23.SpringApplication

SpringApplication类提供了一种方便的方式来引导将从main()方法启动的Spring应用程序。在很多情况下，你可以委托给静态的SpringApplication.run方法：

```
public static void main(String[]args)
{
    SpringApplication.run(MySpringConfiguration.class,args);
}
```

当你的应用程序启动时我们会看到如下信息：

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
 :: Spring Boot ::   v1.5.12.RELEASE

2013-07-31 00:08:16.117  INFO 56603 --- [           main]
o.s.b.s.app.SampleApplication            : Starting SampleApplication v0.1.0 on
mycomputer with PID 56603 (/apps/myapp.jar started by pwebb)
2013-07-31 00:08:16.166  INFO 56603 --- [           main]
ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationCo
ntext@6e5a8246: startup date [Wed Jul 31 00:08:16 PDT 2013]; root of context
hierarchy
2014-03-04 13:09:54.912  INFO 41370 --- [           main]
.t.TomcatEmbeddedServletContainerFactory : Server initialized with port: 8080
2014-03-04 13:09:56.501  INFO 41370 --- [           main]
o.s.b.s.app.SampleApplication            : Started SampleApplication in 2.992
seconds (JVM running for 3.658)
```

默认的情况下INFO信息将会被显示出来，包括相关的启动细节，例如启动应用的用户。

23.1 启动失败

如果你的应用程序启动失败了，被注册的FailureAnaluzers将会提供错误信息并且采取一定操作来解决这个问题。如果你想要在8080端口启动一个web应用，但是该端口被占用了，你会看到如下信息：

```
***************************
APPLICATION FAILED TO START
***************************
```

```
Description:

Embedded servlet container failed to start. Port 8080 was already in use.

Action:

Identify and stop the process that's listening on port 8080 or configure this
application to listen on another port.
```

Spring Boot 提供了多个`FailureAnalyzer`实现，并且你可以轻松添加你自己的实现。

如果没有`FailureAnalyzer`能处理异常，那么你可以去显示整个的自动配置报告，来让你更好的理解哪里出了问题。为了使用这种方式，你需要去启动debug选项或者启动对于`org.springframework.boot.autoconfigure.logging.AutoConfigurationReportLoggingInitializer`.的debug日志。

例如，如果你想要使用`java -jar`来启动你的应用程序，你可以像下面这样启动debug属性：

```
$ java -jar myproject-0.0.1-SNAPSHOT.jar --debug
```

23.2 Customizing the Banner

添加一个`banner.txt`文件到你的类路径或者设置`banner.location`属性只想一个这样的文件的位置，可以让开始时打印出的信息被更改。如果你想设置文件编码你可以通过`banner.charset`属性进行设置（默认是`UTF-8`）。除了一个文本文件以外，你也可以添加一个`banner.gif`,`banner.jpg`或者`banner.png`图像文件到你的类路径中，或者设置`banner.image.location`属性。图像将被转换成ASCII艺术表现形式并打印在任何文字横幅上方。

在banner.txt文件中，您可以使用以下任何占位符：

属性	描述
`${application.version}`	你在`MANIFEST.MF`中声明的应用程序版本数，例如`Implementation-Version: 1.0`被打印出来将会是`1.0`。
`${application.formatted-version}`	你在`MANIFEST.MF`中声明的应用程序版本数格式化显示（用括号包围并以`v`为前缀），例如`(v1.0)`。
`${spring-boot.version}`	你使用的Spring Boot 版本。例如`1.5.12.RELEASE`。
`${spring-boot.formatted-version}`	你使用的Spring Boot的格式化版本（用括号包围并以`v`为前缀）。例如`(v1.5.12.RELEASE)`。

属性	描述
${Ansi.NAME}或者（${AnsiColor.NAME}，AnsiBackground.NAME,AnsiStyle.NAME）	其中NAME是ANSI转义代码的名称。
${application.title}	在MANIFEST.MF中声明的应用题目。例如Implementation-Title:MyApp将会被打印为MyApp。

如果你想用编程的方法去规定一个banner，那么你可以使用SpringApplication.setBanner(...)。使用org.springframework.boot.Banner接口，并实现其printBanner()方法。

你可以使用spring.main.banner-mode属性去指定banner打印在System.out（console）上或者使用配置的logger（log）或者不打印（off）。

被打印的banner将会被注册成为一个名字为springBootBanner的bean。

23.3自定义SpringApplication

如果SpringApplication的默认形式不满足你的口味，你可以创建一个本地实例然后自定义他。例如，你可以按照下面的方法关闭banner。

```
public static void main(String[] args)
{
    SpringApplication app = new SpringApplication(MySpringConfiguration.class);
    app.setBannerMode(Banner.Mode.OFF);
    app.run(args);
}
```

传递给SpringApplication的构造函数参数是spring bean的配置源。 在大多数情况下，这些将是对@Configuration类的引用，但它们也可能是对XML配置或应扫描的包的引用。

也允许使用application.properties文件来配置SpringApplication。对于万丈的配置选项，看SpringApplicaiton文档吧。

23.4 Fluent builder API

如果你想要建立一个ApplicationContext层次结构（多个具有父子关系的上下文结构）,或者你仅仅想使用Fluent builder API，你可以使用SpringApplicationBuilder。

SpringApplicaitonBuilder允许你去链式地调用多个方法，并且parent和child方法允许你去创建一个层次关系。

例如：

```
new SpringApplicationBuilder()
    .source(Parent.class)
    .child(Application.class)
```

```
        .bannerMode(Banner.Mode.OFF)
        .run(args);
```

创建ApplicationContext层次结构时有一些限制，例如，Web组件必须包含在子上下文中，并且相同的环境将用于父和子上下文。关于更多细节，请查看SpringApplicationBuilder的文档。

23.5 应用事件和监听器

除了通常的Spring 框架事件，例如ContextRefreshedEvent，SpringApplication还会产生另外的应用事件。

一些事件会在ApplicationContext被创建之前触发，所以你不能将监听器注册为一个@Bean。你可以通过SpringApplication.addListener(...)或SpringApplicationBuilder.listeners(...)方法注册它们。

如果你想要这些监听器自动被注册，不管应用的创建方式如何，那么你在你的应用中可以添加一个META_INF/spring.factories文件，和使用org.springframework.context.ApplicationListener键去指向这个监听器引用：

```
org.springframework.context.ApplicationListener=com.example.project.MyListener
```

当你的应用程序运行时，应用事件通过下面顺序发送：

1. ApplicationStartingEvent当程序运行的开始，在除了监听器和初始化器注册之前的所有操作时发送。
2. ApplicationEnvironmentPreparedEvent是上下文中的Environment被使用时才会被发送，而不是上下文被创建时。
3. ApplicationPreparedEvent在刷新开始前被发送，而不是bean定义被加载前。
4. ApplicationReadyEvent 在刷新之后，并且处理好相关的回调的情况下发送，这指示了应用程序已准备好为请求提供服务。
5. 如果启动中出现异常会发送ApplicationFailedEvent。

你其实不会去使用这些事件，但是知道它们存在是很方便的。在Spring Boot的内部，其使用这些事件来处理一系列任务。

应用程序事件使用Spring Framework的事件发布机制发送。这个机制的一部分确保发布到子上下文的事件也会发布到其父上下文。正因如此，如果你的应用程序使用的是一个具有层次结构的SpringApplication实例，那么一个监听器可能会接收到多个相同类型的应用事件实例。

为了使监听器能够区分这个事件到底是它的事件还是它后代的事件，它会比较被注入的应用上下文，然后将其与事件上下文进行比较。我们靠实现ApplicationContextAware接口，如果监听器是一个bean，那么你要使用@Autowired。

23.6 Web环境

SpringApplication将尝试代表您创建正确类型的ApplicationContext。默认情况下，AnnotationConfigEmbeddedWebApplicaiton和AnnotatpionConfigEmbeddedWebApplicaitonContext将会被使用，这将取决于拟开发的应用程序是否是一个web应用程序。

用于确定“Web环境”的算法相当简单（基于少数类的存在）。如果你想去覆盖默认的环境，你可以去使用`setWebEnvironment(boolean webEnvironment)`方法。

我们可以调用`setApplicationContextClass(...)`来获取对`ApplicationContext`的完全控制。

当使用`SpringApplication`的JUnit测试时，我们通常要调用`setWebEnvironment(false)`。

23.7 访问应用参数

如果你需要去访问传递给`SpringApplication.run(...)`的参数，你可以去注入一个`org.springframework.boot.ApplicationArguments`bean。`ApplicationArguments`接口提供了对于原始`String[]`参数和被解析的`option`和`non-option`参数进行访问：

```
import org.springframework.boot.*;
import org.springframework.beans.annotation.*;
import org.springframework.stereotype.*;

@Component
public class MyBean
{
    @Autowired
    public MyBean(ApplicationArguments args)
    {
        boolean debug = args.containsOption("debug");
        List<String> files = args.getNoneOptionArgs();
        //如果使用“--debug logfile.txt”运行的话，debug=true，files=["logfile.txt"]
    }
}
```

Spring Boot将会使用Spring的`Environment`注册一个`CommandLinePropertySource`。这个做法允许你去使用`@Value`去注入单一的应用参数值。

23.8 使用ApplicationRunner或者CommandLineRunner

如果你希望某些特定代码在你的应用程序开始时就进行执行，你可以实现`ApplicationRunner`或者`CommandLineRunner`接口。两个接口都以相同的方式工作，并提供了一个将在`SpringApplication.run(...)`完成之前调用的单个运行方法。

这个`CommandLineRunner`接口提供了以字符串数组的方式访问应用参数的支持，而`ApplicationRunner`使用上面讨论的`ApplicationArguments`接口。

```
import org.springframework.boot.*;
import org.springframework.stereotype.*;

@Component
public class MyBean implements CommandLineRunner
{
    public void run(String... args)
```

```
    {
        //Do something...
    }
}
```

如果你有几个CommandLineRunner或ApplicationRunnerbean被定义，那么你想去给他们指定一个特定的顺序，你可以实现org.springframework.core.Orderd接口或者使用org.springframework.core.annotation.Order注解。

23.9 应用退出

每个SpringApplication都会向JVM注册一个关闭钩子，以确保ApplicationContext在退出时正常关闭。可以使用所有标准的Spring生命周期回调（如DisposableBean接口或@PreDestroy注释）。

另外，如果你想要在SpringApplication.exit()被调用后返回一个特定的退出码，那么你可以去创建一个实现了org.springframework.boot.ExitCodeGenerator接口的bean。

```
@SpringBootApplication
public class ExitCodeApplication
{
    @Bean
    public ExitCodeGenerator exitCodeGenerator()
    {
        return new ExitCodeGenerator()
        {
            @Override
            public int getExitCode()
            {
                return 42;
            }
        }
    }

    public static void main(String[] args)
    {
        System.exit(SpringApplication
                        .exit(SpringApplication.run(ExitCodeApplication.class,
args)));
    }
}
```

除此之外，ExitCodeGenerator接口可能靠异常实现。遇到这样的异常时，Spring Boot将返回由实现的getExitCode（）方法提供的退出代码。

23.10 管理功能

我们可以通过spring.application.admin.enabled属性来启动与管理相关的功能。这暴露了平台MBeanServer上的SpringApplicationAdminMXBean。你可以使用这个功能远程管理Spring Boot应用。这对

于任何服务包装器实现也是有用的。

如果你想要知道程序运行在哪个HTTP端口，你可以通过key`local.server.port`来获得这个属性。

启动这个服务MBean会暴露一个关闭程序的方法，所以请小心。