Spring boot web容器实例化

目录

[1. 介绍](#_Toc26063_WPSOffice_Level1) [2](#_Toc26063_WPSOffice_Level1)

[1.1 默认配置：tomcat](#_Toc24807_WPSOffice_Level2) [2](#_Toc24807_WPSOffice_Level2)

[1.2 其他配置：undertow和jetty](#_Toc17514_WPSOffice_Level2) [2](#_Toc17514_WPSOffice_Level2)

[1.3 Web服务器的配置：application.properties](#_Toc489_WPSOffice_Level2) [3](#_Toc489_WPSOffice_Level2)

[2. Web 服务器相关的结构信息](#_Toc24807_WPSOffice_Level1) [4](#_Toc24807_WPSOffice_Level1)

[2.1 WebServer 结构](#_Toc25991_WPSOffice_Level2) [4](#_Toc25991_WPSOffice_Level2)

[2.1.1 接口定义](#_Toc24807_WPSOffice_Level3) [4](#_Toc24807_WPSOffice_Level3)

[2.1.2 实现类](#_Toc17514_WPSOffice_Level3) [4](#_Toc17514_WPSOffice_Level3)

[2.2 WebServerFactory 结构](#_Toc10702_WPSOffice_Level2) [4](#_Toc10702_WPSOffice_Level2)

[2.2.1 介绍](#_Toc489_WPSOffice_Level3) [4](#_Toc489_WPSOffice_Level3)

[2.2.2 TomcatServletWebServerFactory](#_Toc25991_WPSOffice_Level3) [4](#_Toc25991_WPSOffice_Level3)

[3. 启动WebServer的流程](#_Toc17514_WPSOffice_Level1) [6](#_Toc17514_WPSOffice_Level1)

[3.1 创建WebServer](#_Toc1347_WPSOffice_Level2) [6](#_Toc1347_WPSOffice_Level2)

[3.1.1 调用链](#_Toc10702_WPSOffice_Level3) [6](#_Toc10702_WPSOffice_Level3)

[3.1.2 创建过程详解](#_Toc1347_WPSOffice_Level3) [6](#_Toc1347_WPSOffice_Level3)

[3.2 启动WebServer](#_Toc22771_WPSOffice_Level2) [6](#_Toc22771_WPSOffice_Level2)

[3.2.1 调用链](#_Toc22771_WPSOffice_Level3) [6](#_Toc22771_WPSOffice_Level3)

[3.2.2 启动过程详解](#_Toc20200_WPSOffice_Level3) [6](#_Toc20200_WPSOffice_Level3)

1. 介绍

使用spring boot创建web项目不需要手动部署到诸如tomcat的服务容器上并启动，当项目运行起来的时候，spring boot会自动为创建、部署并启动web容器。Spring boot目前内嵌三种web容器，tomcat，undertow和jetty，其中tomcat是默认的web容器。

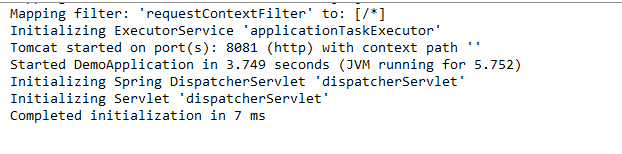
1.1 默认配置：tomcat

Pom.xml文件配置：引入web starter配置即可，starter-web的pom文件中引入了tomcat的starter

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



运行截图

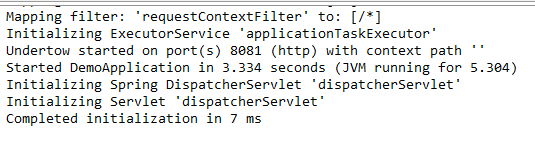


1.2 其他配置：undertow和jetty

Pom.xml文件设置：去掉默认的tomcat的starter配置并引入新容器的starter配置

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-tomcat</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-undertow</**artifactId**>  
</**dependency**>

运行截图



1.3 Web服务器的配置：application.properties

server 配置通用属性

Server.servlet 配置servlet项目的属性

server.tomcat 配置tomcat的属性

server.port=# 配置程序端口，默认8080

server.connection-timeout=# 配置连接超时时间

Server.server-header= #设置响应的Header中Server字段的值

server.servlet.session-timeout= #用户会话session过期时间，以秒为单位

server.servlet.context-path= #配置访问路径。默认为/

server.tomcat.uri-encoding = #配置Tomcat编码，默认UTF-8

server.tomcat.basedir = #Tomcat的根目录，如果没有指定，那么将使用临时目录

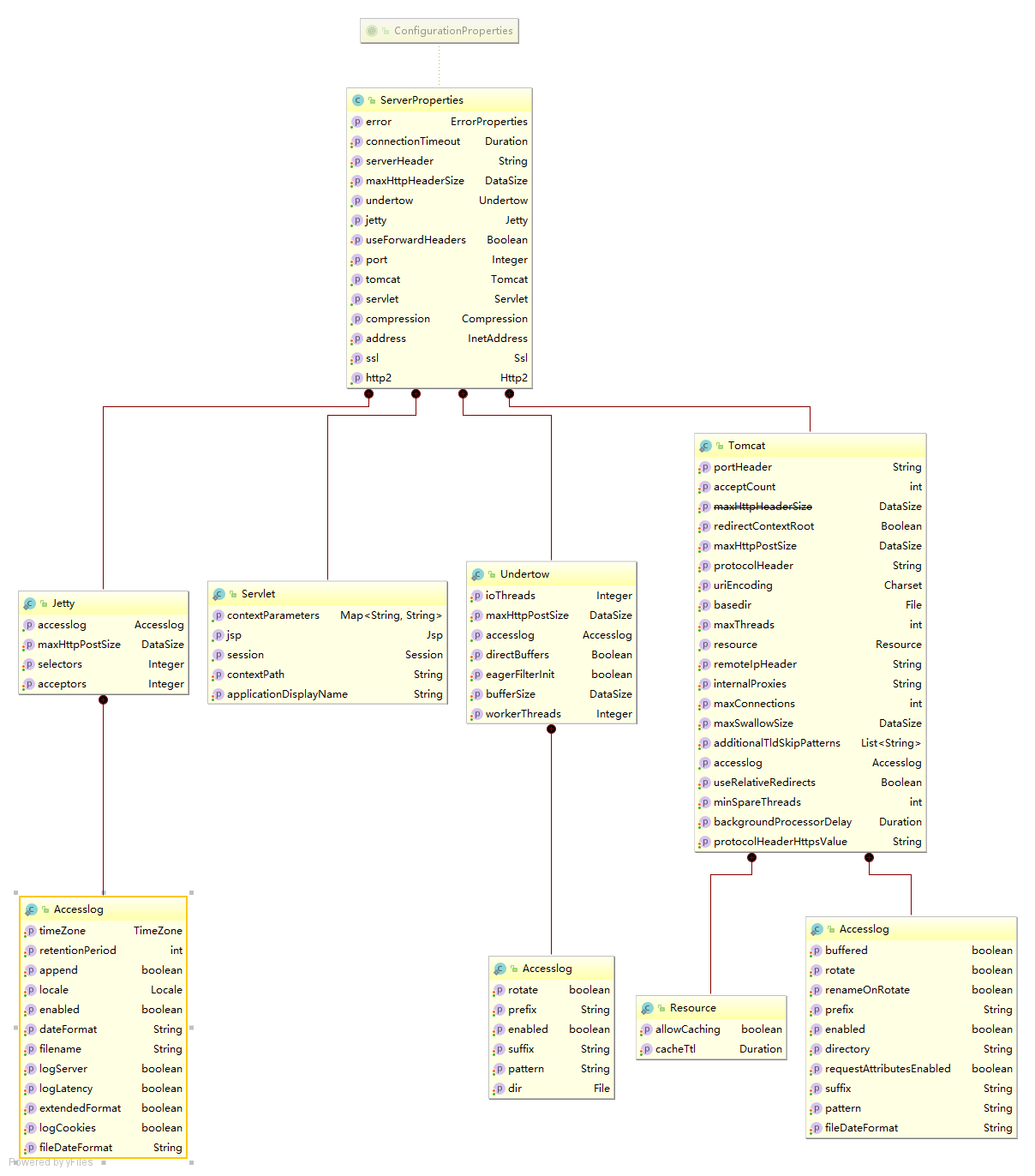
server.tomcat.max-connections= #Tomcat的最大连接数，默认值是10,000

server.tomcat.max-threads= #Tomcat的最大线程数，默认值是200

server.tomcat.resource.allow-caching= #Tomcat中静态资源是否使用缓存，默认开启

server.tomcat.resource.cache-ttl= #Tomcat中静态资源的缓存时间

可配置属性一览，配置相关类org.springframework.boot.autoconfigure.web.ServerProperties:



1. Web 服务器相关的结构信息
   1. WebServer 结构

2.1.1 接口定义

**public interface** WebServer {**void** start() **throws** WebServerException;**void** stop() **throws** WebServerException;**int** getPort();  
}

2.1.2 实现类

以tomcat的实现为例。

**public class** TomcatWebServer **implements** WebServer {

用来存储服务和与之关联的连接器

**private final** Map<Service, Connector[]> **serviceConnectors**;  
tomcat实例  
**private final** Tomcat **tomcat**;

构造函数，传入一个tomcat实例和一个是否自动启动的标志。

**public** TomcatWebServer(Tomcat tomcat, **boolean** autoStart);

初始化方法，在构造函数内被调用

**private void** initialize() **throws** WebServerException;

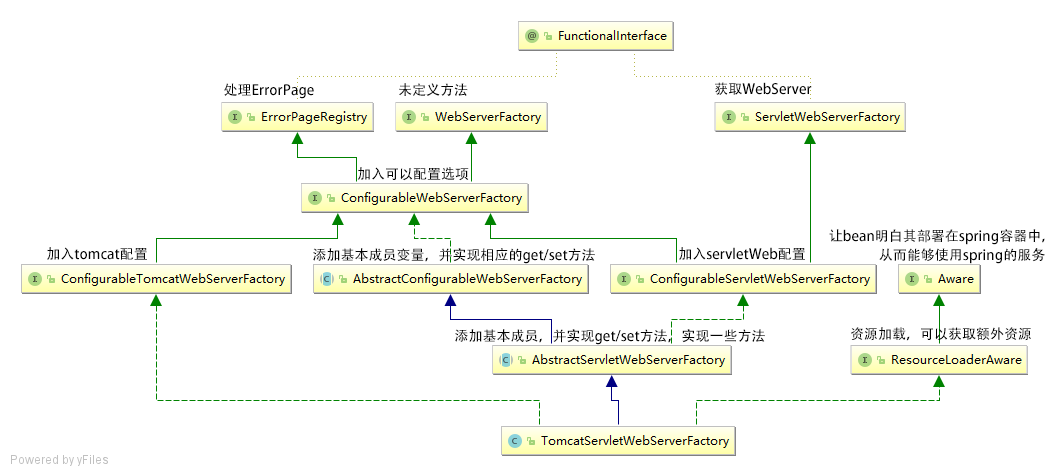
}

* 1. WebServerFactory 结构

2.2.1 介绍

依然以tomcat服务器为例。虽然TomcatWebServer的构造函数是公开的，但是在spring boot中并不推荐直接创建一个TomcatWebServer的实例，而是推荐使用相应的Factory来创建WebServer的实例。Springboot里实现了两个关于tomcat服务器的Factory，分别是TomcatReactiveWebServerFactory（对应于reactive web项目） 和TomcatServletWebServerFactory（对应于普通web项目）。

2.2.2 TomcatServletWebServerFactory



接口ServletWebServerFactory定义了获取WebServer的方法，getWebServer

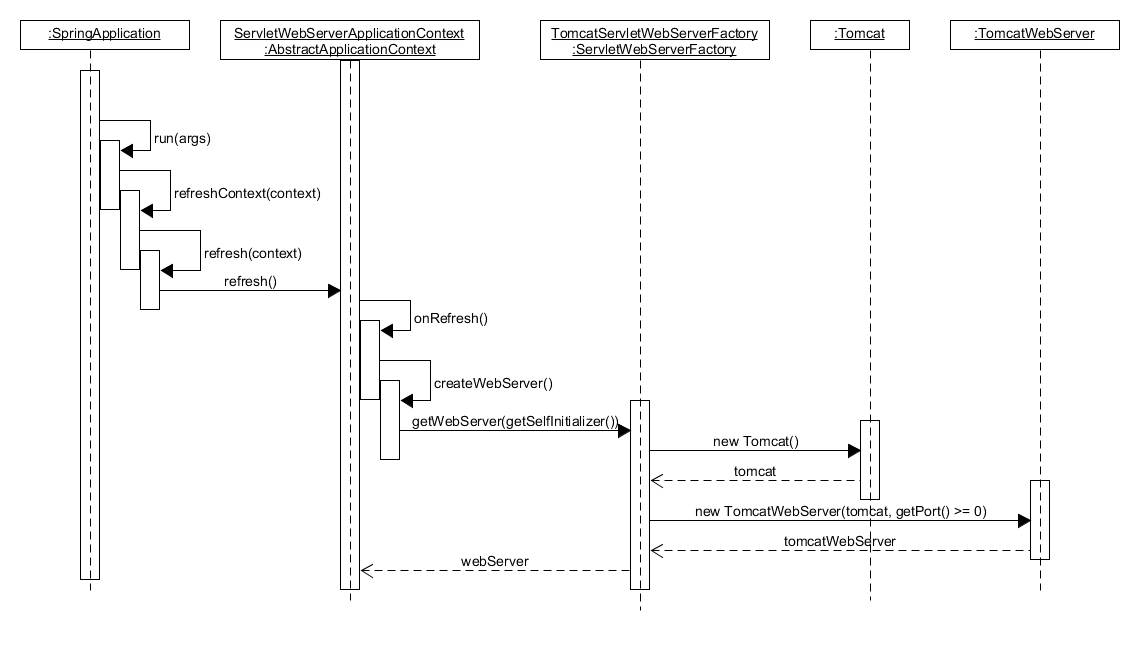
**public interface** ServletWebServerFactory {WebServer getWebServer(ServletContextInitializer... initializers);

}

类TomcatServletWebServerFactory实现了getWebServer方法并返回一个TomcatWebServer对象。

1. 启动WebServer的流程
   1. 创建WebServer
      1. 调用链

只保留了最直接关联的方法调用：



* + 1. 创建过程解析

（1）ServletWebServerApplicationContext.createWebServer方法：

先判断是否存在WebServer和servletContext的实例：第一次启动时是不存在这两个实例的，所以先获取WebServerFactory，然后通过Factory来创建WebServer的实例。

**private void** createWebServer() {  
 WebServer webServer = **this**.**webServer**;  
 ServletContext servletContext = getServletContext();  
 **if** (webServer == **null** && servletContext == **null**) {  
 ServletWebServerFactory factory = getWebServerFactory();   
 **this**.**webServer** = factory.getWebServer(getSelfInitializer());  
 }  
 **else if** (servletContext != **null**) {  
 **try** {  
 getSelfInitializer().onStartup(servletContext);  
 }  
 **catch** (ServletException ex) {  
 **throw new** ApplicationContextException(**"Cannot initialize servlet context"**,ex);  
 }  
 }  
 initPropertySources();  
}

（2）ServletWebServerApplicationContext.getWebServerFactory方法：

通过BeanFactory来获取TomcatServletWebServerFactory

**protected** ServletWebServerFactory getWebServerFactory() {String[] beanNames = getBeanFactory()  
 .getBeanNamesForType(ServletWebServerFactory.**class**);  
 **if** (beanNames.**length** == 0) {  
 **throw new** ApplicationContextException(  
 **"Unable to start ServletWebServerApplicationContext due to missing "**+ **"ServletWebServerFactory bean."**);  
 }  
 **if** (beanNames.**length** > 1) {  
 **throw new** ApplicationContextException(  
 **"Unable to start ServletWebServerApplicationContext due to multiple "** + **"ServletWebServerFactory beans : "** + StringUtils.*arrayToCommaDelimitedString*(beanNames));  
 }  
 **return** getBeanFactory().getBean(beanNames[0], ServletWebServerFactory.**class**);  
}

得到WebServerFactory对象后，调用WebServerFactory的getWebServer方法来获取WebServer。

（3）TomcatServletWebServerFactory.getWebServer方法：

**public** WebServer getWebServer(ServletContextInitializer... initializers) {  
 Tomcat tomcat = **new** Tomcat();  
 File baseDir = (**this**.**baseDirectory** != **null**) ? **this**.**baseDirectory** : createTempDir(**"tomcat"**);  
 tomcat.setBaseDir(baseDir.getAbsolutePath());  
 Connector connector = **new** Connector(**this**.**protocol**);  
 tomcat.getService().addConnector(connector);  
 customizeConnector(connector);  
 tomcat.setConnector(connector);  
 tomcat.getHost().setAutoDeploy(**false**);  
 configureEngine(tomcat.getEngine());  
 **for** (Connector additionalConnector : **this**.**additionalTomcatConnectors**) {  
 tomcat.getService().addConnector(additionalConnector);  
 }  
 prepareContext(tomcat.getHost(), initializers);  
 **return** getTomcatWebServer(tomcat);  
}

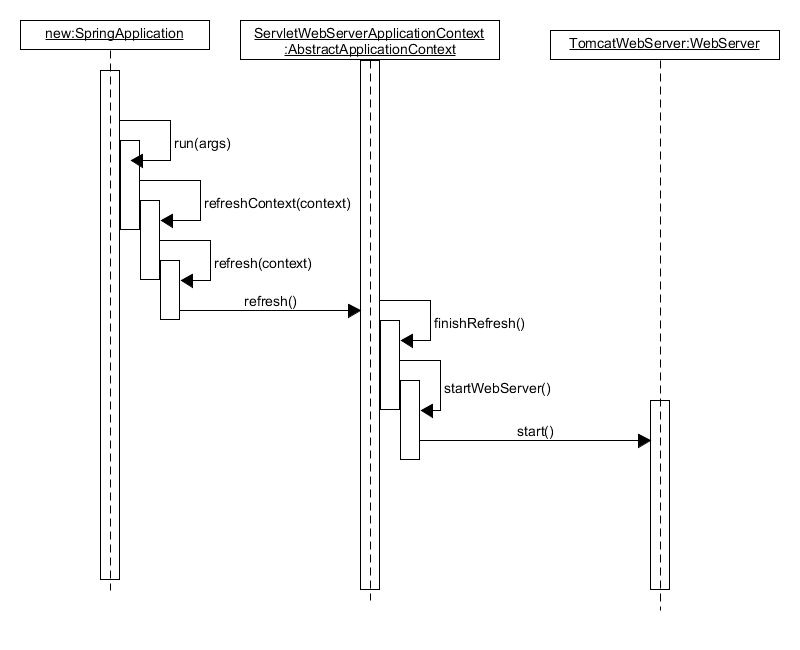
新实例化了一个tomcat对象，并对其进行配置，然后把利用这个tomcat对象实例一个tomcatWebServer并返回。

（4）TomcatServletWebServerFactory.getTomcatWebServer：

实例化一个tomcatWebServer并返回。

**protected** TomcatWebServer getTomcatWebServer(Tomcat tomcat) {  
 **return new** TomcatWebServer(tomcat, getPort() >= 0);  
}

* 1. 启动WebServer
     1. 调用链



* + 1. 启动过程解析

1. ServletWebServerApplicationContext.startWebServer

如果webServer不为空，则调用WebServer的start方法。

**private** WebServer startWebServer() {  
 WebServer webServer = **this**.**webServer**;  
 **if** (webServer != **null**) {  
 webServer.start();   
 }  
 **return** webServer;  
}