
        "service": "iris",
        "internalPort": 52773
      }
```

如果我们检查一下 52773 的相关设置，我们会到没有为 52773 定义映射端口：

```
ports:
      - 51773
      - 52773
      - 53773
```

这意味着将使用主机上的随机可用端口，并且 VSCode 将通过随机端口自动连接到 docker 上的 IRIS。

这是一个非常方便的功能，因为它允许在随机端口上运行任意数量的带 IRIS 的 docker 映像，并让 VSCode 自动连接到它们。

其他文件呢？

我们还有：

.dockerignore - 该文件可用于过滤你不想复制到所构建的 docker 映像中的主机文件。通常 .git 和 .DS_Store 是必须有的行。

.gitattributes - git 的属性用于统一来源中的 ObjectScript 文件的行尾。如果仓库由 Windows 和 Mac/Ubuntu 所有者协作，此文件非常有用。

.gitignore - 你不想 git 跟踪其更改历史记录的文件。通常是一些隐藏的操作系统级文件，例如 .DS_Store。

好了！

如何使你的仓库可被 docker 运行并且对协作友好？

1. 克隆 此仓库。
2. 复制以下所有文件：

Dockerfile

docker-compose.yml

Installer.cls

iris.script

settings.json

.dockerignore
.gitattributes
.gitignore

到你的仓库。

更改 Dockerfile 中的这一行 ，使目录与仓库中要导入 IRIS 的 ObjectScript 匹配（如果在 /src 文件夹中则不要更改）。

就这样。每个人（包括你）都会将你的代码导入到新的 IRISAPP 命名空间的 IRIS 中。

人们如何启动你的项目

在 IRIS 中执行任何 ObjectScript 项目的方法则为：

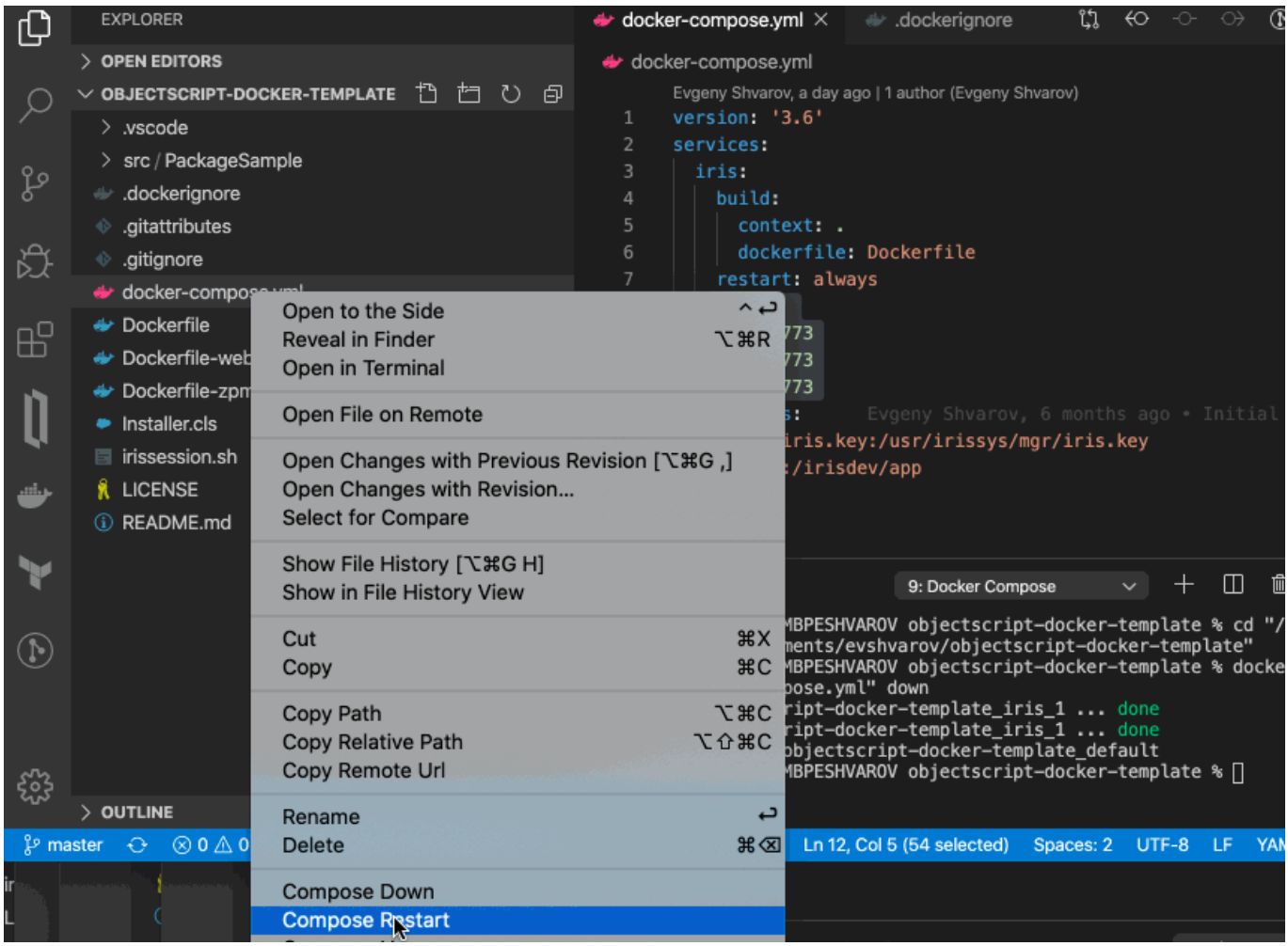
1. Git clone 项目到本地
2. 运行项目：

```
$ docker-compose up -d

$ docker-compose exec iris iris session iris

Node: 05a09e256d6b, Instance: IRIS

USER>zn "IRISAPP"
```

开发者如何为你的项目做出贡献

1. 对仓库执行分叉，并将分叉后的仓库 git clone 到本地
2. 在 VSCode 中打开该文件夹（还需在 VSCode 中安装 Docker 和 ObjectScript 扩展）
3. 右击 docker-compose.yml->重启 - VSCode ObjectScript 将自动连接并准备好编辑/编译/调试
4. 向你的仓库提交、推送和拉取请求更改

以下是操作过程的短 gif：



好了！编码愉快！

#Docker #Git #ObjectScript #开发环境 #教程 #InterSystems IRIS #Open Exchange
在 InterSystems Open Exchange 上检查相关应用程序

源 URL: https://cn.community.intersystems.com/post/dockerfile-%E5%92%8C%E5%AE%83%E7%9A%84%E6%9C%8B%E5%8F%8B%E4%BB%AC%E6%88%96%E8%80%85%E5%A6%82%E4%BD%95%E5%9C%A8-intersystems-iris-%E4%B8%8A%E8%BF%90%E8%A1%8C%E5%92%8C%E5%90%88%E4%BD%9C-objectscript-%E9%A1%B9%E7%9B%AE