Red Hat build of Node.js 20

Node.js 20 发行注记

用于 Node.js 20 LTS

Last Updated: 2024-05-03

Red Hat build of Node.js 20 Node.js 20 发行注记

用于 Node.js 20 LTS

法律通告

摘要

本发行注记包含与 Node.js 20 LTS 相关的重要信息。

目录

前言

发行版本日期： 2024-03-25

第 1 章 所需的基础架构组件版本

在使用红帽构建的 Node.js 时，需要以下基础架构组件。除了明确指定为支持的组件外，红帽不提供对这些组件的支持。

组件名称	版本
Nodeshift	2.1.1
npm 10	10.1.0
OpenShift Container Platform (OCP)[a]	3.11, 4.5
git	2.0 或更高版本
oc 命令行工具	3.11 或更高版本[b]

[a] 红帽支持 OCP

[b] **oc** CLI 工具的版本应该与您正在使用的 OCP 版本对应。

第 2 章 功能

本节包含有关红帽构建的 Node.js 20 发行版本中的功能更改的信息。

2.1. 新特性和更改的功能

Node.js 20 LTS 有以下红帽构建的 Node.js 支持的新功能和增强。

有关 Node.js 20 LTS 的详细信息，请参阅 上游社区 发行注记 和文档。

2.1.1. V8 JavaScript 引擎升级到 v11.3

此发行版本包括将 V8 JavaScript 引擎升级到 v11.3，它是 Chromium 113 的一部分。

升级的 V8 JavaScript 引擎包括以下新功能和增强：

- string.prototype.isWellFormed 和 String.prototype.toWellFormed 方法
- 带有集合表示法和字符串属性的 regexp 对象 **v** 标记
- **resizable ArrayBuffer 和 growable SharedArrayBuffer** 对象
- **WebAssembly tail 调用**

有关 V8 JavaScript 引擎中可用更改的更多信息，请参阅 V8 博客。

2.1.2. 测试运行程序模块的完整支持

红帽构建的 Node.js 20 将测试运行程序模块作为可在生产环境中使用的稳定功能提供。在早期版本中，test runner 模块仅作为实验性功能提供。

test runner 模块不旨在替换 Jest 或 Mocha 等全功能测试框架。test runner 模块提供了一种快速、简单的方法来编写和运行测试套件，而无需安装其他依赖项。

在红帽构建的 Node.js 20 中，测试运行程序模块包括以下类型的增强功能：

- **可以使用 node --test 标志调用的命令行测试运行程序**

- 使用 --test-reporter 标志配置和自定义测试报告器
- 使用 --experimental-test-coverage 标记进行实验性测试覆盖
- 模拟功能

如需更多信息，请参阅 Node.js Test Runner 文档。

2.1.3. 自定义 ESM loader hooks 在专用线程上运行

在红帽构建的 Node.js 20 中，您通过加载程序提供的 ECMAScipt 模块(ESM) hook （例如，--experimental-loader=myhook.mjs）在专用线程上运行，它与主应用程序线程分开。通过为加载程序提供单独的范围，这个增强可保护应用程序代码免受潜在的影响。

2.1.4. synchronous import.meta.resolve （） function

在红帽构建的 Node.js 20 中，import.meta.resolve （） 函数会以同步方式返回值。但是，根据您的偏好，您仍然可以定义自定义加载程序 解析 hook 作为同步或同chronous 功能。即使加载了异步 解析 hook，import.meta.resolve （） 函数也会为应用程序代码同步返回值。

2.1.5. Web Crypto API 使用 WebIDL 转换器

在红帽构建的 Node.js 20 中，Web Crypto API coerces 并根据其 WebIDL 定义验证功能参数，类似于 Web Crypto API 的其他实现。此功能增强有助于提高 Node.js 和 Web Crypto API 的浏览器实现之间的互操作性。

2.1.6. 属性 import.meta.resolve 不再依赖于 CLI 标记

从红帽构建的 Node.js 20 开始，import.meta.resolve (specifier) 属性不再依赖于 --experimental-import-meta-resolve 命令行界面(CLI)标记。现在，您可以使用 import.meta.resolve (specifier) 属性获取指定符字符串解析到的绝对字符串，这与 CommonJS 模块中的 require.resolve 功能类似。此功能增强有助于将 Node.js 与浏览器和其他服务器端运行时保持一致。

如需更多信息，请参阅 Node.js import.meta.resolve 文档。

2.1.7. method module.register 用于模块自定义 hook

红帽构建的 Node.js 20 提供了属于 node:module API 的新 module.register (specifier[, parentURL][, options]) 方法。您可以使用这个新方法指定导出模块自定义 hook 的文件，将数据传递给 hook，并使用这些 hook 建立通信频道。此增强会取代之前版本中的行为，这需要使用 --experimental-loader 标志来指定导出 hook 的文件。

为确保自定义 hook 在任何应用程序代码运行前注册，请考虑在运行使用寄存器功能的应用程序时使用 --import 标志。

例如：

```
node --import ./file-that-calls-register.js ./app.js
```

如需更多信息，请参阅 Node.js module.register 文档。

2.1.8. 对 CommonJS 模块的模块自定义 加载 hook 支持

红帽构建的 Node.js 20 可让模块自定义 hook 作者 处理负载 hook 中的 ESM 和 CommonJS 源。当应用程序的主入口点由 ESM loader 处理时，这个增强适用于使用 导入 或 require 语句引用的 CommonJS 模块。例如，当入口点是 ESM 文件或使用 --import 标志时，这个增强是相关的。这有助于简化 Node.js 加载过程的自定义，因为软件包作者可以执行额外的 Node.js 自定义，而无需依赖已弃用的 API，如 require.extensions。

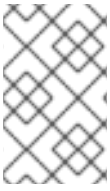
如需更多信息，请参阅 Node.js Hook：加载文档。

2.1.9. 增强的流性能

红帽构建的 Node.js 20 提高了可读和可写入流的性能。此功能增强包括改进，如减少内存开销，改进了对可读流的 async iterator 消耗，并改进了可读流的 pipeTo 消耗。

2.2. 已弃用的功能

红帽构建的 Node.js 20 发行版本中弃用了以下功能。



注意

有关本发行版本中已弃用或删除的功能的更多信息，请参阅 nodejs.org 网站。

2.2.1. 使用无效端口的 url.parse （） 方法运行时弃用

此发行版本包括包含无效端口的 url.parse （） 方法调用的运行时弃用。在以前的版本中，url.parse （） 方法接受包含非数字端口值的 URL，这可能会导致带有意外输入的主机名欺骗。在本发行版本中，如果 URL 包含非数字端口值，则 url.parse （） 方法会发出警告。

如需更多信息，请参阅 Node.js url.parse 文档。

2.3. 技术预览功能

在 Node.js 20 LTS 发行版本中，以下功能作为技术预览功能提供。

2.3.1. 实验性 WASI 功能增强

实验性 WebAsembly 系统接口(WASI)功能在此发行版本中包括以下改进：

- WASI 不再需要使用 experimental-wasi-unstable-preview1 命令行界面(CLI)标记来启用此功能。此功能增强有助于使 WASI 更易于使用。
- WASI 现在要求任何新的 WASI （） 调用都包含 version 选项来请求特定的 WASI 版本。因为 Node.js 支持不同版本的 WASI，且 version 选项没有默认值，所以您必须确保应用程序中的所有新 WASI （） 调用都请求特定版本。否则，错误结果。

如需更多信息，请参阅 Node.js WebAssembly System Interface (WASI)文档。

2.3.2. 权限模型

红帽构建的 Node.js 20 引入了实验性权限模型功能。此功能使开发人员能够限制在程序执行过程中对特定资源的访问，如文件系统操作、子进程生成和 worker 线程创建。这意味着，您可以防止应用程序访问或修改敏感数据，或者运行潜在的代码。

您可以使用 --experimental-permission CLI 标志启用 Permission Model 功能。通过启用此功能，您可以自动限制对所有可用权限的访问。

要管理权限，您可以使用以下 CLI 标志：

- 要管理文件系统权限，请使用 --allow-fs-read 和 --allow-fs-write 标志。
- 若要管理子进程权限，可使用 --allow-child-process 标志。
- 要管理 worker 线程权限，请使用 --allow-worker 标志。

如需更多信息，请参阅 Node.js 权限模型文档。

2.3.3. 追踪频道

红帽构建的 Node.js 20 引入了 TracingChannel 类，作为 Diagnostics Channel 功能的实验性扩展。追踪频道有助于对诊断频道进行分组，这些频道代表单个可追踪操作执行生命周期的不同点，用于生成和使用 trace 数据。因此，追踪频道有助于规范化和简化为追踪应用程序流生成事件的过程。

如需更多信息，请参阅 Node.js TracingChannel 文档。

2.3.4. Mock Timers

红帽构建的 Node.js 20 引入了一个实验性的 Mock Timers 功能，可让开发人员为依赖计时器的功能编写更可预测且可靠的测试。计时器模拟是一种常用的技术，可用于模拟和测试计时器行为，而无需等待指定的时间间隔。Mock Timers 功能包含一个 MockTimers 类，可让您从全局对象和 node:timers /promises API 中模拟对 setTimeout （） 和 setInterval （） 方法的调用。

Mock Timers 功能提供了一个简单的 API，用于提前时间，启用特定的计时器，并释放所有计时器。

例如：

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks setTimeout to be executed synchronously without having to actually wait for it', (context)
=> {
  const fn = context.mock.fn();
  // Optionally choose what to mock
  context.mock.timers.enable(['setTimeout']);
  const nineSecs = 9000;
  setTimeout(fn, nineSecs);
```

```
const threeSeconds = 3000;
context.mock.timers.tick(threeSeconds);
context.mock.timers.tick(threeSeconds);
context.mock.timers.tick(threeSeconds);

assert.strictEqual(fn.mock.callCount(), 1);
```

如需更多信息，请参阅 Node.js Mocking: Timers 文档。

2.3.5. 内置 .env 文件支持

红帽构建的 Node.js 20 引入了一个实验性功能，用于在 .env 文件中配置环境变量，该文件在启动时传递给 Node.js 应用程序。与 .ini 文件格式类似，.env 配置文件中的每一行包含特定环境变量的键值对。例如：

```
PASSWORD=nodejs
```

您可以使用 --env-file 标志，使用预定义的环境变量初始化并运行 Node.js 应用程序。例如：

```
node --env-file=myconfig.env myapp.js
```

按照上例所示初始化应用程序时，您可以使用 process.env 属性访问相关的环境变量。例如，若要访问 PASSWORD=node.js 环境变量，您可以使用 process.env.PASSWORD 属性。

此功能增强还允许您在 .env 配置文件中定义 NODE_OPTIONS 环境变量，无需在 package.json 文件中包含 NODE_OPTIONS。

2.4. 支持的构架

Node.js 构建器镜像和 RPM 软件包可用，并支持与以下 CPU 架构一起使用：

- AMD x86_64
- ARM64

- **OpenShift 环境中的 IBM Z (s390x)**

- **OpenShift 环境中的 IBM Power 系统(ppc64le)**

第 3 章 发行组件

- Node.js 20 Builder Image for RHEL 8
- Node.js 20 Universal Base Image 8
- Node.js 20 Minimal Stand-alone Image for RHEL 8
- Node.js 20 Minimal Base Image 8
- Node.js 20 Builder Image for RHEL 9
- Node.js 20 Universal Base Image 9
- Node.js 20 Minimal Stand-alone Image for RHEL 9
- Node.js 20 Minimal Base Image 9

第 4 章 修复的问题

此发行版本在 Node.js 20 LTS 社区版本中包含了所有修复的问题。

4.1. 增强的 VM 模块 API 中的内存管理

红帽构建的 Node.js 20 解决了一些长期内存泄漏，并在支持 importModuleDynamly 选项的以下虚拟机(VM)模块 API 中解决一些长期的内存泄漏和使用 问题：

- vm.Script
- vm.compileFunction
- vm.SyntheticModule
- vm.SourceTextModule

此功能增强支持受影响的用户从早期版本的 Node.js 升级。

第 5 章 已知问题

没有影响此版本的已知问题。

第 6 章 影响所需基础架构组件的已知问题

以下问题已知会影响此版本所需的基础架构组件。

6.1. NODE.JS 软件包管理器无法正确处理信号

描述

在红帽构建的 Node.js 20 中，Node.js 软件包管理器(npm)不会向子进程发送信号。此问题意味着 Node.js 应用无法正确关闭。

原因

npm 9.6.7 及更新版本中的一个问题会停止 npm 将 SIGINT 和 SIGTERM 信号转发到子进程。此问题会影响需要使用 npm 9.6.7 或更高版本的 Node.js 版本。Node.js 社区正在调查此 npm 问题的修复。

临时解决方案

目前还没有可用的临时解决方案。红帽正在调查红帽构建的 Node.js 20 中的此问题。

第 7 章 与本发行版本相关的公告

以下公告已发布以记录改进、错误修复和 CVE 修复。

- RHSA-2023:7205
- RHEA-2023:6529
- RHEA-2023:7249
- RHEA-2023:7252
- RHBA-2023:6753
- RHBA-2023:7223
- RHBA-2023:7271
- RHBA-2023:7514
- RHBA-2023:7611
- RHBA-2023:7799