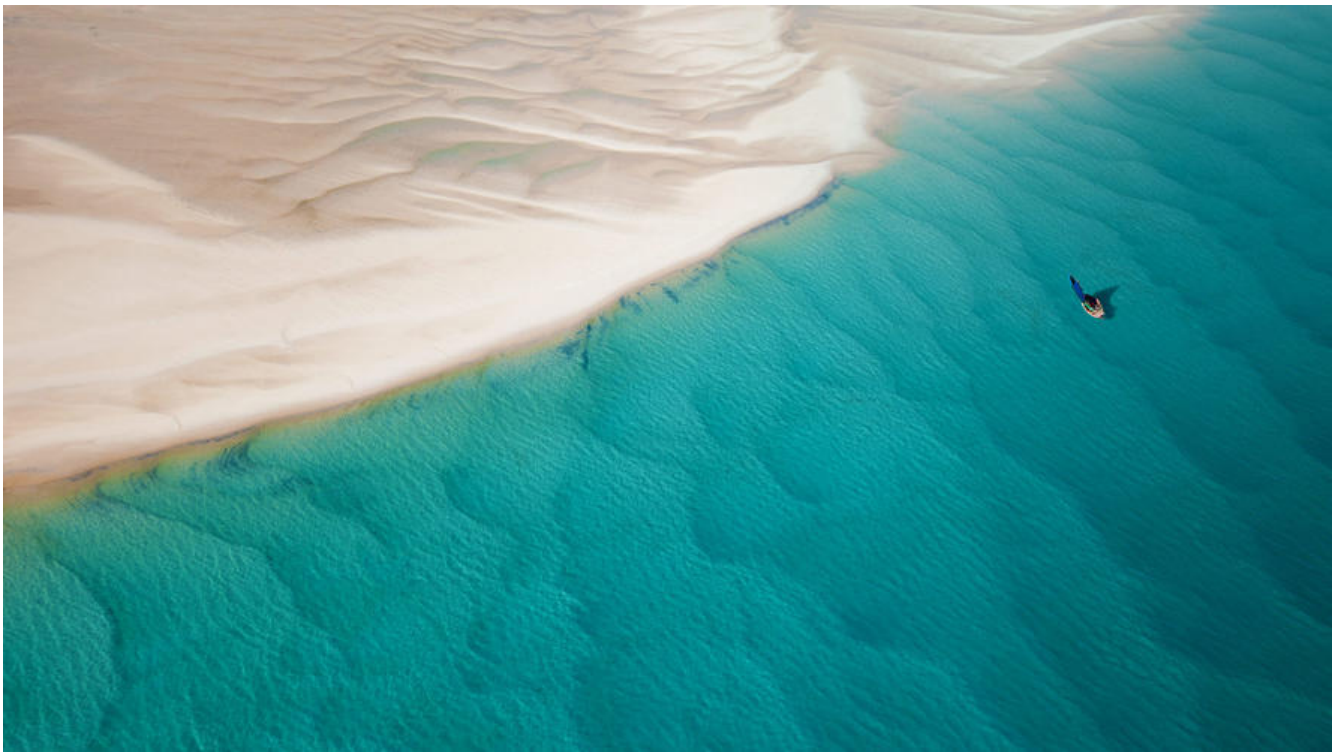
链滴

封印解除 -Solo 博客搭建日记!

作者：someone51937
原文链接：https://ld246.com/article/1568455138482
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



搭建Solo博客的过程，新卤煮作为第一次尝试，可谓是一言难尽。其重点不在代码的修改上，而是网站运维端的一些细节。

下面对本次Solo的成功部署及升级https过程，进行一个简要的文字总结，对于后期网站的部署和升也有一定借鉴意义。

下面正文开始，先列一个部署的前提清单：

名称	详情	备注
云服务器	若干台	可单机、可集群
域名	1个	配置dns解析
SSL证书	绑定域名	升级https用
Solo源代码	Solo	git到本地后进调整并用maven打包
系统环境	Linux+Java+MySQL+Tomcat+Nginx	indows系统也可行

从清单可以大致看出，本次我们想要部署方法的是非官方推荐的Docker部署，而是使用**Tomcat**的一方式，同时使用Nginx进行了**均衡负载**实现，并且加强了网站安全性，使用**https**对网站进行加密。

前三个东西需要预先从各种平台购买、获取和配置（包括DNS解析及备案），这里不再详细描述，从Slo项目开始。

0. Linux基本运行环境安装和数据库建库

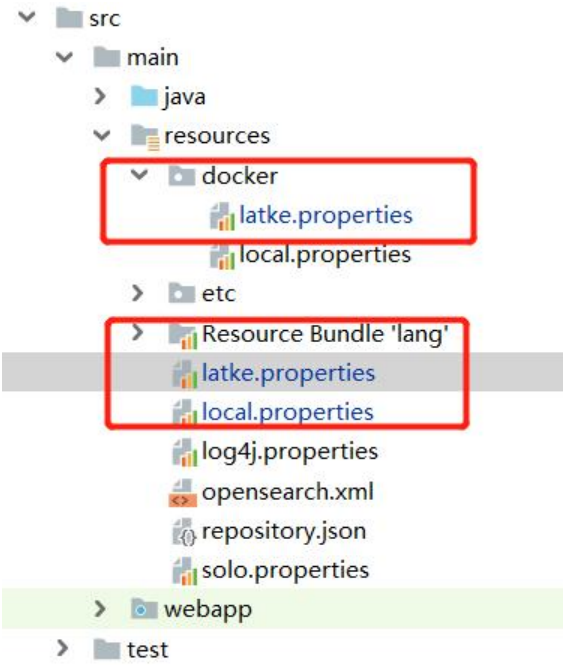
在Linux环境下先后需要安装**Java、MySQL、Tomcat、Nginx**，这里我们先进行基本安装，默认大都知道基本的使用，基础配置不再描述，关键配置后面再讲；

Solo的运行可以自行指定数据库类型，其中官方文档提供了两种方式H2和MySQl，这里我们选择MyQL；

登录MySQL并执行sql语句（数据库名称自定）：

```
CREATE DATABASE database_name DEFAULT CHARACTER SET UTF8MB4 COLLATE UTF8MB4 UNICODE_CI;
```

1. 代码配置修改



从Git到本地并向成功导入IDE的代码中，我们修改了蓝色的几个文件：

在latke中修改了请求协议以及运行环境（**其中请求方式改为https会影响网站部署后的静态资源请求址，https下是不允许网站混合请求http的**），如下：

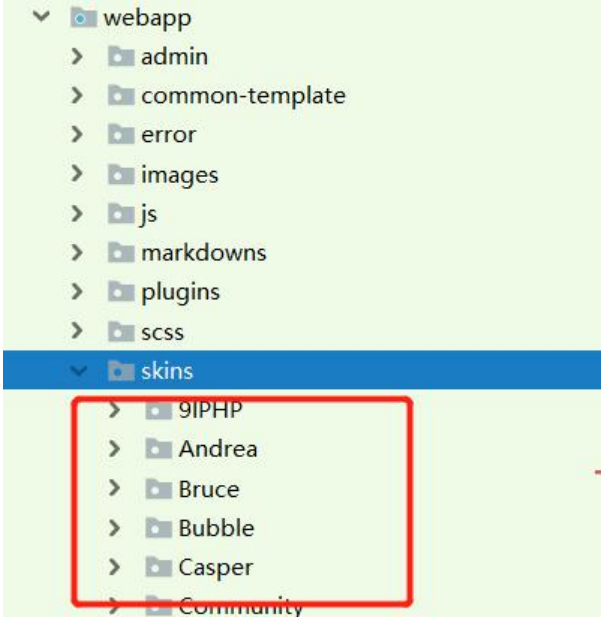
```
#### Server ####
# Browser visit protocol
serverScheme=https #如果不做https升级可以默认http
# Browser visit domain name
serverHost=localhost

#### Runtime Mode ####
### runtimeMode=DEVELOPMENT
runtimeMode=PRODUCTION
```

在local中我们修改了链接数据库的相关参数（根据用户名、密码参数自行配置）；

2. 小坑……皮肤在另外一个项目中

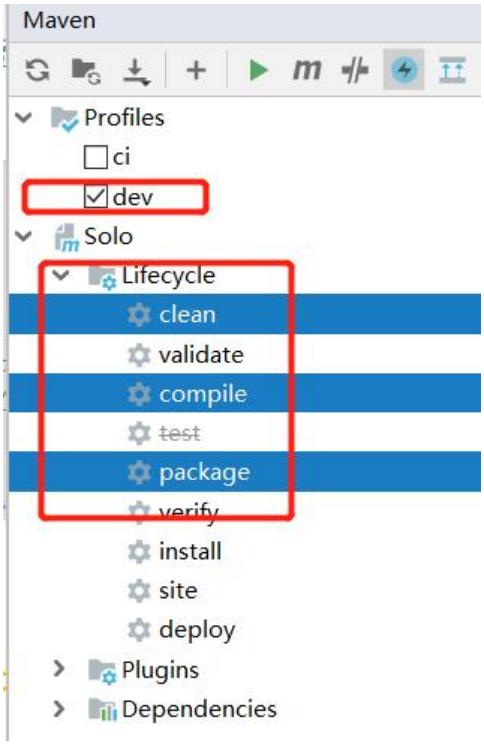
这里是一个坑，卤煮在第一次部署时发现，tomcat日志竟然报错，显示找不到皮肤，一查原因，原项目中已经不包含皮肤，但是却被默认指向一个Pinghsu的皮肤。经过查找，原来皮肤项目在这里：Solo-Skins

git过来后，将里面内容Copy到webapp/skins下面，目录存放如下图：



3. 打war包

带上dev配置，此配置使用了Aliyun的仓库镜像，比较快，推荐~



打包成功！

```
solo [compile,package]
[INFO] Processing war project
[INFO] Copying webapp resources                                    .solo\src\main\webapp]
[INFO] Webapp assembled in [3829 msec]
[INFO] Building war:                                    solo\target\solo.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 17.076 s
[INFO] Finished at: 2019-09-14T15:57:59+08:00
[INFO] ------------------------------------------------------------------------
un   6: TODO   9: Version Control   Terminal   Java Enterprise
```

4. war包上传

将war包上传到Tomcat目录下，启动Tomcat，war包会自动部署；

观察日志，Solo会自动初始化数据库，并提示成功：

```
[INFO ]-[2019-09-14 16:13:05]-[org.b3log.solo.SoloServletListener:99]: Solo is booting [ver=3.
.4, servletContainer=Apache Tomcat/xxx, os=Linux, isDocker=false, markdownHttpAvailable=
alse, pid=26689, runtimeDatabase=MYSQL, runtimeMode=PRODUCTION, jdbc.username=xxx
 jdbc.URL=jdbc:mysql://xxx/database_name?useUnicode=yes&characterEncoding=UTF-8&us
SSL=false&serverTimezone=UTC]
[INFO ]-[2019-09-14 16:13:05]-[com.zaxxer.hikari.HikariDataSource:110]: HikariPool-2 - Starti
g...
[INFO ]-[2019-09-14 16:13:06]-[com.zaxxer.hikari.HikariDataSource:123]: HikariPool-2 - Start c
mpleted.
[INFO ]-[2019-09-14 16:13:06]-[org.b3log.solo.SoloServletListener:138]: Solo is running
```

至此，通过域名到Tomcat的指定端口应该已经可以正常访问！

---

5. **高级部分-Nginx负载均衡**

以上基础步骤完成，现在想要进行高级点的试验——负载均衡，这个时候就是Nginx派上用场的时候；

在Linux下，修改nginx.conf文件：

```
upstream tomcatServer { #负载均衡到不同的服务器上去
 ip_hash;
 server ip1:port1 weight=200;
 server ip2:port2 weight=100;
}

server {
 listen 80;
 server_name localhost;
 location / { # 转发到配置的负载上去
  #index index.html index.htm;
  proxy_pass http://tomcatServer;
```

```
 }
 location ~.* { # 将静态资源转发到配置的负载上去
  proxy_pass http://tomcatServer;
  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 }
}
```

然后进行nginx Reload：

```
nginx -s reload
```

6. **高级部分-网站升级HTTPS**

此部分可谓是艰辛，网站升级https是一件非常麻烦的事，各种坑都有可能遇见，本文列出详细配置以供参考；

首先下载nginx证书，放到特定目录下，并对nginx进行443端口的https配置：

```
# 以下属性中以ssl开头的属性代表与证书配置有关，其他属性请根据自己的需要进行配置
server {
 listen 443;
 server_name localhost;  # localhost修改为您证书绑定的域名
 ssl on;   #设置为on启用SSL功能。
 root html;
 index index.html index.htm;
 ssl_certificate /xxx/domain.pem;   #将domain.pem替换成您证书的文件名，xxx为路径
 ssl_certificate_key /xxx/domain.key;   #将domain.key替换成您证书的密钥文件名，xxx为路径
 ssl_session_timeout 5m;
 ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!
DH:!RC4;  #使用此加密套件
 ssl_protocols TLSv1 TLSv1.1 TLSv1.2;   #使用该协议进行配置
 ssl_prefer_server_ciphers on;
 location / {
  #root html;   #站点目录
  #index index.html index.htm;
  proxy_pass https://tomcatServer;
 }
 location ~.* {
  proxy_pass https://tomcatServer;
  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto https;
 }
}
```

修改原80端口的配置，强制跳转https：

```
server {
 listen 80;
 server_name localhost;
 rewrite ^(.*)$ https://$host$1 permanent; #强行跳转
 location / {
```

```
  #index index.html index.htm;
  proxy_pass https://tomcatServer;
 }
 location ~.* {
  proxy_pass https://tomcatServer;
  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 }
}
```

然后reload nginx；

此时，直接访问域名应该已经可以看到nginx界面，并且提示网站证书是安全的；

接下来，进行Tomcat容器的https的配置，这里以**Tomcat8.0及以上版本**为例，其他版本可能有差别：

```
<!-- xxx.pfx是你的tomcat所用ssl证书的位置及文件名，password是证书密码，type为加密方式，
认RSA -->
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol" maxThrea
s="150" SSLEnabled="true">
        <SSLHostConfig>
            <Certificate certificateKeystoreFile="xxx.pfx" type="RSA" certificateKeystorePassword
"xxx" />
        </SSLHostConfig>
</Connector>
```

这里我们保留了Tomcat的8080端口和8443端口

重启Tomcat……

好了，大功告成，如果一切顺利的话，网站已经实现了负载均衡和https！

介于细节较多，文章只列举了核心的内容，其中滋味体验了才能知道，快来试试吧~不得不说Solo的面和项目设计都是非常赏心悦目的，这里给点个大大的赞！
humbsup thumbsup
humbsup thumbsup
humbsup

转载请注明出处，如有疑问欢迎留言~