**附录A.通用应用程序属性**

可以在application.properties文件内部application.yml，文件内部或命令行开关中指定各种属性。本附录提供了常用Spring Boot属性的列表以及对使用它们的基础类的引用。

*＃================================================= ==================*

*＃COMMON SPRING BOOT PROPERTIES*

*＃＃*

*此示例文件作为指导提供。请勿将其*

*全部内容*

*复制到您自己的应用程序中。^^^ ＃============================================== =====================*

*＃----------------------------------------*

*＃核心属性*

*＃----- -----------------------------------*

debug = false *＃启用调试日志。*

trace = false *＃启用跟踪日志。*

*＃LOGGING*

logging.config = *＃日志配置文件的位置。例如，Logback的`classpath：logback.xml`。*

logging.exception-conversion-word =%wEx *＃记录异常时使用的转换字。*

logging.file = *＃日志文件名（例如`myapp.log`）。名称可以是确切的位置或相对于当前目录。*

logging.file.max-history = 0 *＃要保留的归档日志文件的最大数量。仅支持默认的登录设置。*

logging.file.max-size = 10MB *＃最大日志文件大小。仅支持默认的登录设置。*

logging.level.\* =*＃日志级别严重性映射。例如`logging.level.org.springframework = DEBUG`。*

logging.path = *＃日志文件的位置。例如，`/ var / log`。*

logging.pattern.console = *＃输出到控制台的Appender模式。仅使用默认的Logback设置支持。*

logging.pattern.dateformat = yyyy-MM-dd HH：mm：ss.SSS *＃日志格式的Appender模式。仅使用默认的Logback设置支持。*

logging.pattern.file = *＃输出到文件的Appender模式。仅使用默认的Logback设置支持。*

logging.pattern.level =%5p *＃日志级别的Appender模式。仅使用默认的Logback设置支持。*

logging.register-shutdown-hook = false *＃为日志记录系统初始化时注册一个关闭钩子。*

*＃AOP*

spring.aop.auto =真*＃添加@EnableAspectJAutoProxy。*

spring.aop.proxy-target-class = true *＃是否创建基于子类的（CGLIB）代理（true），而不是基于标准Java接口的代理（false）。*

＃IDENTITY *（*[ContextIdApplicationContextInitializer](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/ContextIdApplicationContextInitializer.java)）

spring.application.name = *＃应用程序名称。*

[＃ADMIN](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.java) *（*[SpringApplicationAdminJmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.java)）

spring.application.admin.enabled = false *＃是否为应用程序启用管理功能。*

spring.application.admin.jmx-name = org.springframework.boot：type = Admin，name = SpringApplication *＃JMX应用程序的名称admin MBean。*

*＃AUTO-CONFIGURATION*

spring.autoconfigure.exclude = *＃要排除的自动配置类。*

*＃BANNER*

spring.banner.charset = UTF-8 *＃横幅文件编码。*

spring.banner.location = classpath：banner.txt *＃横幅文本资源位置。*

spring.banner.image.location = classpath：banner.gif *＃横幅图像文件位置（也可以使用jpg或png）。*

spring.banner.image.width = 76 *＃字符图片的宽度。*

spring.banner.image.height = *＃以字符形式显示横幅图像的高度（默认基于图像高度）。*

spring.banner.image.margin = 2 *＃在字符中留下左手边缘图像。*

spring.banner.image.invert = false *＃图像是否应该反转为黑暗的终端主题。*

＃SPRING *CORE* spring.beaninfo.ignore = true *＃是否跳过对BeanInfo类的搜索。*

＃SPRING *CACHE（*[CacheProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cache/CacheProperties.java)）

spring.cache.cache-names = *＃如果基础高速缓存管理器支持，则创建缓存名称的逗号分隔列表。*

spring.cache.caffeine.spec = *＃用于创建缓存的规范。有关规格格式的更多详细信息，请参阅CaffeineSpec。*

spring.cache.couchbase.expiration = 0ms *＃进入到期。默认情况下，这些条目永不过期。请注意，该值最终转换为秒。*

spring.cache.ehcache.config = *＃用于初始化EhCache的配置文件的位置。*

spring.cache.infinispan.config = *＃用于初始化Infinispan的配置文件的位置。*

spring.cache.jcache.config = *＃用于初始化缓存管理器的配置文件的位置。*

spring.cache.jcache.provider = *＃用于检索符合JSR-107的缓存管理器的CachingProvider实现的完全限定名称。只有在类路径中有多个JSR-107实现可用时才需要。*

spring.cache.redis.cache-null-values = true *＃允许缓存空值。*

spring.cache.redis.key-prefix = *＃键字前缀。*

spring.cache.redis.time-to-live = 0ms *＃进入到期。默认情况下，这些条目永不过期。*

spring.cache.redis.use-key-prefix = true*＃写入Redis时是否使用密钥前缀。*

spring.cache.type = *＃缓存类型。默认情况下，根据环境自动检测。*

*＃SPRING CONFIG - 仅使用环境属性（*[ConfigFileApplicationListener](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/config/ConfigFileApplicationListener.java)）

spring.config.additional-location = *＃除默认值之外使用的配置文件位置。*

spring.config.location = *＃配置替换默认值的文件位置。*

spring.config.name = application *＃配置文件名。*

*＃HAZELCAST（*[HazelcastProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastProperties.java)）

spring.hazelcast.config = *＃用于初始化Hazelcast的配置文件的位置。*

*＃项目信息（*[ProjectInfoProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/info/ProjectInfoProperties.java)）

spring.info.build.location = classpath：META-INF / build-info.properties *＃生成的build-info.properties文件的位置。*

spring.info.git.location =类路径：git.properties *生成的git.properties文件＃所在。*

*＃JMX*

spring.jmx.defaultfield = *＃JMX域名。*

spring.jmx.enabled = true *＃将管理bean展示给JMX域。*

spring.jmx.server = mbeanServer *＃MBeanServer bean名称。*

*＃电子邮件（*[MailProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailProperties.java)）

spring.mail.default-encoding = UTF-8 *＃默认MimeMessage编码。*

spring.mail.host = *＃SMTP服务器主机。例如，`smtp.example.com`。*

spring.mail.jndi-name = *＃会话JNDI名称。设置时，优先于其他邮件设置。*

spring.mail.password = *＃登录SMTP服务器的密码。*

spring.mail.port = *＃SMTP服务器端口。*

spring.mail.properties.\* = *＃其他JavaMail会话属性。*

spring.mail.protocol = smtp *＃SMTP服务器使用的协议。*

spring.mail.test-connection = false*＃是否测试邮件服务器在启动时是否可用。*

spring.mail.username = *＃登录SMTP服务器的用户。*

*＃应用程序设置（*[SpringApplication](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/SpringApplication.java)）

spring.main.banner-mode = console *＃用于在应用程序运行时显示*

标题的模式*。*spring.main.sources = *＃包含在ApplicationContext中的源（类名，包名或XML资源位置）。*

spring.main.web-application-type = *＃显式请求特定类型的Web应用程序的标志。如果未设置，则根据类路径自动检测。*

[＃FILE](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/FileEncodingApplicationListener.java) *ENCODING（*[FileEncodingApplicationListener](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/FileEncodingApplicationListener.java)）

spring.mandatory-file-encoding = *＃应用程序必须使用的期望字符编码。*

＃INTERNATIONALIZATION *（*[MessageSourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/MessageSourceProperties.java)）

spring.messages.always-use-message-format = false *＃是否始终应用MessageFormat规则，甚至可以解析不带参数的消息。*

spring.messages.basename = messages *＃以逗号分隔的基本名称列表（本质上是一个完全限定的类路径位置），每个都遵循ResourceBundle约定，对基于斜杠的位置提供宽松的支持。*

spring.messages.cache-duration = *＃加载的资源包文件缓存持续时间。未设置时，捆绑包将永久缓存。如果未指定持续时间后缀，则将使用秒。*

spring.messages.encoding = UTF-8 *＃消息包编码。*

spring.messages.fallback-to-system-locale = true *＃是否回退到系统区域设置，如果没有找到特定语言环境的文件。*

spring.messages.use-code-as-default-message = false *＃是否使用消息代码作为默认消息，而不是抛出“NoSuchMessageException”。仅在开发期间推荐。*

*＃OUTPUT*

spring.output.ansi.enabled =检测*＃配置的ANSI输出。*

＃PID *FILE（*[ApplicationPidFileWriter](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/ApplicationPidFileWriter.java)）

spring.pid.fail-on-write-error = *＃如果使用ApplicationPidFileWriter，将失败，但不能写入PID文件。*

spring.pid.file = *＃要写入的PID文件的位置（如果使用ApplicationPidFileWriter）。*

*＃PROFILES*

spring.profiles.active = *＃逗号分隔的有源配置文件列表。可以被命令行开关覆盖。*

spring.profiles.include = *＃无条件激活指定的以逗号分隔的配置文件列表（或使用YAML配置文件列表）。*

*＃Quartz调度（*[QuartzProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/quartz/QuartzProperties.java)）

spring.quartz.jdbc.initialize-架构 =嵌入*＃数据库模式初始化模式。*

spring.quartz.jdbc.schema = classpath中：组织/石英/ IMPL / jdbcjobstore / tables\_ @ @ 平台@ @ .SQL *＃的路径SQL文件，以用于初始化数据库架构。*

spring.quartz.job-store-type =内存*＃石英作业存储类型。*

spring.quartz.properties.\* = *＃额外的Quartz Scheduler属性。*

[＃REACTOR](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/reactor/core/ReactorCoreProperties.java) *（*[ReactorCoreProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/reactor/core/ReactorCoreProperties.java)）

[spring.reactor.stacktrace](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/reactor/core/ReactorCoreProperties.java) -mode.enabled = false *＃Reactor是否应该在运行时收集*堆栈跟踪*信息。*

*＃SENDGRID（*[SendGridAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.java)）

spring.sendgrid.api-key = *＃SendGrid API密钥。*

spring.sendgrid.proxy.host = *＃SendGrid代理主机。*

spring.sendgrid.proxy.port = *＃SendGrid代理端口。*

*＃----------------------------------------*

*＃WEB PROPERTIES*

*＃----- -----------------------------------*

*＃嵌入式服务器配置（*[ServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java)）

server.address = *＃服务器应绑定到的网络地址。*

server.compression.enabled = false *＃是否启用响应压缩。*

server.compression.excluded-user-agents = *＃要从压缩中排除的用户代理列表。*

server.compression.mime-types = text / html，text / xml，text / plain，text / css，text / javascript，application / javascript *＃应该压缩的逗号分隔的MIME类型列表。*

server.compression.min-response-size = 2048 *＃压缩执行所需的最小“Content-Length”值。*

server.connection超时= *＃连接器在关闭连接之前等待另一个HTTP请求的时间。未设置时，使用连接器的容器特定默认值。使用值-1来表示否（即无限）超时。*

server.error.include-exception = false *＃包含“exception”属性。*

server.error.include-stacktrace = never *＃何时包含“stacktrace”属性。*

server.error.path = / error *＃错误控制器的路径。*

server.error.whitelabel.enabled = true *＃是否在服务器出错时启用浏览器中显示的默认错误页面。*

server.http2.enabled = false*＃如果当前环境支持，是否启用HTTP / 2支持。*

server.jetty.acceptors = *＃要使用的接受者线程的数量。*

server.jetty.accesslog.append = false *＃附加到日志。*

server.jetty.accesslog.date-format = dd/MMM/yyyy:HH:mm:ss Z *＃请求日志的时间戳格式。*

server.jetty.accesslog.enabled = false *＃启用访问日志。*

server.jetty.accesslog.extended-format = false *＃启用扩展的NCSA格式。*

server.jetty.accesslog.file-date-format = *＃放置在日志文件名中的日期格式。*

server.jetty.accesslog.filename =*＃日志文件名。如果未指定，日志重定向到“System.err”。*

server.jetty.accesslog.locale = *＃请求日志的语言环境。*

server.jetty.accesslog.log-cookies = false *＃启用记录请求cookie。*

server.jetty.accesslog.log-latency = false *＃启用记录请求处理时间。*

server.jetty.accesslog.log-server = false *＃启用对请求主机名的记录。*

server.jetty.accesslog.retention-period = 31 *＃删除旋转的日志文件之前的天数。*

server.jetty.accesslog.time-zone = GMT *＃请求日志的时区。*

server.jetty.max-http-post-size = 0*＃HTTP帖子或放置内容的最大大小（以字节为单位）。*

server.jetty.selectors = *＃要使用的选择器线程数。*

server.max-http-header-size = 0 *＃HTTP消息头的最大大小（以字节为单位）。*

server.port = 8080 *＃服务器HTTP端口。*

server.server-header = *＃用于服务器响应头的值（如果为空，则不会发送头）。*

server.use-forward-headers = *＃是否应将X-Forwarded- \*标头应用于HttpRequest。*

server.servlet.context-parameters.\* = *＃Servlet上下文初始化参数。*

server.servlet.context-path = *＃应用程序的上下文路径。*

server.servlet.application-display-name = application *＃显示*

应用程序的*名称。*server.servlet.jsp.class-name = org.apache.jasper.servlet.JspServlet *＃JSP servlet的类名称。*

server.servlet.jsp.init-parameters.\* = *＃用于配置JSP servlet的初始参数。*

server.servlet.jsp.registered = true *＃JSP servlet是否已注册。*

server.servlet.path = / *＃主调度程序servlet的路径。*

server.servlet.session.cookie.comment = *＃评论会话cookie。*

server.servlet.session.cookie.domain = *＃会话cookie的域名。*

server.servlet.session.cookie.http-only = *＃会话cookie的“HttpOnly”标志。*

server.servlet.session.cookie.max-age = *＃会话cookie的最大年龄。如果未指定持续时间后缀，则将使用秒。*

server.servlet.session.cookie.name = *＃会话cookie名称。*

server.servlet.session.cookie.path = *＃会话cookie的路径。*

server.servlet.session.cookie.secure = *＃会话cookie的“安全”标志。*

server.servlet.session.persistent = false *＃是否在重新启动之间保留会话数据。*

server.servlet.session.store-dir = *＃用于存储会话数据的目录。*

server.servlet.session.timeout = *＃会话超时。如果未指定持续时间后缀，则将使用秒。*

server.servlet.session.tracking-modes = *＃会话跟踪模式（以下一项或多项：“cookie”，“url”，“ssl”）。*

server.ssl.ciphers = *＃支持的SSL密码。*

server.ssl.client-auth = *＃是否需要客户端身份验证（“需要”）或需要（“需要”）。需要信任商店。*

server.ssl.enabled = *＃启用SSL支持。*

server.ssl.enabled-protocols = *＃启用SSL协议。*

server.ssl.key-alias = *＃标识密钥库中密钥的别名。*

server.ssl.key-password = *＃用于访问密钥存储区中密钥的密码。*

server.ssl.key-store = *＃保存SSL证书的密钥存储区的路径（通常是一个jks文件）。*

server.ssl.key-store-password = *＃用于访问密钥存储区的密码。*

server.ssl.key-store-provider = *＃密钥存储的提供者。*

server.ssl.key-store-type = *＃密钥存储的类型。*

server.ssl.protocol = *要使用的*

TLS *＃SSL协议。*server.ssl.trust-store = *＃持有SSL证书的信任库。*

server.ssl.trust-store-password = *＃用于访问信任存储的密码。*

server.ssl.trust-store-provider = *＃信任存储的提供程序。*

server.ssl.trust-store-type = *＃信任存储的类型。*

server.tomcat.accept-count = 0 *＃所有可能的请求处理线程正在使用时传入连接请求的最大队列长度。*

server.tomcat.accesslog.buffered = true *＃是否缓冲输出，使其仅定期刷新。*

server.tomcat.accesslog.directory = logs *＃创建日志文件的目录。可以是绝对的或相对于Tomcat的基本目录。*

server.tomcat.accesslog.enabled = false *＃启用访问日志。*

server.tomcat.accesslog.file最新格式= .yyyy-MM-dd *＃放置在日志文件名中的日期格式。*

server.tomcat.accesslog.pattern = common *＃访问日志的格式模式。*

server.tomcat.accesslog.prefix = access\_log *＃记录文件名前缀。*

server.tomcat.accesslog.rename-on-rotate = false *＃是否推迟在文件名中包含日期标记，直到旋转时间。*

server.tomcat.accesslog.request-attributes-enabled = false *＃为请求使用的IP地址，主机名，协议和端口设置请求属性。*

server.tomcat.accesslog.rotate = true *＃是否启用访问日志循环。*

server.tomcat.accesslog.suffix = .log*＃日志文件名后缀。*

server.tomcat.additional-tld-skip-patterns = *＃与TLD扫描相匹配的要匹配的瓶子的逗号分隔列表。*

server.tomcat.background-processor-delay = 30s *＃调用backgroundProcess方法之间的延迟。如果未指定持续时间后缀，则将使用秒。*

server.tomcat.basedir = *＃Tomcat基本目录。如果未指定，则使用临时目录。*

server.tomcat.internal-proxies = 10 \\。\\ d {1,3} \\。\\ d {1,3} \\。\\ d {1,3} | \\

。192 \\ 168 \\ d {1,3} \\ d {1,3} | \\

。169 \\ 254 \\ d {1,3} \\ d {1,3} | \\

。127 \\ d {1,3} \\ d {1,3} \\ d {1,3} | \\

172 \\ 1 [6-9] {1} \\ d {1,3} \\ d {1,3} |。。\\

172 \\ 2 [0-9] {1} \\ d {1,3} \\ d {1,3} |。。\\

172 \\。3 [0-1] {1} \\。\\ d {1,3} \\。\\ d {1,3} *＃匹配可信IP地址的正则表达式。*

server.tomcat.max-connections = 0 *＃服务器在任何给定时间接受和处理的最大连接数。*

server.tomcat.max-http-header-size = 0 *＃HTTP消息头的最大大小（以字节为单位）。*

server.tomcat.max-http-post-size = 0 *＃HTTP邮件内容的最大大小（以字节为单位）。*

server.tomcat.max-threads = 0 *＃工作线程的最大数量。*

server.tomcat.min-spare-threads = 0 *＃工作线程的最小数量。*

server.tomcat.port-header = X-Forwarded-Port*＃用于覆盖原始端口值的HTTP标头的名称。*

server.tomcat.protocol-header = *＃保存传入协议的头部，通常名为“X-Forwarded-Proto”。*

server.tomcat.protocol-header-https-value = https *＃协议头的值，指示传入请求是否使用SSL。*

server.tomcat.redirect-context-root = *＃是否应通过将/附加到路径来重定向对上下文根的请求。*

server.tomcat.remote-ip-header = *＃从中提取远程IP的HTTP头的名称。例如，“X-FORWARDED-FOR”。*

server.tomcat.resource.cache-ttl = *＃静态资源缓存的生存时间。*

server.tomcat.uri-encoding = UTF-8 *＃用于解码URI的字符编码。*

server.tomcat.use-relative-redirects = *＃对于sendRedirect调用生成的HTTP 1.1和更高版本位置标头是否使用相对或绝对重定向。*

server.undertow.accesslog.dir = *＃*

取消*访问日志目录。*server.undertow.accesslog.enabled = false *＃是否启用访问日志。*

server.undertow.accesslog.pattern = common *＃访问日志的格式模式。*

server.undertow.accesslog.prefix = access\_log。*＃日志文件名称前缀。*

server.undertow.accesslog.rotate = true*＃是否启用访问日志轮换。*

server.undertow.accesslog.suffix = log *＃日志文件名后缀。*

server.undertow.buffer-size = *＃每个缓冲区的大小，以字节为单位。*

server.undertow.direct-buffers = *＃是否在Java堆外分配缓冲区。*

server.undertow.io-threads = *＃为worker创建的I / O线程数量。*

server.undertow.eager-filter-init = true *＃是否应该在启动时初始化servlet过滤器。*

server.undertow.max-http-post-size = 0 *＃HTTP邮件内容的最大大小（以字节为单位）。*

server.undertow.worker-threads = *＃工作线程数。*

*#FREEMARKER（*[FreeMarkerProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerProperties.java)）

spring.freemarker.allow-request-override = false *＃是否允许HttpServletRequest属性覆盖（隐藏）具有相同名称的控制器生成的模型属性。*

spring.freemarker.allow-session-override = false *＃是否允许HttpSession属性覆盖（隐藏）具有相同名称的控制器生成的模型属性。*

spring.freemarker.cache = false *＃是否启用模板缓存。*

spring.freemarker.charset = UTF-8 *＃模板编码。*

spring.freemarker.check-template-location = true *＃是否检查模板位置是否存在。*

spring.freemarker.content-type = text / html *＃Content-Type值。*

spring.freemarker.enabled = true *＃是否为此技术启用MVC视图分辨率。*

spring.freemarker.expose-request-attributes = false *＃在与模板合并之前是否应将所有请求属性添加到模型中。*

spring.freemarker.expose-session-attributes = false *＃是否应该在与模板合并之前将所有HttpSession属性添加到模型中。*

spring.freemarker.expose-spring-macro-helpers = true *＃是否公开名为“springMacroRequestContext”的Spring的宏库使用的RequestContext。*

spring.freemarker.prefer-file-system-access = true *＃是否喜欢文件系统访问模板加载。文件系统访问使模板更改的热检测成为可能。*

spring.freemarker.prefix = *＃构建URL时预先查看名称的前缀。*

spring.freemarker.request-context-attribute = *＃所有视图的*

RequestContext属性的*名称。*spring.freemarker.settings.\* = *＃众所周知的FreeMarker键被传递给FreeMarker的配置。*

spring.freemarker.suffix = .ftl *＃在构建URL时附加到查看名称的后缀。*

spring.freemarker.template-loader-path = classpath：/ templates /*＃逗号分隔的模板路径列表。*

spring.freemarker.view-names = *＃可以解析的视图名称的白名单。*

[＃GROOVY](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateProperties.java) *TEMPLATES（*[GroovyTemplateProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateProperties.java)）

spring.groovy.template.allow-request-override = false *＃是否允许HttpServletRequest属性覆盖（隐藏）具有相同名称的控制器生成的模型属性。*

spring.groovy.template.allow-session-override = false *＃是否允许HttpSession属性覆盖（隐藏）具有相同名称的控制器生成的模型属性。*

spring.groovy.template.cache = false *＃是否启用模板缓存。*

spring.groovy.template.charset = UTF-8 *＃模板编码。*

spring.groovy.template.check-template-location = true*＃是否检查模板位置是否存在。*

spring.groovy.template.configuration.\* = *＃请参阅*[GroovyMarkupConfigurer](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/view/groovy/GroovyMarkupConfigurer.html)

spring.groovy.template.content-type = text / html *＃Content-Type值。*

spring.groovy.template.enabled = true *＃是否为此技术启用MVC视图分辨率。*

spring.groovy.template.expose-request-attributes = false *＃在与模板合并之前是否应将所有请求属性添加到模型中。*

spring.groovy.template.expose-session-attributes = false *＃是否应该在与模板合并之前将所有HttpSession属性添加到模型中。*

spring.groovy.template.expose-spring-macro-helpers = true *＃是否公开名为“springMacroRequestContext”的Spring的宏库使用的RequestContext。*

spring.groovy.template.prefix = *＃构建URL时预先查看名称的前缀。*

spring.groovy.template.request-context-attribute = *＃所有视图的*

RequestContext属性的*名称。*spring.groovy.template.resource-loader-path = classpath：/ templates / *＃模板路径。*

spring.groovy.template.suffix = .tpl *＃在构建URL时被附加到视图名称后缀。*

spring.groovy.template.view-names =*＃可以解析的视图名称的白名单。*

*＃SPRING HATEOAS（*[HateoasProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hateoas/HateoasProperties.java)）

spring.hateoas.use-hal-as-default-json-media-type = true *＃应用程序/ hal + json响应是否应发送到接受application / json的请求。*

＃HTTP *消息转换*spring.http.converters.preferred-json-mapper = *＃用于HTTP消息转换的首选JSON映射器。默认情况下，根据环境自动检测。*

[＃HTTP](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/http/HttpEncodingProperties.java) *编码（*[HttpEncodingProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/http/HttpEncodingProperties.java)）

spring.http.encoding.charset = UTF-8 *＃HTTP请求和响应的字符集。如果未明确设置，则添加到“Content-Type”标题中。*

spring.http.encoding.enabled = true *＃是否启用http编码支持。*

spring.http.encoding.force = *＃是否强制对HTTP请求和响应的配置字符集进行编码。*

spring.http.encoding.force-request = *＃是否强制编码到HTTP请求上配置的字符集。未指定“强制”时默认为true。*

spring.http.encoding.force-response =*＃是否强制编码到HTTP响应上配置的字符集。*

spring.http.encoding.mapping = *＃映射的编码区域。*

＃MULTIPART *（*[MultipartProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/MultipartProperties.java)）

spring.servlet.multipart.enabled = true *＃是否启用对分段上传的支持。*

spring.servlet.multipart.file-size-threshold = 0 *＃文件写入磁盘后的阈值。值可以使用后缀“MB”或“KB”分别表示兆字节或千字节。*

spring.servlet.multipart.location = *＃上传文件的中间位置。*

spring.servlet.multipart.max-file-size = 1MB *＃最大文件大小。值可以使用后缀“MB”或“KB”分别表示兆字节或千字节。*

spring.servlet.multipart.max-request-size = 10MB*＃最大请求大小。值可以使用后缀“MB”或“KB”分别表示兆字节或千字节。*

spring.servlet.multipart.resolve-lazily = false *＃是否在文件或参数访问时懒惰地解析多部分请求。*

[＃JACKSON](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jackson/JacksonProperties.java) *（*[JacksonProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jackson/JacksonProperties.java)）

spring.jackson.date-format = *＃日期格式字符串或完全合格的日期格式类名称。例如，`yyyy-MM-dd HH：mm：ss`。*

spring.jackson.default-property-inclusion = *＃在序列化过程中控制*

属性的包含*。使用Jackson的JsonInclude.Include枚举中的一个值进行配置。*spring.jackson.deserialization。\* = *＃杰克逊开/关功能，影响Java对象反序列化的方式。*

spring.jackson.generator.\* = *＃生成器的Jackson开/关功能。*

spring.jackson.joda-date-time-format =*＃乔达日期时间格式字符串。如果未配置，如果使用格式字符串配置“date-format”作为后备。*

spring.jackson.locale = *＃用于格式化的区域设置。*

spring.jackson.mapper.\* = *＃杰克逊通用开/关功能。*

spring.jackson.parser.\* = *＃解析器的Jackson开/关功能。*

spring.jackson.property-naming-strategy = *＃Jackson的PropertyNamingStrategy上的常量之一。也可以是PropertyNamingStrategy子类的完全限定类名。*

spring.jackson.serialization.\* = *＃杰克逊开/关功能，影响Java对象序列化的方式。*

spring.jackson.time-zone =*＃格式化日期时使用的时区。例如“America / Los\_Angeles”或“GMT + 10”。*

*＃GSON（*[GsonProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/gson/GsonProperties.java)）

spring.gson.date-format = *＃在序列化Date对象时使用的格式。*

spring.gson.disable-html-escaping = *＃是否禁用HTML字符的转义，例如'<'，'>'等*

。spring.gson.disable-inner-class-serialization = *＃是否在排除内部类序列化。*

spring.gson.enable-complex-map-key-serialization = *＃是否启用复杂映射键的序列化（即非基元化）。*

spring.gson.exclude-fields-without-expose-annotation = *＃是否排除没有“Expose”注释的序列化或反序列化考虑的所有字段。*

spring.gson.field-naming-policy = *＃在序列化和反序列化过程中应该应用于对象字段的命名策略。*

spring.gson.generate-non-executable-json = *＃是否通过在输出前添加一些特殊文本来生成不可执行的JSON。*

spring.gson.lenient = *＃是否对分析不符合RFC 4627的JSON宽容*

。spring.gson.long-serialization-policy = *＃长和长类型的序列化策略。*

spring.gson.pretty-printing = *＃是否输出适合漂亮打印的页面的序列化JSON。*

spring.gson.serialize-nulls = *＃是否序列化空字段。*

[＃JERSEY](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jersey/JerseyProperties.java) *（*[JerseyProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jersey/JerseyProperties.java)）

spring.jersey.application-path = *＃作为应用程序的基本URI的路径。如果指定，则覆盖“@ApplicationPath”的值。*

spring.jersey.filter.order = 0 *＃Jersey过滤器链顺序。*

spring.jersey.init.\* = *＃通过servlet或过滤器传递给Jersey的初始化参数。*

spring.jersey.servlet.load-on-startup = -1 *＃加载泽西岛servlet的启动优先级。*

spring.jersey.type = servlet *＃Jersey集成类型。*

[#SPRING](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/LdapProperties.java) *LDAP（*[LdapProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/LdapProperties.java)）

spring.ldap.anonymous-read-only = false *＃只读操作是否应使用匿名环境。*

spring.ldap.base = *＃所有操作应从其发起的基本后缀。*

spring.ldap.base-environment.\* = *＃LDAP规范设置。*

spring.ldap.password = *＃登录服务器的密码。*

spring.ldap.urls = *＃服务器的LDAP URL。*

spring.ldap.username = *＃登录服务器的用户名。*

＃EMBEDDED *LDAP（*[EmbeddedLdapProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapProperties.java)）

spring.ldap.embedded.base-dn = *＃基本DN的列表。*

spring.ldap.embedded.credential.username = *＃嵌入式LDAP用户名。*

spring.ldap.embedded.credential.password = *＃嵌入式LDAP密码。*

spring.ldap.embedded.ldif = classpath：schema.ldif *＃Schema（LDIF）脚本资源引用。*

spring.ldap.embedded.port = 0 *＃嵌入式LDAP端口。*

spring.ldap.embedded.validation.enabled = true *＃是否启用LDAP模式验证。*

spring.ldap.embedded.validation.schema = *＃自定义模式的路径。*

#MUSTACHE *TEMPLATES（*[MustacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.java)）

spring.mustache.allow-request-override = false *＃是否允许HttpServletRequest属性覆盖（隐藏）同名控制器生成的模型属性。*

spring.mustache.allow-session-override = false *＃是否允许HttpSession属性覆盖（隐藏）控制器生成的具有相同名称的模型属性。*

spring.mustache.cache = false *＃是否启用模板缓存。*

spring.mustache.charset = UTF-8 *＃模板编码。*

spring.mustache.check-template-location = true *＃是否检查模板位置是否存在。*

spring.mustache.content-type = text / html *＃Content-Type值。*

spring.mustache.enabled = true *＃是否为此技术启用MVC视图分辨率。*

spring.mustache.expose-request-attributes = false *＃是否所有的请求属性都应该在与模板合并之前添加到模型中。*

spring.mustache.expose-session-attributes = false *＃是否应该在与模板合并之前将所有HttpSession属性添加到模型中。*

spring.mustache.expose-spring-macro-helpers = true *＃是否公开名为“springMacroRequestContext”的Spring的宏库使用的RequestContext。*

spring.mustache.prefix= classpath：/ templates / *＃应用于模板名称的前缀。*

spring.mustache.request-context-attribute = *＃所有视图的*

RequestContext属性的*名称。*spring.mustache.suffix = .mustache *＃适用于模板名称的后缀。*

spring.mustache.view-names = *＃可以解析的视图名称的白名单。*

[＃SPRING](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcProperties.java) *MVC（*[WebMvcProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcProperties.java)）

spring.mvc.async.request-timeout = *＃异步请求处理*

超时*前的时间量。*spring.mvc.contentnegotiation.favor-parameter = false *＃是否应该使用请求参数（默认为“format”）来确定请求的媒体类型。*

spring.mvc.contentnegotiation.favor-path-extension = false *＃是否应该使用URL路径中的路径扩展来确定所请求的媒体类型。*

spring.mvc.contentnegotiation.media-types。\* = *＃将文件扩展名映射到媒体类型以进行内容协商。例如，yml到text / yaml。*

spring.mvc.contentnegotiation.parameter-name =*＃查询“使用参数”时使用的参数名称。*

spring.mvc.date-format = *＃要使用的日期格式。例如，`dd / MM / yyyy`。*

spring.mvc.dispatch-trace-request = false *＃是否将TRACE请求分派给FrameworkServlet doService方法。*

spring.mvc.dispatch-options-request = true *＃是否将OPTIONS请求分派给FrameworkServlet doService方法。*

spring.mvc.favicon.enabled = true *＃是否启用favicon.ico的解析。*

spring.mvc.formcontent.putfilter.enabled = true *＃是否启用Spring的HttpPutFormContentFilter。*

spring.mvc.ignore-default-model-on-redirect = true*＃重定向场景中是否应该忽略“默认”模型的内容。*

spring.mvc.locale = *＃使用的语言环境。默认情况下，此语言环境由“Accept-Language”标题覆盖。*

spring.mvc.locale-resolver = accept-header *＃定义应如何解析区域设置。*

spring.mvc.log-resolved-exception = false *＃是否启用由“HandlerExceptionResolver”解决的异常的警告日志记录。*

spring.mvc.message-codes-resolver-format = *＃消息代码的格式化策略。例如，`PREFIX\_ERROR\_CODE`。*

spring.mvc.pathmatch.use-registered-suffix-pattern = false*＃后缀模式匹配是否仅适用于使用“spring.mvc.contentnegotiation.media-types。\*”注册的扩展名。*

spring.mvc.pathmatch.use-suffix-pattern = false *＃匹配模式到请求时是否使用后缀模式匹配（“。\*”）。*

spring.mvc.servlet.load-on-startup = -1 *＃加载调度程序servlet的启动优先级。*

spring.mvc.static-path-pattern = / \*\* *＃用于静态资源的路径模式。*

spring.mvc.throw-exception-if-no-handler-found = false *＃如果没有发现Handler处理请求，是否应抛出“NoHandlerFoundException”。*

spring.mvc.view.prefix = *＃Spring MVC视图前缀。*

spring.mvc.view.suffix = *＃Spring MVC视图后缀。*

*＃SPRING RESOURCES HANDLING（*[ResourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ResourceProperties.java)）

spring.resources.add-mappings = true *＃是否启用默认资源处理。*

spring.resources.cache.cachecontrol.cache-private = *＃指示响应消息是针对单个用户的，不能由共享缓存存储。*

spring.resources.cache.cachecontrol.cache-public = *＃指示任何缓存都可以存储响应。*

spring.resources.cache.cachecontrol.max-age = *＃如果未指定持续时间后缀，则应该缓存响应的最长时间（以秒为单位）。*

spring.resources.cache.cachecontrol.must-revalidate =*＃指示一旦它变得陈旧，缓存就不能使用该响应，而不必在服务器上重新验证它。*

spring.resources.cache.cachecontrol.no-cache = *＃指示只有在服务器重新验证后才能重新使用缓存的响应。*

spring.resources.cache.cachecontrol.no-store = *＃表示在任何情况下都不缓存响应。*

spring.resources.cache.cachecontrol.no-transform = *＃指示不应该转换响应内容的中介（缓存和其他）。*

spring.resources.cache.cachecontrol.proxy-revalidate = *＃与“must-revalidate”指令的含义相同，只是它不适用于私有缓存。*

spring.resources.cache.cachecontrol.s-max-age = *＃共享缓存响应应该被缓存的最大时间，如果没有指定持续时间后缀，则以秒为单位。*

spring.resources.cache.cachecontrol.stale-if-error = *＃遇到错误时可以使用响应的最长时间，如果没有指定持续时间后缀，则以秒为单位。*

spring.resources.cache.cachecontrol.stale-while-revalidate = *＃如果未指定持续时间后缀，则可以在响应失效后的最长响应时间（以秒为单位）。*

spring.resources.cache.period = *＃资源处理程序服务的资源的缓存期。如果未指定持续时间后缀，则将使用秒。*

spring.resources.chain.cache= true *＃是否在资源链中启用缓存。*

spring.resources.chain.enabled = *＃是否启用Spring资源处理链。默认情况下，除非至少有一个策略已启用，否则禁用。*

spring.resources.chain.gzipped = false *＃是否启用已解压缩资源的解析。*

spring.resources.chain.html-application-cache = false *＃是否启用HTML5应用程序缓存清单重写。*

spring.resources.chain.strategy.content.enabled = false *＃是否启用内容版本策略。*

spring.resources.chain.strategy.content.paths = / \*\**＃应用于内容版本策略的逗号分隔模式列表。*

spring.resources.chain.strategy.fixed.enabled = false *＃是否启用固定版本策略。*

spring.resources.chain.strategy.fixed.paths = / \*\* *＃用逗号分隔的模式列表应用于固定版本策略。*

spring.resources.chain.strategy.fixed.version = *＃用于固定版本策略的版本字符串。*

spring.resources.static-locations = classpath：/ META-INF / resources /，classpath：/ resources /，classpath：/ static /，classpath：/ public / *＃静态资源的位置。*

＃SPRING *SESSION（*[SessionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/SessionProperties.java)）

spring.session.store-type = *＃会话存储类型。*

spring.session.servlet.filter-order = -2147483598 *＃会话存储库过滤器顺序。*

spring.session.servlet.filter-dispatcher-types = async，error，request *＃会话存储库过滤器调度程序类型。*

*＃春季会话HAZELCAST（*[HazelcastSessionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/HazelcastSessionProperties.java)）

spring.session.hazelcast.flush-mode = on-save *＃会话刷新模式。*

spring.session.hazelcast.map-name = spring：session：sessions *＃用于存储会话的地图名称。*

[＃SPRING](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/JdbcSessionProperties.java) *SESSION JDBC（*[JdbcSessionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/JdbcSessionProperties.java)）

spring.session.jdbc.cleanup-cron = 0 \* \* \* \* \* *＃过期会话清理作业的Cron表达式。*

spring.session.jdbc.initialize-schema = embedded *＃数据库模式初始化模式。*

spring.session.jdbc.schema = classpath：org / springframework / session / jdbc / schema- @ @ platform @@ .sql *＃用于初始化数据库模式的SQL文件的路径。*

spring.session.jdbc.table-name = SPRING\_SESSION *＃用于存储会话的数据库表的名称。*

*＃SPRING SESSION MONGODB（*[MongoSessionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/MongoSessionProperties.java)）

spring.session.mongodb.collection-name = sessions *＃用于存储会话的集合名称。*

*＃SPRING SESSION REDIS（*[RedisSessionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/RedisSessionProperties.java)）

spring.session.redis.cleanup-cron = 0 \* \* \* \* \* *＃过期会话清理作业的Cron表达式。*

spring.session.redis.flush-mode = on-save *＃会话刷新模式。*

spring.session.redis.namespace = spring：session *＃用于存储会话的密钥的命名空间。*

*＃THYMELEAF（*[ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java)）

spring.thymeleaf.cache = true *＃是否启用模板缓存。*

spring.thymeleaf.check-template = true *＃是否在渲染之前检查模板是否存在。*

spring.thymeleaf.check-template-location = true *＃是否检查模板位置是否存在。*

spring.thymeleaf.enabled = true *＃是否为Web框架启用Thymeleaf视图分辨率。*

spring.thymeleaf.enable-spring-el-compiler = false *＃在SpringEL表达式中启用SpringEL编译器。*

spring.thymeleaf.encoding = UTF-8 *＃模板文件编码。*

spring.thymeleaf.excluded-view-names = *＃应该从解析中排除的逗号分隔的视图名称列表（允许的模式）。*

spring.thymeleaf.mode = HTML *＃应用于模板的模板模式。另请参阅Thymeleaf的TemplateMode枚举。*

spring.thymeleaf.prefix = classpath：/ templates / *＃构建URL时预先查看名称的前缀。*

spring.thymeleaf.reactive.chunked-mode-view-names = *＃逗号分隔的视图名称列表（允许的模式），当设置最大块大小时，应该是CHUNKED模式中唯一执行的视图名称列表。*

spring.thymeleaf.reactive.full-mode-view-names =*＃即使设置了最大块大小，也应该在FULL模式下执行逗号分隔的视图名称列表（允许的模式）。*

spring.thymeleaf.reactive.max-chunk-size = 0 *＃用于写入响应的数据缓冲区的最大大小（以字节为单位）。*

spring.thymeleaf.reactive.media-types = *＃视图技术支持的媒体类型。*

spring.thymeleaf.servlet.content-type = text / html *＃写入HTTP响应的Content-Type值。*

spring.thymeleaf.suffix = .html *＃在构建URL时附加到视图名称的后缀。*

spring.thymeleaf.template-resolver-order = *＃链中模板解析器的顺序。*

spring.thymeleaf.view-名= *＃可以解析的逗号分隔的视图名称列表（允许的模式）。*

*＃SPRING WEBFLUX（*[WebFluxProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/WebFluxProperties.java)）

spring.webflux.date-format = *＃要使用的日期格式。例如，`dd / MM / yyyy`。*

spring.webflux.static-path-pattern = / \*\* *＃用于静态资源的路径模式。*

#SPRING *WEB SERVICES（*[WebServicesProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/webservices/WebServicesProperties.java)）

spring.webservices.path = / services *＃作为服务基础URI的路径。*

spring.webservices.servlet.init = *＃传递给Spring Web Services的Servlet初始化参数。*

spring.webservices.servlet.load-on-startup = -1 *＃加载Spring Web Services servlet的启动优先级。*

spring.webservices.wsdl-locations = *＃以逗号分隔的WSDL位置以及随附的XSD将作为bean公开的位置列表。*

*＃----------------------------------------*

*＃SECURITY PROPERTIES*

*＃----- -----------------------------------*

*＃SECURITY（*[SecurityProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/SecurityProperties.java)）

spring.security.filter.order = -100 *＃安全过滤器链顺序。*

spring.security.filter.dispatcher-types = async，error，request *＃安全性筛选器链调度程序类型。*

spring.security.user.name = user *＃默认用户名。*

spring.security.user.password = *＃默认用户名的密码。*

spring.security.user.roles = *＃授予默认用户名的角色。*

*＃SECURITY OAUTH2客户端（*[OAuth2ClientProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/client/OAuth2ClientProperties.java)）

spring.security.oauth2.client.provider。\* = *＃OAuth提供程序详细信息。*

spring.security.oauth2.client.registration。\* = *＃OAuth客户端注册。*

*＃----------------------------------------*

*＃DATA PROPERTIES*

*＃----- -----------------------------------*

[＃FLYWAY](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayProperties.java) *（*[FlywayProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayProperties.java)）

spring.flyway.baseline-description = *＃*

spring.flyway.baseline-on-migrate = *＃*

spring.flyway.baseline-version = 1 *＃开始迁移的版本*

spring.flyway.check-location = true *＃是否检查是否存在迁移脚本位置。*

spring.flyway.clean-disabled = *＃*

spring.flyway.clean-on-validation-error = *＃*

spring.flyway.dry-run-output = *＃*

spring.flyway.enabled = true *＃是否启用flyway。*

spring.flyway.encoding = *＃*

spring.flyway.error-handlers = *＃*

spring.flyway.group = *＃*

spring.flyway.ignore-future-migrations = *＃*

spring.flyway.ignore-missing-migrations = *＃*

spring.flyway.init-sqls = *＃SQL语句在获得它之后立即执行初始化连接。*

spring.flyway.installed-by = *＃*

spring.flyway.locations = classpath：db / migration *＃迁移脚本的位置。*

spring.flyway.mixed = *＃*

spring.flyway.out-of-order = *＃*

spring.flyway.password =*＃要使用Flyway创建自己的DataSource的JDBC密码。*

spring.flyway.placeholder-prefix = *＃*

spring.flyway.placeholder-replacement = *＃*

spring.flyway.placeholder-suffix = *＃*

spring.flyway.placeholders.\* = *＃*

spring.flyway.repeatable-sql-migration-prefix = *＃*

spring .flyway.schemas = *＃要更新的模式*

spring.flyway.skip-default-callbacks = *＃*

spring.flyway.skip-default-resolvers = *＃*

spring.flyway.sql-migration-prefix = V *＃*

spring.flyway.sql-migration -separator =*＃*

spring.flyway.sql-migration-suffix = .sql *＃*

spring.flyway.sql-migration-suffixes = *＃*

spring.flyway.table = *＃*

spring.flyway.target = *＃*

spring.flyway.undo-sql-migration-prefix = *＃*

spring.flyway.url = *＃要迁移的数据库的JDBC URL。如果未设置，则使用主要配置的数据源。*

spring.flyway.user = *＃登录要迁移的数据库的用户。*

spring.flyway.validate-on-migrate = *＃*

*＃LIQUIBASE（*[LiquibaseProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseProperties.java)）

spring.liquibase.change-log=classpath:/db/changelog/db.changelog-master.yaml

＃*更改日志配置路径。*

spring.liquibase.check-change-log-location = true *＃是否检查更改日志位置是否存在。*

spring.liquibase.contexts = *＃使用的运行时上下文的逗号分隔列表。*

spring.liquibase.default-schema = *＃默认数据库模式。*

spring.liquibase.drop-first = false *＃是否首先删除数据库模式。*

spring.liquibase.enabled = true *＃是否启用Liquibase支持。*

spring.liquibase.labels =*＃使用的运行时标签的逗号分隔列表。*

spring.liquibase.parameters.\* = *＃更改日志参数。*

spring.liquibase.password = *＃登录要迁移的数据库的密码。*

spring.liquibase.rollback-file = *＃执行更新时写回滚SQL的文件。*

spring.liquibase.url = *＃要迁移的数据库的JDBC URL。如果未设置，则使用主要配置的数据源。*

spring.liquibase.user = *＃登录要迁移的数据库的用户。*

*＃COUCHBASE（*[CouchbaseProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/couchbase/CouchbaseProperties.java)）

spring.couchbase.bootstrap-hosts = *＃从中引导的Couchbase节点（主机或IP地址）。*

spring.couchbase.bucket.name = default *＃要连接的存储桶的名称。*

spring.couchbase.bucket.password = *＃桶的密码。*

spring.couchbase.env.endpoints.key-value = 1 *＃针对键/值服务的每个节点的套接字数量。*

spring.couchbase.env.endpoints.query = 1 *＃针对查询（N1QL）服务的每个节点的套接字数量。*

spring.couchbase.env.endpoints.view = 1 *＃针对视图服务的每个节点的套接字数量。*

spring.couchbase.env.ssl.enabled = *＃是否启用SSL支持。除非另有规定，否则如果提供“keyStore”，则自动启用。*

spring.couchbase.env.ssl.key-store = *＃持有证书的JVM密钥存储的路径。*

spring.couchbase.env.ssl.key-store-password = *＃用于访问密钥存储区的密码。*

spring.couchbase.env.timeouts.connect = 5000ms *＃桶连接超时。*

spring.couchbase.env.timeouts.key-value = 2500ms *＃在特定的按键超时上执行阻塞操作。*

spring.couchbase.env.timeouts.query = 7500ms *＃N1QL查询操作超时。*

spring.couchbase.env.timeouts.socket-connect = 1000ms *＃套接字连接超时。*

spring.couchbase.env.timeouts.view = 7500ms *＃定期和地理空间视图操作超时。*

＃DAO *（*[PersistenceExceptionTranslationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.java)）

spring.dao.exceptiontranslation.enabled = true *＃是否启用PersistenceExceptionTranslationPostProcessor。*

＃CASSANDRA *（*[CassandraProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cassandra/CassandraProperties.java)）

spring.data.cassandra.cluster-name = *＃Cassandra集群的名称。*

spring.data.cassandra.compression = none *＃Cassandra二进制协议支持的压缩。*

spring.data.cassandra.connect-timeout = *＃套接字选项：连接超时。*

spring.data.cassandra.consistency-level = *＃查询一致性级别。*

spring.data.cassandra.contact-points = localhost *＃集群节点地址。*

spring.data.cassandra.fetch-size = *＃查询默认获取大小。*

spring.data.cassandra.keyspace-name = *＃使用的Keyspace名称。*

spring.data.cassandra.load-balancing-policy = *＃负载均衡策略的类名称。*

spring.data.cassandra.port = *＃Cassandra服务器的端口。*

spring.data.cassandra.password = *＃登录服务器的密码。*

spring.data.cassandra.pool.heartbeat-interval = 30s *＃心跳间隔后，在空闲连接上发送消息以确保其仍处于活动状态。如果未指定持续时间后缀，则将使用秒。*

spring.data.cassandra.pool.idle-timeout = 120s *＃空闲连接被移除前的空闲超时。如果未指定持续时间后缀，则将使用秒。*

spring.data.cassandra.pool.max-queue-size = 256*＃如果没有连接可用，请求排队的最大请求数。*

spring.data.cassandra.pool.pool-timeout = 5000ms *＃尝试从主机池获取连接时的池超时。*

spring.data.cassandra.read-timeout = *＃套接字选项：读取超时。*

spring.data.cassandra.reconnection-policy = *＃重新连接策略类。*

spring.data.cassandra.repositories.type = auto *＃启用Cassandra存储库的类型。*

spring.data.cassandra.retry-policy = *＃重试策略的类名称。*

spring.data.cassandra.serial-consistency-level = *＃查询串行一致性级别。*

spring.data.cassandra.schema-action = none *＃在启动时采取的模式操作。*

spring.data.cassandra.ssl = false *＃启用SSL支持。*

spring.data.cassandra.username = *＃服务器的登录用户。*

*＃DATA COUCHBASE（*[CouchbaseDataProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataProperties.java)）

spring.data.couchbase.auto-index = false *＃自动创建视图和索引。*

spring.data.couchbase.consistency = read-your-own-writes *＃在生成的查询中默认应用的一致性。*

spring.data.couchbase.repositories.type = auto *＃启用的Couchbase存储库的类型。*

*＃ELASTICSEARCH（*[ElasticsearchProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchProperties.java)）

spring.data.elasticsearch.cluster-name = elasticsearch *＃Elasticsearch集群名称。*

spring.data.elasticsearch.cluster-nodes = *＃以逗号分隔的集群节点地址列表。*

spring.data.elasticsearch.properties.\* = *＃用于配置客户端的其他属性。*

spring.data.elasticsearch.repositories.enabled = true *＃是否启用Elasticsearch存储库。*

＃DATA *LDAP* spring.data.ldap.repositories.enabled = true *＃是否启用LDAP存储库。*

*＃MONGODB（*[MongoProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/MongoProperties.java)）

spring.data.mongodb.authentication-database = *＃认证数据库名称。*

spring.data.mongodb.database = *＃数据库名称。*

spring.data.mongodb.field -naming-strategy = *＃要使用的FieldNamingStrategy的完全限定名称。*

spring.data.mongodb.grid-fs-database = *＃GridFS数据库名称。*

spring.data.mongodb.host = *＃Mongo服务器主机。不能使用URI进行设置。*

spring.data.mongodb.password = *＃登录mongo服务器的密码。不能使用URI进行设置。*

spring.data.mongodb.port = *＃Mongo服务器端口。不能使用URI进行设置。*

spring.data.mongodb.repositories.type = auto *＃启用Mongo存储库的类型。*

spring.data.mongodb.uri = mongodb：// localhost / test *＃Mongo数据库URI。无法使用主机，端口和凭证进行设置。*

spring.data.mongodb.username = *＃mongo服务器的登录用户。不能使用URI进行设置。*

*＃DATA REDIS*

spring.data.redis.repositories.enabled = true *＃是否启用Redis存储库。*

*＃NEO4J（*[Neo4jProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jProperties.java)）

spring.data.neo4j.auto-index = none *＃自动索引模式。*

spring.data.neo4j.embedded.enabled = true *＃是否在嵌入式驱动程序可用时启用嵌入式模式。*

spring.data.neo4j.open-in-view = true *＃注册OpenSessionInViewInterceptor。将Neo4j会话绑定到线程，以完成请求的整个处理。*

spring.data.neo4j.password = *＃登录服务器的密码。*

spring.data.neo4j.repositories.enabled = true *＃是否启用Neo4j存储库。*

spring.data.neo4j.uri = *驱动程序使用的*＃URI *。自动检测默认。*

spring.data.neo4j.username = *＃服务器的登录用户。*

＃DATA *REST（*[RepositoryRestProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/rest/RepositoryRestProperties.java)）

spring.data.rest.base-path = *＃Spring Data REST用于公开资源库资源的基础路径。*

spring.data.rest.default-media-type = *＃当没有指定任何内容时，默认使用的内容类型。*

spring.data.rest.default-page-size = *＃*

页面的*默认大小。*spring.data.rest.detection-strategy = default *＃用于确定哪些存储库暴露的策略。*

spring.data.rest.enable-enum-translation = *＃是否通过Spring Data REST默认资源包启用枚举值转换。*

spring.data.rest.limit-param-name =*＃URL查询字符串参数的名称，指示一次返回多少个结果。*

spring.data.rest.max-page-size = *＃*

页面的*最大尺寸。*spring.data.rest.page-param-name = *＃指示要返回哪个页面的URL查询字符串参数的名称。*

spring.data.rest.return-body-on-create = *＃是否在创建实体后返回响应主体。*

spring.data.rest.return-body-on-update = *＃是否在更新实体后返回响应主体。*

spring.data.rest.sort-param-name = *＃URL查询字符串参数的名称，指示对结果进行排序的方向。*

[＃SOLR](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/solr/SolrProperties.java) *（*[SolrProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/solr/SolrProperties.java)）

spring.data.solr.host = http：//127.0.0.1：8983 / solr *＃Solr主机。如果设置了“zk-host”，则忽略。*

spring.data.solr.repositories.enabled = true *＃是否启用Solr存储库。*

spring.data.solr.zk-host = *＃HOST：PORT形式的*＃ZooKeeper *主机地址。*

*＃DATA WEB（*[SpringDataWebProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/web/SpringDataWebProperties.java)）

spring.data.web.pageable.default页大小 = 20 *＃缺省页大小。*

spring.data.web.pageable.max-page-size = 2000 *＃要接受的最大页面大小。*

spring.data.web.pageable.one-indexed-parameters = false *＃是否公开并假设基于1的页码索引。*

spring.data.web.pageable.page-parameter = page *＃页面索引参数名称。*

spring.data.web.pageable.prefix = *＃页面编号和页面大小参数前面的一般前缀。*

spring.data.web.pageable.qualifier-delimiter = \_*＃限定符与实际页码和大小属性之间使用的分隔符。*

spring.data.web.pageable.size-parameter = size *＃页面大小参数名称。*

spring.data.web.sort.sort-parameter = sort *＃排序参数名称。*

＃DATASOURCE *（*[DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java)＆[DataSourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceProperties.java)）

spring.datasource.continue-on-error = false *＃是否在初始化数据库时发生错误时停止。*

spring.datasource.data = *＃数据（DML）脚本资源引用。*

spring.datasource.data-username = *＃执行DML脚本的数据库*

*的用户名（如果不同）。*

spring.datasource.data-password = *＃执行DML脚本的数据库的密码（如果不同）。*

spring.datasource.dbcp2.\* = *＃Commons DBCP2特定设置*

spring.datasource.driver-class-name =*＃JDBC驱动程序的完全限定名称。默认情况下基于URL自动检测。*

spring.datasource.generate-unique-name = false *＃是否生成随机数据源名称。*

spring.datasource.hikari.\* = *＃Hikari特定设置*

spring.datasource.initialization-mode = embedded *＃使用可用的DDL和DML脚本初始化数据源。*

spring.datasource.jmx-enabled = false *＃是否启用JMX支持（如果由底层池提供）。*

spring.datasource.jndi-name = *＃数据源的JNDI位置。设置时会忽略类，网址，用户名和密码。*

spring.datasource.name =*＃数据源的名称。使用嵌入式数据库时，默认为“testdb”。*

spring.datasource.password = *＃登录数据库的密码。*

spring.datasource.platform = all *＃在DDL或DML脚本中使用的平台（例如schema - $ {platform} .sql或data - $ {platform} .sql）。*

spring.datasource.schema = *＃架构（DDL）脚本资源引用。*

spring.datasource.schema-username = *＃执行DDL脚本的数据库*

的用户名*（如果不同）。*spring.datasource.schema-password = *＃执行DDL脚本的数据库的密码（如果不同）。*

spring.datasource.separator =;*＃SQL初始化脚本中的语句分隔符。*

spring.datasource.sql-script-encoding = *＃SQL脚本编码。*

spring.datasource.tomcat.\* = *＃Tomcat数据源特定设置*

spring.datasource.type = *＃要使用的连接池实现的完全限定名称。默认情况下，它是从类路径中自动检测的。*

spring.datasource.url = *＃数据库的JDBC URL。*

spring.datasource.username = *＃登录数据库的用户名。*

spring.datasource.xa.data-source-class-name = *＃XA数据源完全限定名称。*

spring.datasource.xa.properties =*＃传递给XA数据源的属性。*

[＃JEST](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/elasticsearch/jest/JestProperties.java) *（Elasticsearch HTTP客户端）（*[JestProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/elasticsearch/jest/JestProperties.java)）

spring.elasticsearch.jest.connection-timeout = 3s *＃连接超时。*

spring.elasticsearch.jest.multi-threaded = true *＃是否启用来自多个执行线程的连接请求。*

spring.elasticsearch.jest.password = *＃登录密码。*

spring.elasticsearch.jest.proxy.host = *＃HTTP客户端应该使用的代理主机。*

spring.elasticsearch.jest.proxy.port = *＃HTTP客户端应该使用的代理端口。*

spring.elasticsearch.jest.read-timeout = 3s *＃读取超时。*

spring.elasticsearch.jest.uris = http://localhost:9200*＃要使用的Elasticsearch实例的逗号分隔列表。*

spring.elasticsearch.jest.username = *＃登录用户名。*

*＃H2 Web控制台（*[H2ConsoleProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/h2/H2ConsoleProperties.java)）

spring.h2.console.enabled = false *＃是否启用控制台。*

spring.h2.console.path =/h2-console *＃控制台可用的路径。*

spring.h2.console.settings.trace = false *＃是否启用跟踪输出。*

spring.h2.console.settings.web-allow-others = false *＃是否启用远程访问。*

*＃InfluxDB（*[InfluxDbProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/influx/InfluxDbProperties.java)）

spring.influx.password = *＃登录密码。*

spring.influx.url = *＃要连接的InfluxDB实例的URL。*

spring.influx.user = *＃登录用户。*

[＃JOOQ](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqProperties.java) *（*[JooqProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqProperties.java)）

spring.jooq.sql-dialect = *＃使用SQL方言。自动检测默认。*

[＃JDBC](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JdbcProperties.java) *（*[JdbcProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JdbcProperties.java)）

spring.jdbc.template.fetch-size = -1 *＃当需要更多行时，应从数据库中获取的行数。*

spring.jdbc.template.max-rows = -1 *＃最大行数。*

spring.jdbc.template.query-timeout = *＃查询超时。默认是使用JDBC驱动程序的默认配置。如果未指定持续时间后缀，则将使用秒。*

[＃JPA](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java) *（*[JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java)，[HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java)）

spring.data.jpa.repositories.enabled = true *＃是否启用JPA存储库。*

spring.jpa.database = *＃目标数据库进行操作，默认为自动检测。可以使用“databasePlatform”属性进行替代设置。*

spring.jpa.database-platform = *＃要运行的目标数据库的名称，默认为自动检测。也可以使用“数据库”枚举进行设置。*

spring.jpa.generate-ddl = false *＃是否在启动时初始化模式。*

spring.jpa.hibernate.ddl-auto =*＃DDL模式。这实际上是“hibernate.hbm2ddl.auto”属性的快捷方式。当使用嵌入式数据库并且没有检测到模式管理器时，默认为“创建 - 删除”。否则，默认为“无”。*

spring.jpa.hibernate.naming.implicit-strategy = *＃隐式命名策略的完全限定名称。*

spring.jpa.hibernate.naming.physical-strategy = *＃物理命名策略的完全限定名称。*

spring.jpa.hibernate.use-new-id-generator-mappings = *＃是否将Hibernate的新的IdentifierGenerator用于AUTO，TABLE和SEQUENCE。*

spring.jpa.mapping-resources = *＃映射资源（相当于persistence.xml中的“映射文件”条目）。*

弹簧。= true *＃注册OpenEntityManagerInViewInterceptor。将JPA EntityManager绑定到线程，以完成请求的整个处理。*

spring.jpa.properties.\* = *＃在JPA提供程序上设置的其他本机属性。*

spring.jpa.show-sql = false *＃是否启用SQL语句的日志记录。*

[＃JTA](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.java) *（*[JtaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.java)）

spring.jta.enabled = true *＃是否启用JTA支持。*

spring.jta.log-dir = *＃事务日志目录。*

spring.jta.transaction-manager-id = *＃事务管理器唯一标识符。*

*＃ATOMIKOS（*[AtomikosProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jta/atomikos/AtomikosProperties.java)）

spring.jta.atomikos.connectionfactory.borrow-connection-timeout = 30 *＃从池中借用连接超时，以秒为单位。*

spring.jta.atomikos.connectionfactory.ignore-session-transacted-flag = true *＃创建会话时是否忽略事务处理标志。*

spring.jta.atomikos.connectionfactory.local-transaction-mode = false *＃是否需要本地事务。*

spring.jta.atomikos.connectionfactory.maintenance-interval = 60 *＃池的维护线程运行之间的时间，以秒为单位。*

spring.jta.atomikos.connectionfactory.max-idle-time = 60*＃从池中清除连接之后的时间，以秒为单位。*

spring.jta.atomikos.connectionfactory.max-lifetime = 0 *＃以秒为单位的连接可以在被销毁前汇集的时间。0表示没有限制。*

spring.jta.atomikos.connectionfactory.max-pool-size = 1 *＃池的最大尺寸。*

spring.jta.atomikos.connectionfactory.min-pool-size = 1 *＃池的最小大小。*

spring.jta.atomikos.connectionfactory.reap-timeout = 0 *＃借用连接的收获超时（以秒为单位）。0表示没有限制。*

spring.jta.atomikos.connectionfactory.unique-resource-name = jmsConnectionFactory*＃恢复期间用于识别资源的唯一名称。*

spring.jta.atomikos.connectionfactory.xa-connection-factory-class-name = *＃供应商特定的XAConnectionFactory实现。*

spring.jta.atomikos.connectionfactory.xa-properties = *＃供应商特定的XA属性。*

spring.jta.atomikos.datasource.borrow-connection-timeout = 30 *＃用于从池中借用连接的超时，以秒为单位。*

spring.jta.atomikos.datasource.concurrent-connection-validation = *＃是否使用并发连接验证。*

spring.jta.atomikos.datasource.default-isolation-level = *＃池提供的连接的默认隔离级别。*

spring.jta.atomikos.datasource.login-timeout = *＃建立数据库连接*

的超时时间*，以秒为单位。*spring.jta.atomikos.datasource.maintenance-interval = 60 *＃池维护线程运行之间的时间，以秒为单位。*

spring.jta.atomikos.datasource.max-idle-time = 60 *＃从池中清除连接之后的时间，以秒为单位。*

spring.jta.atomikos.datasource.max-lifetime = 0 *＃以秒为单位的连接可以在被销毁前汇集的时间。0表示没有限制。*

spring.jta.atomikos.datasource.max-pool-size = 1 *＃池的最大尺寸。*

spring.jta.atomikos.datasource.min-pool-size = 1*＃池的最小尺寸。*

spring.jta.atomikos.datasource.reap-timeout = 0 *＃借用连接的收获超时（以秒为单位）。0表示没有限制。*

spring.jta.atomikos.datasource.test-query = *＃返回之前用于验证连接的SQL查询或语句。*

spring.jta.atomikos.datasource.unique-resource-name = dataSource *＃在恢复期间用于标识资源的唯一名称。*

spring.jta.atomikos.datasource.xa-data-source-class-name = *＃供应商特定的XAConnectionFactory实现。*

spring.jta.atomikos.datasource.xa-properties = *＃供应商特定的XA属性。*

spring.jta.atomikos.properties.allow-sub-transactions = true *＃指定是否允许子交易。*

spring.jta.atomikos.properties.checkpoint-interval = 500 *＃检查点之间的时间间隔，表示为两个检查点之间的日志写入次数。*

spring.jta.atomikos.properties.default-jta-timeout = 10000ms *＃JTA事务的默认超时。*

spring.jta.atomikos.properties.default-max-wait-time-on-shutdown = 9223372036854775807 *＃正常关机（无强制）等待事务完成多长时间。*

spring.jta.atomikos.properties.enable-logging = true *＃是否启用磁盘日志记录。*

spring.jta.atomikos.properties.force-shutdown-on-vm-exit = false *＃VM关闭是否应触发事务核心的强制关闭。*

spring.jta.atomikos.properties.log-base-dir = *＃应该存储日志文件的目录。*

spring.jta.atomikos.properties.log -base -name = tmlog *＃事务日志文件的基本名称。*

spring.jta.atomikos.properties.max-actives = 50 *＃活动事务的最大数量。*

spring.jta.atomikos.properties.max-timeout = 300000ms *＃交易允许的最大超时时间。*

spring.jta.atomikos.properties.recovery.delay = 10000ms *＃两次恢复扫描之间的延迟。*

spring.jta.atomikos.properties.recovery.forget- orphaned -log-entries-delay = 86400000ms *＃延迟后恢复可以清除挂起（'孤立'）日志条目。*

spring.jta.atomikos.properties.recovery.max-retries = 5 *＃抛出异常之前尝试提交事务的重试次数。*

spring.jta.atomikos.properties.recovery.retry-interval = 10000ms *＃重试尝试之间的延迟。*

spring.jta.atomikos.properties.serial-jta-transactions = true *＃是否应该在可能的情况下连接子事务。*

spring.jta.atomikos.properties.service = *＃应该启动的事务管理器实现。*

spring.jta.atomikos.properties.threaded-two-phase-commit = false *＃是否在参与资源上使用不同（并发）的线程进行两阶段提交。*

spring.jta.atomikos.properties.transaction-manager-unique-name = *＃事务管理器的唯一名称。*

*＃BITRONIX*

spring.jta.bitronix.connectionfactory.acquire-increment = 1 *＃增长池时创建的连接数。*

spring.jta.bitronix.connectionfactory.acquisition-interval = 1 *＃在获取无效连接后尝试重新获取连接之前，需要等待的时间（以秒为单位）。*

spring.jta.bitronix.connectionfactory.acquisition-timeout = 30 *＃以秒为单位的超时时间，用于从池中获取连接。*

spring.jta.bitronix.connectionfactory.allow-local-transactions = true *＃事务管理器是否应允许混合XA和非XA事务。*

spring.jta.bitronix.connectionfactory.apply-transaction-timeout = false*＃在注册时是否应该在XAResource上设置事务超时。*

spring.jta.bitronix.connectionfactory.automatic-enlisting-enabled = true *＃资源是否应该自动注册和退出。*

spring.jta.bitronix.connectionfactory.cache-producer-consumers = true *＃生产者和消费者是否应该被缓存。*

spring.jta.bitronix.connectionfactory.class-name = *＃XA资源的基础实现类名称。*

spring.jta.bitronix.connectionfactory.defer-connection-release = true *＃提供者是否可以在同一连接上运行多个事务并支持事务交叉。*

spring.jta.bitronix.connectionfactory.disabled= *＃该资源是否被禁用，意味着暂时禁止从其池中获取连接。*

spring.jta.bitronix.connectionfactory.driver-properties = *＃应该在底层实现上设置的属性。*

spring.jta.bitronix.connectionfactory.failed = *＃标记此资源生产者失败。*

spring.jta.bitronix.connectionfactory.ignore-recovery-failures = false *＃是否应该忽略恢复失败。*

spring.jta.bitronix.connectionfactory.max-idle-time = 60 *＃连接从池中清理之后的时间，以秒为单位。*

spring.jta.bitronix.connectionfactory.max-pool-size = 10*＃池的最大尺寸。0表示没有限制。*

spring.jta.bitronix.connectionfactory.min-pool-size = 0 *＃池的最小大小。*

spring.jta.bitronix.connectionfactory.password = *＃用于连接到JMS提供程序的密码。*

spring.jta.bitronix.connectionfactory.share-transaction-connections = false *＃ACCESSIBLE状态下的连接是否可以在事务上下文中共享。*

spring.jta.bitronix.connectionfactory.test-connections = true *＃连接是否需要从池中获取时进行测试。*

spring.jta.bitronix.connectionfactory.two-pc-ordering-position = 1*＃这个资源在两阶段提交期间应该采用的位置（总是首先是Integer.MIN\_VALUE，总是最后是Integer.MAX\_VALUE）。*

spring.jta.bitronix.connectionfactory.unique-name = jmsConnectionFactory *＃在恢复期间用于标识资源的唯一名称。*

spring.jta.bitronix.connectionfactory.use-tm-join = true *＃启动XAResources时是否应使用TMJOIN。*

spring.jta.bitronix.connectionfactory.user = *＃用于连接到JMS提供程序的用户。*

spring.jta.bitronix.datasource.acquire-increment = 1 *＃增长池时创建的连接数。*

spring.jta.bitronix.datasource.acquisition-interval = 1*＃在获取无效连接后，尝试重新获取连接之前，需要等待几秒钟。*

spring.jta.bitronix.datasource.acquisition-timeout = 30 *＃以秒为单位超时获取池中的连接。*

spring.jta.bitronix.datasource.allow-local-transactions = true *＃事务管理器是否应允许混合XA和非XA事务。*

spring.jta.bitronix.datasource.apply-transaction-timeout = false *＃在注册时是否应该在XAResource上设置事务超时。*

spring.jta.bitronix.datasource.automatic-enlisting-enabled = true *＃资源是否应该自动登录和退出。*

spring.jta.bitronix.datasource.class-name = *＃XA资源的基础实现类名称。*

spring.jta.bitronix.datasource.cursor-holdability = *＃连接的默认光标可保存性。*

spring.jta.bitronix.datasource.defer-connection-release = true *＃数据库是否可以在同一连接上运行多个事务并支持事务交叉。*

spring.jta.bitronix.datasource.disabled = *＃该资源是否被禁用，意味着暂时禁止从其池中获取连接。*

spring.jta.bitronix.datasource.driver-properties = *＃应该在底层实现中设置的属性。*

spring.jta.bitronix.datasource.enable -jdbc4-connection-test = *＃在从池中获取连接时是否调用Connection.isValid（）。*

spring.jta.bitronix.datasource.failed = *＃标记此资源生产者失败。*

spring.jta.bitronix.datasource.ignore-recovery-failures = false *＃是否应该忽略恢复失败。*

spring.jta.bitronix.datasource.isolation-level = *＃连接的默认隔离级别。*

spring.jta.bitronix.datasource.local-auto-commit = *＃本地事务的默认自动提交模式。*

spring.jta.bitronix.datasource.login-timeout =*＃建立数据库连接的超时时间，以秒为单位。*

spring.jta.bitronix.datasource.max-idle-time = 60 *＃从池中清除连接之后的时间，以秒为单位。*

spring.jta.bitronix.datasource.max-pool-size = 10 *＃池的最大尺寸。0表示没有限制。*

spring.jta.bitronix.datasource.min-pool-size = 0 *＃池的最小尺寸。*

spring.jta.bitronix.datasource.prepared-statement-cache-size = 0 *＃准备好的语句缓存的目标大小。0禁用缓存。*

spring.jta.bitronix.datasource.share-transaction-connections = false*＃ACCESSIBLE状态下的连接是否可以在事务上下文中共享。*

spring.jta.bitronix.datasource.test-query = *＃返回之前用于验证连接的SQL查询或语句。*

spring.jta.bitronix.datasource.two-pc-ordering-position = 1 *＃这个资源在两阶段提交期间应该采取的位置（总是首先是Integer.MIN\_VALUE，并且总是最后是Integer.MAX\_VALUE）。*

spring.jta.bitronix.datasource.unique-name = dataSource *＃在恢复期间用于标识资源的唯一名称。*

spring.jta.bitronix.datasource.use-tm-join = true *＃启动XAResources时是否应使用TMJOIN。*

spring.jta.bitronix.properties.allow-multiple-lrc = false *＃是否允许多个LRC资源被列入同一事务。*

spring.jta.bitronix.properties.asynchronous2-pc = false *＃是否启用两阶段*

落实的*异步执行。*spring.jta.bitronix.properties.background-recovery-interval-seconds = 60 *＃在后台运行恢复进程的间隔秒数。*

spring.jta.bitronix.properties.current-node-only-recovery = true *＃是否仅恢复当前节点。*

spring.jta.bitronix.properties.debug-zero-resource-transaction = false*＃是否记录创建并提交未执行单个登记资源的事务的调用堆栈。*

spring.jta.bitronix.properties.default-transaction-timeout = 60 *＃默认事务超时，以秒为单位。*

spring.jta.bitronix.properties.disable-jmx = false *＃是否启用JMX支持。*

spring.jta.bitronix.properties.exception-analyzer = *＃设置要使用的异常分析器实现的完全限定名称。*

spring.jta.bitronix.properties.filter-log-status = false *＃是否启用对日志的过滤，以便仅写入强制日志。*

spring.jta.bitronix.properties.force-batching-enabled = true*＃磁盘力量是否成批。*

spring.jta.bitronix.properties.forced-write-enabled = true *＃日志是否被强制为磁盘。*

spring.jta.bitronix.properties.graceful-shutdown-interval = 60 *＃TM在等待事务在关闭时中止之前完成的最大秒数。*

spring.jta.bitronix.properties.jndi-transaction-synchronization-registry-name = *＃TransactionSynchronizationRegistry的JNDI名称。*

spring.jta.bitronix.properties.jndi-user-transaction-name = *＃UserTransaction的JNDI名称。*

spring.jta.bitronix.properties.journal = disk *＃日志的名称。可以是'磁盘'，'空'或类名。*

spring.jta.bitronix.properties.log-part1-filename = btm1.tlog *＃日志的第一个片段的名称。*

spring.jta.bitronix.properties.log-part2-filename = btm2.tlog *＃日志的第二个片段的名称。*

spring.jta.bitronix.properties.max-log-size-in-mb = 2 *＃日志片段的最大大小（以兆字节为单位）。*

spring.jta.bitronix.properties.resource-configuration-filename = *＃ResourceLoader配置文件名。*

spring.jta.bitronix.properties.server-id = *＃必须唯一标识此TM实例的ASCII ID。默认为机器的IP地址。*

spring.jta.bitronix.properties.skip-corrupted-logs = false*＃跳过损坏的事务日志条目。*

spring.jta.bitronix.properties.warn-about-zero-resource-transaction = true *＃是否为未执行单个登记资源而执行的事务记录警告。*

*＃NARAYANA（*[NarayanaProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jta/narayana/NarayanaProperties.java)）

spring.jta.narayana.default-timeout = 60s *＃交易超时。如果未指定持续时间后缀，则将使用秒。*

spring.jta.narayana.expiry-scanners =

com.arjuna.ats.internal.arjuna.recovery.ExpiredTransactionStatusManagerScanner *＃过期扫描仪的逗号分隔列表。*

spring.jta.narayana.log-dir = *＃交易对象存储目录。*

spring.jta.narayana.one-phase-commit = true *＃是否启用一个阶段提交优化。*

spring.jta.narayana.periodic-recovery-period = 120s*＃执行周期性恢复扫描的时间间隔。如果未指定持续时间后缀，则将使用秒。*

spring.jta.narayana.recovery-backoff-period = 10s *＃恢复扫描的第一阶段和第二阶段之间*

的退避*阶段。如果未指定持续时间后缀，则将使用秒。*spring.jta.narayana.recovery-db-pass = *＃恢复管理器要使用的数据库密码。*

spring.jta.narayana.recovery-db-user = *＃恢复管理器使用的数据库用户名。*

spring.jta.narayana.recovery-jms-pass = *＃恢复管理器要使用的JMS密码。*

spring.jta.narayana.recovery-jms-user =*＃恢复管理器使用的JMS用户名。*

spring.jta.narayana.recovery-modules = *＃以逗号分隔的恢复模块列表。*

spring.jta.narayana.transaction-manager-id = 1 *＃唯一的事务管理器ID。*

spring.jta.narayana.xa-resource-orphan-filters = *＃孤立过滤器的逗号分隔列表。*

*＃EMBEDDED MONGODB（*[EmbeddedMongoProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoProperties.java)）

spring.mongodb.embedded.features = sync\_delay *＃要启用的功能的逗号分隔列表。*

spring.mongodb.embedded.storage.database-dir = *＃用于数据存储的目录。*

spring.mongodb.embedded.storage.oplog-size = *＃oplog的最大大小，以兆字节为单位。*

spring.mongodb.embedded.storage.repl-set-name = *＃副本集的名称。*

spring.mongodb.embedded.version = 3.2.2 *＃使用Mongo版本。*

*＃REDIS（*[RedisProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisProperties.java)）

[spring.redis.cluster.max](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisProperties.java) -redirects = *＃在群集中执行命令时要遵循的最大重定向数。*

spring.redis.cluster.nodes = *＃以逗号分隔的“主机：端口”对列表进行引导。*

spring.redis.database = 0 *＃连接工厂使用的数据库索引。*

spring.redis.url = *＃连接网址。覆盖主机，端口和密码。用户被忽略。示例：redis：// user：password@example.com ：6379*

spring.redis.host = localhost *＃Redis服务器主机。*

spring.redis.jedis.pool.max-active = 8*＃在给定时间池可以分配的最大连接数。使用负值无限制。*

spring.redis.jedis.pool.max-idle = 8 *＃池中“空闲”连接的最大数量。使用负值表示无限数量的空闲连接。*

spring.redis.jedis.pool.max-wait = -1ms *＃当池被耗尽时抛出异常之前连接分配应该阻塞的最大时间量。使用负值可以无限期地阻止。*

spring.redis.jedis.pool.min-idle = 0 *＃目标为保持在池中的最小空闲连接数。如果该设置是肯定的，则该设置仅起作用。*

spring.redis.lettuce.pool.max-active = 8*＃在给定时间池可以分配的最大连接数。使用负值无限制。*

spring.redis.lettuce.pool.max-idle = 8 *＃池中“空闲”连接的最大数量。使用负值表示无限数量的空闲连接。*

spring.redis.lettuce.pool.max-wait = -1ms *＃连接分配在池耗尽时抛出异常之前应阻塞的最长时间量。使用负值可以无限期地阻止。*

spring.redis.lettuce.pool.min-idle = 0 *＃目标为要在池中维护的最小空闲连接数。如果该设置是肯定的，则该设置仅起作用。*

spring.redis.lettuce.shutdown-timeout = 100ms*＃关机超时。*

spring.redis.password = *＃登录redis服务器的密码。*

spring.redis.port = 6379 *＃Redis服务器端口。*

spring.redis.sentinel.master = *＃Redis服务器的名称。*

spring.redis.sentinel.nodes = *＃“主机：端口”对的逗号分隔列表。*

spring.redis.ssl = false *＃是否启用SSL支持。*

spring.redis.timeout = *＃连接超时。*

＃TRANSACTION *（*[TransactionProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/TransactionProperties.java)）

spring.transaction.default-timeout = *＃默认事务超时。如果未指定持续时间后缀，则将使用秒。*

spring.transaction.rollback-on-commit-failure = *＃是否回滚提交失败。*

*＃----------------------------------------*

*＃INTEGRATION PROPERTIES*

*＃----- -----------------------------------*

*＃ACTIVEMQ（*[ActiveMQProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQProperties.java)）

spring.activemq.broker-url = *＃ActiveMQ代理的URL。自动生成默认。*

spring.activemq.close-timeout = 15s *＃在考虑完成之前等待的时间。*

spring.activemq.in-memory = true *＃默认代理URL是否应该在内存中。如果指定了明确的代理，则忽略。*

spring.activemq.non-blocking-redelivery = false *＃是否在重新传递回退事务中的消息之前停止消息传递。这意味着启用此功能时不会保留消息顺序。*

spring.activemq.password = *＃登录经纪人的密码。*

spring.activemq.send-timeout = 0ms *＃等待响应消息发送的时间。将其设置为0以永久等待。*

spring.activemq.user = *＃代理的登录用户。*

spring.activemq.packages.trust-all = *＃是否信任所有包。*

spring.activemq.packages.trusted = *＃以逗号分隔的特定软件包列表（不信任所有软件包）。*

spring.activemq.pool.block-if-full = true *＃是否阻止请求连接并且池已满。将其设置为false以代替引发“JMSException”。*

spring.activemq.pool.block-if-full-timeout = -1ms*＃如果池仍然已满，则在抛出异常之前阻塞期。*

spring.activemq.pool.create-connection-on-startup = true *＃是否在启动时创建连接。可用于在启动时预热池。*

spring.activemq.pool.enabled = false *＃是否应该创建PooledConnectionFactory，而不是常规的ConnectionFactory。*

spring.activemq.pool.expiry-timeout = 0ms *＃连接到期超时。*

spring.activemq.pool.idle-timeout = 30s *＃连接空闲超时。*

spring.activemq.pool.max-connections = 1 *＃*

共享连接的*最大数量。*spring.activemq.pool.maximum-active-session-per-connection = 500*＃每个连接的最大活动会话数。*

spring.activemq.pool.reconnect-on-exception = true *＃发生“JMSException”时重置连接。*

spring.activemq.pool.time-between-expiration-check = -1ms *＃空闲连接驱逐线程运行之间的休眠时间。否定时，不会有空闲连接逐出线程运行。*

spring.activemq.pool.use-anonymous-producers = true *＃是否只使用一个匿名的“MessageProducer”实例。将其设置为false以在每次需要时创建一个“MessageProducer”。*

[＃ARTEMIS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisProperties.java) *（*[ArtemisProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisProperties.java)）

spring.artemis.embedded.cluster-password = *＃集群密码。默认情况下在启动时随机生成。*

spring.artemis.embedded.data-directory = *＃日记文件目录。如果关闭持久性，则不需要。*

spring.artemis.embedded.enabled = true *＃是否在Artemis服务器API可用时启用嵌入模式。*

spring.artemis.embedded.persistent = false *＃是否启用持久存储。*

spring.artemis.embedded.queues = *＃在启动时创建的逗号分隔列表。*

spring.artemis.embedded.server-id =*＃服务器ID。默认情况下，使用自动递增的计数器。*

spring.artemis.embedded.topics = *＃启动时要创建的主题的逗号分隔列表。*

spring.artemis.host = localhost *＃阿蒂米斯经纪人主机。*

spring.artemis.mode = *＃Artemis部署模式，默认为自动检测。*

spring.artemis.password = *＃代理的登录密码。*

spring.artemis.port = 61616 *＃阿蒂米斯经纪人港口。*

spring.artemis.user = *＃代理的登录用户。*

＃SPRING *BATCH（*[BatchProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchProperties.java)）

spring.batch.initialize-schema = embedded *＃数据库模式初始化模式。*

spring.batch.job.enabled = true *＃启动时执行上下文中的所有Spring批处理作业。*

spring.batch.job.names = *＃逗号分隔的启动时要执行的作业名称列表（例如`job1，job2`）。默认情况下，执行在上下文中找到的所有作业。*

spring.batch.schema = classpath：org / springframework / batch / core / schema- @ @ platform @@ .sql *＃用于初始化数据库模式的SQL文件的路径。*

spring.batch.table-prefix =*＃所有批量元数据表的表格前缀。*

*#SPRING INTEGRATION（*[IntegrationProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationProperties.java)）

spring.integration.jdbc.initialize-schema = embedded *＃数据库模式初始化模式。*

spring.integration.jdbc.schema = classpath中：组织/ springframework的/集成/ JDBC / schema- @ @ 平台@ @ .SQL *＃的路径SQL文件，以用于初始化数据库架构。*

[＃JMS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JmsProperties.java) *（*[JmsProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JmsProperties.java)）

spring.jms.jndi-name = *＃连接工厂的JNDI名称。设置时，优先于其他连接工厂自动配置。*

spring.jms.listener.acknowledge-mode = *＃容器的确认模式。默认情况下，侦听器通过自动确认进行事务处理。*

spring.jms.listener.auto-startup = true *＃启动时自动启动容器。*

spring.jms.listener.concurrency = *＃最小并发消费者数量。*

spring.jms.listener.max-concurrency = *＃最大并发消费者数量。*

spring.jms.pub-sub-domain = false*＃默认目标类型是否为主题。*

spring.jms.template.default-destination = *＃在没有目标参数的发送和接收操作上使用的默认目标。*

spring.jms.template.delivery-delay = *＃发送延迟用于发送呼叫。*

spring.jms.template.delivery-mode = *＃传送模式。设置时启用QoS（服务质量）。*

spring.jms.template.priority = *＃发送时的消息优先级。设置时启用QoS（服务质量）。*

spring.jms.template.qos-enabled = *＃发送消息时是否启用显式QoS（服务质量）。*

spring.jms.template.receive-timeout =*＃超时使用接收电话。*

spring.jms.template.time-to-live = *＃发送消息时的生存时间。设置时启用QoS（服务质量）。*

*＃APACHE KAFKA（*[KafkaProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaProperties.java)）

spring.kafka.admin.client-id = *＃发送请求时传递给服务器的ID。用于服务器端日志记录。*

spring.kafka.admin.fail-fast = false *＃如果代理在启动时不可用，是否快速失败。*

spring.kafka.admin.properties。\* = *＃用于配置客户端的其他特定于管理员的属性。*

spring.kafka.admin.ssl.key-password = *＃密钥存储文件中的私钥密码。*

spring.kafka.admin.ssl.keystore-location = *＃密钥存储文件的位置。*

spring.kafka.admin.ssl.keystore-password =*＃存储密钥存储文件的密码。*

spring.kafka.admin.ssl.truststore-location = *＃信任存储文件的位置。*

spring.kafka.admin.ssl.truststore-password = *＃存储信任存储文件的密码。*

spring.kafka.bootstrap-servers = *＃主机：端口对的逗号分隔列表，用于建立到Kafka集群的初始连接。*

spring.kafka.client-id = *＃发送请求时传递给服务器的ID。用于服务器端日志记录。*

spring.kafka.consumer.auto-commit-interval = *＃如果'enable.auto.commit'设置为true，则消费者偏移自动提交给Kafka的频率。*

spring.kafka.consumer.auto-offset-reset = *＃在Kafka中没有初始偏移量或当前偏移量不再存在于服务器上时该做什么。*

spring.kafka.consumer.bootstrap-servers = *＃主机：端口对的逗号分隔列表，用于建立与Kafka集群的初始连接。*

spring.kafka.consumer.client-id = *＃在发出请求时传递给服务器的ID。用于服务器端日志记录。*

spring.kafka.consumer.enable-auto-commit = *＃用户的偏移是否在后台定期提交。*

spring.kafka.consumer.fetch-max-wait =*＃如果没有足够的数据立即满足“fetch.min.bytes”给出的要求，服务器在应答提取请求之前阻塞的最大时间量。*

spring.kafka.consumer.fetch-min-size = *＃服务器为获取请求返回的最小数据量（以字节为单位）。*

spring.kafka.consumer.group-id = *＃标识此用户所属的用户组的唯一字符串。*

spring.kafka.consumer.heartbeat-interval = *＃心跳到消费者协调员之间的预期时间。*

spring.kafka.consumer.key-deserializer = *＃键*

的反序列*化程序类。*spring.kafka.consumer.max-poll-records =*＃在一次调用poll（）中返回的最大记录数。*

spring.kafka.consumer.properties。\* = *＃用于配置客户端的其他消费者特定属性。*

spring.kafka.consumer.ssl.key-password = *＃密钥存储文件中的私钥密码。*

spring.kafka.consumer.ssl.keystore-location = *＃密钥存储文件的位置。*

spring.kafka.consumer.ssl.keystore-password = *＃存储密钥存储文件的密码。*

spring.kafka.consumer.ssl.truststore-location = *＃信任存储文件的位置。*

spring.kafka.consumer.ssl.truststore-password = *＃存储信任存储文件的密码。*

spring.kafka.consumer.value-deserializer = *＃*

解析器*类的值。*spring.kafka.jaas.control-flag = required *＃登录配置的控制标志。*

spring.kafka.jaas.enabled = false *＃是否启用JAAS配置。*

spring.kafka.jaas.login-module = com.sun.security.auth.module.Krb5LoginModule *＃登录模块。*

spring.kafka.jaas.options = *＃其他JAAS选项。*

spring.kafka.listener.ack-count = *＃当ackMode为“COUNT”或“COUNT\_TIME”时，偏移量之间的记录数。*

spring.kafka.listener.ack-mode = *＃Listener AckMode。请参阅spring-kafka文档。*

spring.kafka.listener.ack-time = *＃当ackMode为“TIME”或“COUNT\_TIME”时，偏移提交之间的时间。*

spring.kafka.listener.client-id = *＃监听器的消费者client.id属性的前缀。*

spring.kafka.listener.concurrency = *＃在侦听器容器中运行的线程数。*

spring.kafka.listener.idle-event-interval = *＃发布空闲消费者事件（未收到数据）之间的时间。*

spring.kafka.listener.log-container-config = *＃是否在初始化期间记录容器配置（INFO级别）。*

spring.kafka.listener.monitor-interval =*＃检查无响应客户的时间。如果未指定持续时间后缀，则将使用秒。*

spring.kafka.listener.no-poll-threshold = *＃乘数应用于“pollTimeout”以确定消费者是否无响应。*

spring.kafka.listener.poll-timeout = *＃轮询消费者时使用的超时。*

spring.kafka.listener.type = single *＃监听器类型。*

spring.kafka.producer.acks = *＃生产者在考虑请求完成之前要求领导者收到的确认数量。*

spring.kafka.producer.batch-size = *＃发送前批量记录的数量。*

spring.kafka.producer.bootstrap的服务器= *＃主机：端口对的逗号分隔列表，用于建立到Kafka集群的初始连接。*

spring.kafka.producer.buffer-memory = *＃生产者可用于缓冲等待发送到服务器的记录的总内存字节数。*

spring.kafka.producer.client-id = *＃在发出请求时传递给服务器的ID。用于服务器端日志记录。*

spring.kafka.producer.compression-type = *＃生产者生成的所有数据的压缩类型。*

spring.kafka.producer.key-serializer = *＃键的序列化类。*

spring.kafka.producer.properties.\* = *＃用于配置客户端的其他特定于生产者的属性。*

spring.kafka.producer.retries = *＃当大于零时，允许重试失败的发送。*

spring.kafka.producer.ssl.key-password = *＃密钥存储文件中的私钥密码。*

spring.kafka.producer.ssl.keystore-location = *＃密钥存储文件的位置。*

spring.kafka.producer.ssl.keystore-password = *＃存储密钥存储文件的密码。*

spring.kafka.producer.ssl.truststore-location = *＃信任存储文件的位置。*

spring.kafka.producer.ssl.truststore-password = *＃存储信任存储文件的密码。*

spring.kafka.producer.transaction-id-prefix =*＃非空时，为生产者启用事务支持。*

spring.kafka.producer.value-serializer = *＃值的串行器类。*

spring.kafka.properties.\* = *＃用于配置客户端的其他属性，通常用于生产者和使用者。*

spring.kafka.ssl.key-password = *＃密钥存储文件中的私钥密码。*

spring.kafka.ssl.keystore-location = *＃密钥存储文件的位置。*

spring.kafka.ssl.keystore-password = *＃存储密钥存储文件的密码。*

spring.kafka.ssl.truststore-location = *＃信任存储文件的位置。*

spring.kafka.ssl.truststore-password =*＃存储信任存储文件的密码。*

spring.kafka.template.default-topic = *＃发送消息的默认主题。*

*＃RABBIT（*[RabbitProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitProperties.java)）

spring.rabbitmq.addresses = *＃客户端应该连接的地址的逗号分隔列表。*

spring.rabbitmq.cache.channel.checkout-timeout = *＃如果已达到缓存大小，则等待获取频道的持续时间。*

spring.rabbitmq.cache.channel.size = *＃缓存中要保留的通道数量。*

spring.rabbitmq.cache.connection.mode = channel *＃连接工厂缓存模式。*

spring.rabbitmq.cache.connection.size = *＃要缓存的连接数。*

spring.rabbitmq.connection-timeout = *＃连接超时。将其设置为零以永久等待。*

spring.rabbitmq.dynamic = true *＃是否创建一个AmqpAdmin bean。*

spring.rabbitmq.host = localhost *＃RabbitMQ主机。*

spring.rabbitmq.listener.direct.acknowledge-mode = *＃容器的确认模式。*

spring.rabbitmq.listener.direct.auto-startup = true *＃是否在启动时*

*自动启动容器。*

spring.rabbitmq.listener.direct.consumers-per-queue = *＃每个队列的使用者数量。*

spring.rabbitmq.listener.direct.default-requeue-rejected = *＃默认情况下拒绝的交付是否重新排队。*

spring.rabbitmq.listener.direct.idle-event-interval =*＃空闲容器事件应该多久发布一次。*

spring.rabbitmq.listener.direct.prefetch = *＃单个请求中要处理的消息数。它应该大于或等于事务大小（如果使用）。*

spring.rabbitmq.listener.direct.retry.enabled = false *＃是否启用发布重试。*

spring.rabbitmq.listener.direct.retry.initial-interval = 1000ms *＃第一次和第二次尝试传递消息之间的持续时间。*

spring.rabbitmq.listener.direct.retry.max-attempts = 3 *＃传递消息的最大尝试次数。*

spring.rabbitmq.listener.direct.retry.max-interval = 10000ms *＃尝试之间的最大持续时间。*

spring.rabbitmq.listener.direct.retry.multiplier = 1 *＃乘数应用于以前的重试间隔。*

spring.rabbitmq.listener.direct.retry.stateless = true *＃重试是无状态还是有状态。*

spring.rabbitmq.listener.simple.acknowledge-mode = *＃容器的确认模式。*

spring.rabbitmq.listener.simple.auto-startup = true *＃是否在启动时自动启动容器。*

spring.rabbitmq.listener.simple.concurrency = *＃监听器调用者线程的最小数量。*

spring.rabbitmq.listener.simple.default-requeue-rejected = *＃默认情况下是否拒绝交付重新排队。*

spring.rabbitmq.listener.simple.idle-event-interval = *＃应该发布空闲容器事件的频率。*

spring.rabbitmq.listener.simple.max-concurrency = *＃监听器调用者线程的最大数量。*

spring.rabbitmq.listener.simple.prefetch = *＃在单个请求中要处理的消息数。它应该大于或等于事务大小（如果使用）。*

spring.rabbitmq.listener.simple.retry.enabled = false *＃是否启用发布重试。*

spring.rabbitmq.listener.simple.retry.initial-interval = 1000ms *＃第一次和第二次尝试传递消息之间的持续时间。*

spring.rabbitmq.listener.simple.retry.max-attempt= 3 *＃传递消息的最大尝试次数。*

spring.rabbitmq.listener.simple.retry.max-interval = 10000ms *＃尝试之间的最大持续时间。*

spring.rabbitmq.listener.simple.retry.multiplier = 1 *＃乘数应用于之前的重试间隔。*

spring.rabbitmq.listener.simple.retry.stateless = true *＃重试是无状态还是有状态。*

spring.rabbitmq.listener.simple.transaction-size = *＃事务中要处理的消息数。也就是说，ack之间的消息数量。为了获得最佳结果，它应该小于或等于预取计数。*

spring.rabbitmq.listener.type = simple *＃监听器容器类型。*

spring.rabbitmq.password = guest *＃登录以对经纪人进行身份验证。*

spring.rabbitmq.port = 5672 *＃RabbitMQ端口。*

spring.rabbitmq.publisher-confirms = false *＃是否启用发布者确认。*

spring.rabbitmq.publisher-returns = false *＃是否启用发布商退货。*

spring.rabbitmq.requested-heartbeat = *＃请求的心跳超时; 零为零。如果未指定持续时间后缀，则将使用秒。*

spring.rabbitmq.ssl.enabled = false *＃是否启用SSL支持。*

spring.rabbitmq.ssl.key-store = *＃保存SSL证书的密钥存储区的路径。*

spring.rabbitmq.ssl.key-store-password = *＃用于访问密钥存储区的密码。*

spring.rabbitmq.ssl.key-store-type = PKCS12 *＃密钥库类型。*

spring.rabbitmq.ssl.trust-store = *＃持有SSL证书的信任库。*

spring.rabbitmq.ssl.trust-store-password = *＃用于访问信任存储的密码。*

spring.rabbitmq.ssl.trust-store-type = JKS *＃信任商店类型。*

spring.rabbitmq.ssl.algorithm = *＃使用的SSL算法。默认情况下，由Rabbit客户端库配置。*

spring.rabbitmq.template.exchange = *＃用于发送操作的默认交换的名称。*

spring.rabbitmq.template.mandatory = *＃是否启用强制消息。*

spring.rabbitmq.template.receive-timeout = *＃receive（）操作超时。*

spring.rabbitmq.template.reply-timeout = *＃sendAndReceive（）操作超时。*

spring.rabbitmq.template.retry.enabled = false *＃是否启用发布重试。*

spring.rabbitmq.template.retry.initial-interval = 1000ms *＃第一次和第二次尝试传递消息之间的持续时间。*

spring.rabbitmq.template.retry.max-attempts = 3 *＃传递消息的最大尝试次数。*

spring.rabbitmq.template.retry.max -interval = 10000ms*＃尝试之间的最大持续时间*

spring.rabbitmq.template.retry.multiplier = 1 *＃乘数应用于以前的重试间隔。*

spring.rabbitmq.template.routing-key = *＃用于发送操作的默认路由密钥的值。*

spring.rabbitmq.username = guest *＃登录用户向代理进行身份验证。*

spring.rabbitmq.virtual-host = *＃连接到代理时使用的虚拟主机。*

*＃----------------------------------------*

*＃ACTUATOR PROPERTIES*

*＃----- -----------------------------------*

*＃MANAGEMENT HTTP SERVER（*[ManagementServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/server/ManagementServerProperties.java)）

management.server.add-application-context-header = false *＃在每个响应中添加“X-Application-Context”HTTP标头。*

management.server.address = *＃管理端点应该绑定的网络地址。需要一个自定义的management.server.port。*

management.server.port = *＃管理端点HTTP端口（默认使用与应用程序相同的端口）。配置不同的端口以使用特定于管理的SSL。*

management.server.servlet.context-path = *＃管理端点上下文路径（例如`/ management`）。需要一个自定义的management.server.port。*

management.server.ssl.ciphers= *＃支持的SSL密码。需要一个自定义的management.port。*

management.server.ssl.client-auth = *＃是否需要客户端身份验证（“需要”）或需要（“需要”）。需要信任商店。需要一个自定义的management.server.port。*

management.server.ssl.enabled = *＃是否启用SSL支持。需要一个自定义的management.server.port。*

management.server.ssl.enabled-protocols = *＃启用SSL协议。需要一个自定义的management.server.port。*

management.server.ssl.key-alias = *＃标识密钥库中密钥的别名。需要一个自定义的management.server.port。*

management.server.ssl.key-password =*＃用于访问密钥存储区中密钥的密码。需要一个自定义的management.server.port。*

management.server.ssl.key-store = *＃保存SSL证书的密钥存储区的路径（通常是一个jks文件）。需要一个自定义的management.server.port。*

management.server.ssl.key-store-password = *＃用于访问密钥存储的密码。需要一个自定义的management.server.port。*

management.server.ssl.key-store-provider = *＃密钥存储的提供者。需要一个自定义的management.server.port。*

management.server.ssl.key-store-type = *＃密钥存储的类型。需要一个自定义的management.server.port。*

management.server.ssl.protocol = TLS*＃使用SSL协议。需要一个自定义的management.server.port。*

management.server.ssl.trust-store = *＃持有SSL证书的信任库。需要一个自定义的management.server.port。*

management.server.ssl.trust-store-password = *＃用于访问信任存储的密码。需要一个自定义的management.server.port。*

management.server.ssl.trust-store-provider = *＃信任存储的提供者。需要一个自定义的management.server.port。*

management.server.ssl.trust-store-type = *＃信任存储的类型。需要一个自定义的management.server.port。*

*＃CLOUDFOUNDRY*

management.cloudfoundry.enabled = true *＃是否启用扩展Cloud Foundry执行器端点。*

management.cloudfoundry.skip-ssl-validation = false *＃是否跳过针对Cloud Foundry执行器端点安全调用的SSL验证。*

*＃*

*终端常规配置*management.endpoints.enabled-by-default = *＃是否*默认*启用或禁用所有终端。*

[＃ENDPOINTS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/jmx/JmxEndpointProperties.java) *JMX配置（*[JmxEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/jmx/JmxEndpointProperties.java)）

management.endpoints.jmx.domain = org.springframework.boot *＃终结点JMX域名。如果设置，则回退到'spring.jmx.default-domain'。*

management.endpoints.jmx.exposure.include = \* *＃应包含的端点ID或全部包含的“\*”。*

management.endpoints.jmx.exposure.exclude = *＃应排除的端点ID。*

management.endpoints.jmx.static-names = *＃附加到表示端点的所有MBean的ObjectName的静态属性。*

management.endpoints.jmx.unique-names = false *＃是否确保ObjectNames在发生冲突时被修改。*

*＃终端WEB配置（*[WebEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/web/WebEndpointProperties.java)）

management.endpoints.web.exposure.include =健康，信息*＃应包含的终端ID或全部为'\*'。*

management.endpoints.web.exposure.exclude = *＃应该排除的端点ID。*

management.endpoints.web.base-path =/actuators *＃Web端点的基本路径。相对于server.servlet.context-path或management.server.servlet.context-path，如果配置了management.server.port。*

management.endpoints.web.path-mapping = *＃端点ID和应该暴露它们的路径之间的映射。*

[＃ENDPOINTS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/web/CorsEndpointProperties.java) *CORS CONFIGURATION（*[CorsEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/web/CorsEndpointProperties.java)）

management.endpoints.web.cors.allow-credentials = *＃是否支持凭证。未设置时，不支持凭证。*

management.endpoints.web.cors.allowed-headers = *＃在请求中允许使用逗号分隔的标题列表。'\*'允许所有标题。*

management.endpoints.web.cors.allowed-methods = *＃允许使用逗号分隔的方法列表。'\*'允许所有方法。未设置时，默认为GET。*

management.endpoints.web.cors.allowed-origins = *＃逗号分隔的起源列表允许。'\*'允许所有的来源。未设置时，CORS支持被禁用。*

management.endpoints.web.cors.exposed-headers = *＃包含在响应中的逗*

*号分隔的标题列表。*

management.endpoints.web.cors.max-age = 1800s *＃客户端可以缓存飞行前请求的响应时间。如果未指定持续时间后缀，则将使用秒。*

*＃审计事件ENDPOINT（*[AuditEventsEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/audit/AuditEventsEndpoint.java)）

management.endpoint.auditevents.cache.time-to-live = 0ms *＃响应可以被缓存的最长时间。*

management.endpoint.auditevents.enabled = true *＃是否启用auditevents端点。*

[＃BEANS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/beans/BeansEndpoint.java) *ENDPOINT（*[BeansEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/beans/BeansEndpoint.java)）

management.endpoint.beans.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.beans.enabled = true *＃是否启用bean端点。*

*＃条件REPORT ENDPOINT（*[ConditionsReportEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/condition/ConditionsReportEndpoint.java)）

management.endpoint.conditions.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.conditions.enabled = true *＃是否启用条件端点。*

*＃配置属性报告ENDPOINT（*[ConfigurationPropertiesReportEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/context/properties/ConfigurationPropertiesReportEndpoint.java)，[ConfigurationPropertiesReportEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/context/properties/ConfigurationPropertiesReportEndpointProperties.java)）

management.endpoint.configprops.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.configprops.enabled = true *＃是否启用configprops端点。*

management.endpoint.configprops.keys-to-sanitize =密码，秘密，密钥，令牌，\*凭证。\*，vcap\_services *＃应该清理的密钥。键可以是属性以正则表达式结尾的简单字符串。*

*＃ENVIRONMENT ENDPOINT（*[EnvironmentEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/env/EnvironmentEndpoint.java)，[EnvironmentEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/env/EnvironmentEndpointProperties.java)）

management.endpoint.env.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.env.enabled = true *＃是否启用env端点。*

management.endpoint.env.keys-to-sanitize =密码，秘密，密钥，令牌，\*凭证。\*，vcap\_services *＃应该清理的密钥。键可以是属性以正则表达式结尾的简单字符串。*

[＃FLYWAY](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/flyway/FlywayEndpoint.java) *ENDPOINT（*[FlywayEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/flyway/FlywayEndpoint.java)）

management.endpoint.flyway.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.flyway.enabled = true *＃是否启用飞桥端点。*

*＃HEALTH ENDPOINT（*[HealthEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthEndpoint.java)，[HealthEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/health/HealthEndpointProperties.java)）

management.endpoint.health.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.health.enabled = true *＃是否启用运行状况端点。*

management.endpoint.health.roles = *＃用于确定用户是否有权显示详细信息的角色。如果为空，则所有经过身份验证的用户均被授权*

management.endpoint.health.show-details = never *＃何时显示完整健康详情。*

*＃HEAP DUMP ENDPOINT（*[HeapDumpWebEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/management/HeapDumpWebEndpoint.java)）

management.endpoint.heapdump.cache.time-to-live = 0ms *＃响应可以被缓存的最长时间。*

management.endpoint.heapdump.enabled = true *＃是否启用heapdump端点。*

[＃HTTP](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/trace/http/HttpTraceEndpoint.java) *TRACE ENDPOINT（*[HttpTraceEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/trace/http/HttpTraceEndpoint.java)）

management.endpoint.httptrace.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.httptrace.enabled = true *＃是否启用httptrace端点。*

[＃INFO](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoEndpoint.java) *ENDPOINT（*[InfoEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoEndpoint.java)）

info = *＃添加到信息端点的任意属性。*

management.endpoint.info.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.info.enabled = true *＃是否启用信息端点。*

*＃JOLOKIA ENDPOINT（*[JolokiaProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/jolokia/JolokiaProperties.java)）

management.endpoint.jolokia.config.\* = *＃Jolokia设置。有关更多详细信息，请参阅Jolokia的文档。*

management.endpoint.jolokia.enabled = true *＃是否启用jolokia端点。*

*＃LIQUIBASE ENDPOINT（*[LiquibaseEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/liquibase/LiquibaseEndpoint.java)）

management.endpoint.liquibase.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.liquibase.enabled = true *＃是否启用liquibase端点。*

[＃LOG](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/logging/LogFileWebEndpoint.java) *FILE ENDPOINT（*[LogFileWebEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/logging/LogFileWebEndpoint.java)，[LogFileWebEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/logging/LogFileWebEndpointProperties.java)）

management.endpoint.logfile.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.logfile.enabled = true *＃是否启用日志文件端点。*

management.endpoint.logfile.external-file = *＃要访问的外部日志文件。如果日志文件是由输出重定向写入的，而不是日志记录系统本身，则可以使用。*

[＃LOGGERS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/logging/LoggersEndpoint.java) *ENDPOINT（*[LoggersEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/logging/LoggersEndpoint.java)）

management.endpoint.loggers.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.loggers.enabled = true *＃是否启用记录器端点。*

*＃请求MAPPING ENDPOINT（*[MappingsEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/web/mappings/MappingsEndpoint.java)）

management.endpoint.mappings.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.mappings.enabled = true *＃是否启用映射端点。*

[＃METRICS](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/MetricsEndpoint.java) *ENDPOINT（*[MetricsEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/MetricsEndpoint.java)）

management.endpoint.metrics.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.metrics.enabled = true *＃是否启用度量标准端点。*

*＃PROMETHEUS ENDPOINT（*[PrometheusScrapeEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/metrics/export/prometheus/PrometheusScrapeEndpoint.java)）

management.endpoint.prometheus.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.prometheus.enabled = true *＃是否启用普罗米修斯端点。*

*＃SCHEDULED TASKS ENDPOINT（*[ScheduledTasksEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/scheduling/ScheduledTasksEndpoint.java)）

management.endpoint.scheduledtasks.cache.time-to-live = 0ms *＃响应可以被缓存的最大时间。*

management.endpoint.scheduledtasks.enabled = true *＃是否启用scheduledtasks端点。*

*＃SESSIONS ENDPOINT（*[SessionsEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/session/SessionsEndpoint.java)）

management.endpoint.sessions.enabled = true *＃是否启用会话端点。*

*＃SHUTDOWN ENDPOINT（*[ShutdownEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/context/ShutdownEndpoint.java)）

management.endpoint.shutdown.enabled = false *＃是否启用关闭端点。*

*＃THREAD DUMP ENDPOINT（*[ThreadDumpEndpoint](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/management/ThreadDumpEndpoint.java)）

management.endpoint.threaddump.cache.time-to-live = 0ms *＃可以缓存响应的最长时间。*

management.endpoint.threaddump.enabled = true *＃是否启用线程转储端点。*

*＃健康指标*

management.health.db.enabled = true *＃是否启用数据库健康检查。*

management.health.cassandra.enabled = true *＃是否启用Cassandra健康检查。*

management.health.couchbase.enabled = true *＃是否启用Couchbase运行状况检查。*

management.health.defaults.enabled = true *＃是否启用默认运行状况指示器。*

management.health.diskspace.enabled = true *＃是否启用磁盘空间运行状况检查。*

management.health.diskspace.path = *＃用于计算可用磁盘空间的路径。*

management.health.diskspace.threshold = 0*＃应该可用的最小磁盘空间（以字节为单位）。*

management.health.elasticsearch.enabled = true *＃是否启用Elasticsearch运行状况检查。*

management.health.elasticsearch.indices = *＃逗号分隔的索引名称。*

management.health.elasticsearch.response-timeout = 100ms *＃等待集群响应的时间。*

management.health.influxdb.enabled = true *＃是否启用InfluxDB运行状况检查。*

management.health.jms.enabled = true *＃是否启用JMS运行状况检查。*

management.health.ldap.enabled = true *＃是否启用LDAP运行状况检查。*

management.health.mail.enabled = true*＃是否启用邮件运行状况检查。*

management.health.mongo.enabled = true *＃是否启用MongoDB运行状况检查。*

management.health.neo4j.enabled = true *＃是否启用Neo4j运行状况检查。*

management.health.rabbit.enabled = true *＃是否启用RabbitMQ运行状况检查。*

management.health.redis.enabled = true *＃是否启用Redis运行状况检查。*

management.health.solr.enabled = true *＃是否启用Solr运行状况检查。*

management.health.status.http-mapping = *＃健康状态到HTTP状态代码的映射。默认情况下，注册的健康状态映射到合理的默认值（例如，UP映射为200）。*

management.health.status.order = DOWN，OUT\_OF\_SERVICE，UP，UNKNOWN *＃以严重性顺序的逗号分隔的健康状态列表。*

[＃HTTP](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceProperties.java) *TRACING（*[HttpTraceProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceProperties.java)）

management.trace.http.enabled = true *＃是否启用HTTP请求 - 响应跟踪。*

management.trace.http.include = request-headers，response-headers，cookies，errors *＃要包含在跟踪中的项目。*

*＃信息*[贡献者](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/info/InfoContributorProperties.java)*（*[InfoContributorProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/info/InfoContributorProperties.java)）

management.info.build.enabled = true *＃是否启用*[*编译*](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/info/InfoContributorProperties.java)*信息。*

management.info.defaults.enabled = true *＃是否启用默认信息撰稿人。*

management.info.env.enabled = true *＃是否启用环境信息。*

management.info.git.enabled = true *＃是否启用git信息。*

management.info.git.mode = simple *＃用于公开git信息的模式。*

*＃METRICS*

management.metrics.binders.files.enabled = true *＃是否启用文件度量标准。*

management.metrics.binders.integration.enabled = true *＃是否启用Spring Integration指标。*

management.metrics.binders.jvm.enabled = true *＃是否启用JVM度量标准。*

management.metrics.binders.logback.enabled = true *＃是否启用Logback指标。*

management.metrics.binders.processor.enabled = true *＃是否启用处理器指标。*

management.metrics.binders.uptime.enabled = true *＃是否启用正常运行时间指标。*

management.metrics.distribution.percentiles-histogram。\* =*＃以指定名称开头的电表ID是否应该公布百分比直方图。*

management.metrics.distribution.percentiles。\* = *＃从指定名称开始计算出的计量器ID从后端运送到特定计算的不可汇总百分点。*

management.metrics.distribution.sla。\* = *＃以特定名称开始的电表ID的特定SLA边界。最长的匹配胜出，关键'全部'也可用于配置所有仪表。*

management.metrics.enable。\* = *＃是否启用以指定名称开头的电表ID。最长的匹配胜出，关键'全部'也可用于配置所有仪表。*

management.metrics.export.atlas.batch-size = 10000*＃每个请求用于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.atlas.config-refresh-frequency = 10s *＃刷新LWC服务配置设置的频率。*

management.metrics.export.atlas.config-time-to-live = 150s *＃为LWC服务的订阅生存的时间。*

management.metrics.export.atlas.config-uri = http：// localhost：7101 / lwc / api / v1 / expressions / local-dev *＃Atlas LWC端点检索当前订阅的URI。*

management.metrics.export.atlas.connect-timeout = 1s *＃对后端请求的连接超时。*

management.metrics.export.atlas.enabled= true *＃是否启用指标到此后端的导出。*

management.metrics.export.atlas.eval-uri = http：// localhost：7101 / lwc / api / v1 / evaluate *＃URI用于Atlas LWC端点评估订阅的数据。*

management.metrics.export.atlas.lwc-enabled = false *＃是否启用流式传输到Atlas LWC。*

management.metrics.export.atlas.meter-time-to-live = 15m *＃生活在*

*没有任何活动的米的时间。经过这段时间后，电表将被视为过期，并不会被报告。*

management.metrics.export.atlas.num-threads = 2 *＃用于度量标准发布计划程序的线程数。*

management.metrics.export.atlas.read超时= 10s *＃读取该后端请求的超时时间。*

management.metrics.export.atlas.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.atlas.uri = http：// localhost：7101 / api / v1 / publish *＃Atlas服务器的URI。*

management.metrics.export.datadog.api-key = *＃Datadog API密钥。*

management.metrics.export.datadog.application-key = *＃Datadog应用程序密钥。不是严格要求，而是通过向Datadog发送电表描述，类型和基本单位来改善Datadog体验。*

management.metrics.export.datadog.batch-size = 10000*＃每个请求用于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.datadog.connect-timeout = 1s *＃对后端请求的连接超时。*

management.metrics.export.datadog.descriptions = true *＃是否将描述元数据发布到Datadog。关闭此功能可最大限度地减少发送的元数据量。*

management.metrics.export.datadog.enabled = true *＃是否启用指标到此后端的导出。*

management.metrics.export.datadog.host-tag = instance *＃将标准传送到Datadog时将被映射到“主机”的标签。*

management.metrics.export.datadog.num线程= 2 *＃用于度量标准发布计划程序的线程数。*

management.metrics.export.datadog.read-timeout = 10s *＃读取该后端请求的超时时间。*

management.metrics.export.datadog.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.datadog.uri = https：//app.datadoghq.com＃*将指标发送到的URI。如果您需要将指标发布到通往Datadog的内部代理，则可以使用此代理定义代理的位置。*

management.metrics.export.ganglia.addressing-mode = multicast *＃UDP寻址模式，可以是单播或多播。*

management.metrics.export.ganglia.duration-units =毫秒*＃用于报告持续时间的基本时间单位。*

management.metrics.export.ganglia.enabled = true *＃是否启用向Ganglia导出度量标准。*

management.metrics.export.ganglia.host = localhost *＃Ganglia服务器的主机接收导出的度量标准。*

management.metrics.export.ganglia.port = 8649 *＃Ganglia服务器的端*

*口，用于接收导出的度量标准。*

management.metrics.export.ganglia.protocol-version = 3.1 *＃Ganglia协议版本。必须是3.1或3.0。*

management.metrics.export.ganglia.rate-units = seconds *＃用于报告费率的基本时间单位。*

management.metrics.export.ganglia.step = 1m*＃步长（即报告频率）使用。*

management.metrics.export.ganglia.time-to-live = 1 *＃生活在Ganglia指标上的时间。将多播时间生存时间设置为比主机之间的跳数（路由器）数量多一个。*

management.metrics.export.graphite.duration-units = milliseconds *＃用于报告持续时间的基本时间单位。*

management.metrics.export.graphite.enabled = true *＃是否启用指标到Graphite的导出。*

management.metrics.export.graphite.host = localhost *＃接收导出指标的Graphite服务器的主机。*

management.metrics.export.graphite.port = 2004*＃接收导出指标的Graphite服务器的端口。*

management.metrics.export.graphite.protocol = pickled *＃将数据传输到Graphite时使用的协议。*

management.metrics.export.graphite.rate-units = seconds *＃用于报告费率的基本时间单位。*

management.metrics.export.graphite.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.graphite.tags-as-prefix = *＃对于默认的命名约定，将指定的标记键转换为度量标准前缀的一部分。*

management.metrics.export.influx.auto-create-db = true *＃是否在尝试向其发布指标之前创建Influx数据库（如果它不存在）。*

management.metrics.export.influx.batch-size = 10000 *＃每个请求用于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.influx.compressed = true *＃是否启用发布到Influx的指标批次的GZIP压缩。*

management.metrics.export.influx.connect-timeout = 1s *＃对此后端请求的连接超时。*

management.metrics.export.influx.consistency = 1 *＃为每个点编写一致性。*

management.metrics.export.influx.db = mydb *＃将标准传送到Influx时将被映射到“主机”的标签。*

management.metrics.export.influx.enabled= true *＃是否启用指标到此后端的导出。*

management.metrics.export.influx.num-threads = 2 *＃用于指标发布计划程序的线程数。*

management.metrics.export.influx.password = *＃Influx服务器的登录密码。*

management.metrics.export.influx.read-timeout = 10s *＃读取该后端请求的超时时间。*

management.metrics.export.influx.retention-policy = *＃使用的保留策略（如果未指定DEFAULT保留策略，Influx写入DEFAULT保留策略）。*

management.metrics.export.influx.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.influx.uri= http://localhost:8086 *＃Influx服务器的URI。*

management.metrics.export.influx.user-name = *＃Influx服务器的登录用户。*

management.metrics.export.jmx.enabled = true *＃是否启用指标到JMX的导出。*

management.metrics.export.jmx.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.newrelic.account-id = *＃新的遗物账户ID。*

management.metrics.export.newrelic.api-key = *＃新的Relic API密钥。*

management.metrics.export.newrelic.batch-size = 10000*＃每个请求用于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.newrelic.connect-timeout = 1s *＃对此后端请求的连接超时。*

management.metrics.export.newrelic.enabled = true *＃是否启用度量标准导出到此后端。*

management.metrics.export.newrelic.num-threads = 2 *＃用于指标发布调度程序的线程数。*

management.metrics.export.newrelic.read-timeout = 10s *＃读取该后端请求的超时时间。*

management.metrics.export.newrelic.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.newrelic.uri = https://insights-collector.newrelic.com＃*将指标发送到的URI。*

management.metrics.export.prometheus.descriptions = true *＃是否启用发布说明，作为Prometheus的有效载荷的一部分。关闭此功能可以最大限度地减少每次扫描发送的数据量。*

management.metrics.export.prometheus.enabled = true *＃是否启用指标到Prometheus的导出。*

management.metrics.export.prometheus.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.signalfx.access-token = *＃SignalFX访问令牌。*

management.metrics.export.signalfx.batch-size = 10000*＃每个请求用*

*于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.signalfx.connect-timeout = 1s *＃对此后端*

*请求的连接超时。*

management.metrics.export.signalfx.enabled = true *＃是否启用度量标准导出到此后端。*

management.metrics.export.signalfx.num-threads = 2 *＃用于指标发布计划程序的线程数。*

management.metrics.export.signalfx.read-timeout = 10s *＃读取该后端请求的超时时间。*

management.metrics.export.signalfx.source =*＃唯一标识将度量标准发布到SignalFx的应用程序实例。默认为本地主机名称。*

management.metrics.export.signalfx.step = 10s *＃使用步长（即报告频率）。*

management.metrics.export.signalfx.uri = https：//ingest.signalfx.com＃*将指标发送到的URI。*

management.metrics.export.simple.enabled = true *＃是否在没有任何其他导出器的情况下启用度量标准导出到内存后端。*

management.metrics.export.simple.mode =累计*＃计数模式。*

management.metrics.export.simple.step = 1m *＃使用步长（即报告频率）。*

management.metrics.export.statsd.enabled= true *＃是否启用指标到StatsD的导出。*

management.metrics.export.statsd.flavor = datadog *＃StatsD要使用的协议。*

management.metrics.export.statsd.host = localhost *＃StatsD服务器的主机接收导出的度量标准。*

management.metrics.export.statsd.max-packet-length = 1400 *＃单个有效负载的总长度应保留在网络的MTU内。*

management.metrics.export.statsd.polling-frequency = 10s *＃调查仪表*

的频率*。当轮询仪表时，其值将被重新计算，如果值已更改（或者publishUnchangedMeters为true），则会将其发送到StatsD服务器。*management.metrics.export.statsd.port= 8125 *＃StatsD服务器的端口，用于接收导出的指标。*

management.metrics.export.statsd.publish-unchanged-meters = true *＃是否向StatsD服务器发送未更改的计量表。*

management.metrics.export.statsd.queue-size = 2147483647 *＃等待发送到StatsD服务器的项目队列的最大大小。*

management.metrics.export.wavefront.api-token = *＃将API度量标准直接发布到Wavefront API主机时使用。*

management.metrics.export.wavefront.batch-size = 10000 *＃每个请求用于此后端的度量数。如果找到更多的测量结果，则会发出多个请求。*

management.metrics.export.wavefront.connect-timeout = 1s *＃对后端请求的连接超时。*

management.metrics.export.wavefront.enabled = true *＃是否启用度量标准导出到此后端。*

management.metrics.export.wavefront.global-prefix = *＃在Wavefront用户界面中查看时，将源自此应用的白色盒子检测的度量与源自其他Wavefront集成的度量分开的全局前缀。*

management.metrics.export.wavefront.num-threads = 2 *＃用于指标发布计划程序的线程数。*

management.metrics.export.wavefront.read-timeout = 10s *＃读取该后端请求的超时时间。*

management.metrics.export.wavefront.source = *＃作为发布到Wavefront的指标来源的应用实例的唯一标识符。默认为本地主机名称。*

management.metrics.export.wavefront.step = 10s *＃步长（即报告频率）使用。*

management.metrics.export.wavefront.uri = https：//longboard.wavefront.com＃*将指标发送到的URI。*

management.metrics.use-global-registry = true *＃自动配置的MeterRegistry实现是否应绑定到Metrics上的全局静态注册表。*

management.metrics.web.client.max-uri-tags = 100*＃允许的最大唯一URI标记值数量。达到标签值的最大数量后，具有附加标签值的度量标准将被过滤器拒绝。*

management.metrics.web.client.requests-metric-name = http.client.requests *＃发送请求的度量标准的名称。*

management.metrics.web.server.auto-time-requests = true *＃是否应该自动计时由Spring MVC或WebFlux处理的请求。*

management.metrics.web.server.requests-metric-name = http.server.requests *＃接收请求的度量标准名称。*

*＃----------------------------------------*

*＃DEVTOOLS PROPERTIES*

*＃----- -----------------------------------*

*＃DEVTOOLS（*[DevToolsProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/autoconfigure/DevToolsProperties.java)）

spring.devtools.livereload.enabled = true *＃是否启用livereload.com兼容服务器。*

spring.devtools.livereload.port = 35729 *＃服务器端口。*

spring.devtools.restart.additional-exclude = *＃应该从触发完全重新启动时排除的其他模式。*

spring.devtools.restart.additional-paths = *＃观察更改的其他路径。*

spring.devtools.restart.enabled = true *＃是否启用自动重启。*

spring.devtools.restart.exclude= META-INF /行家/ \*\*，META-INF /资源/ \*\*，资源/ \*\*，静态/ \*\*，公共/ \*\*，模板/ \*\*，\*\* / \*的Test.class，\*\* / \* Tests.class，git.properties，META-INF / build-info.properties *＃应该排除触发完全重新启动的模式。*

spring.devtools.restart.log-condition-evaluation-delta = true *＃是否在重新启动时记录条件评估增量。*

spring.devtools.restart.poll-interval = 1s *＃轮询类路径更改之间等待的时间。*

spring.devtools.restart.quiet-period = 400ms *＃触发重新启动之前所需的静默时间，不需要任何类路径更改。*

spring.devtools.restart.trigger-file =*＃特定文件的名称，如果更改，则会触发重新启动检查。如果未指定，则任何类路径文件更改都会触发重新启动。*

*＃REMOTE DEVTOOLS（*[RemoteDevToolsProperties](https://github.com/spring-projects/spring-boot/tree/v2.0.0.RELEASE/spring-boot-project/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/autoconfigure/RemoteDevToolsProperties.java)）

spring.devtools.remote.context-path = /.~~ spring-boot！〜*＃用于处理远程连接的上下文路径。*

spring.devtools.remote.proxy.host = *＃用于连接远程应用程序的代理主机。*

spring.devtools.remote.proxy.port = *＃用于连接远程应用程序的代理端口。*

spring.devtools.remote.restart.enabled = true *＃是否启用远程重启。*

spring.devtools.remote.secret = *＃建立连接所需的共享密钥（启用远程支持所必需的）。*

spring.devtools.remote.secret头名= *用于传输共享密钥的* X-AUTH-TOKEN *＃HTTP标头。*

*＃----------------------------------------*

*＃测试属性*

*＃----- -----------------------------------*

spring.test.database.replace = any *＃要替换的现有DataSource的类型。*

spring.test.mockmvc.print =默认＃MVC *打印选项。*