

CICS Transaction Server for z/  
OSバージョン 5 リリース 6

*CICS* における *Node.js* の使用



## 注記

本書および本書で紹介する製品をご使用になる前に、[製品の特記事項](#)に記載されている情報をお読みください。

本書は、IBM® CICS® Transaction Server for z/OS®, バージョン 5 リリース 6 (製品番号 5655-Y305655-BTA)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

### 原典：

CICS Transaction Server for z/OS  
Version 5 Release 5  
Using Node.js in CICS

### 発行：

日本アイ・ビー・エム株式会社

### 担当：

トランスレーション・サービス・センター

© Copyright International Business Machines Corporation 1974, 2020.

---

# 目次

この PDF について.....	v
<b>第 1 章 CICS および Node.js.....</b>	<b>1</b>
Node.js ランタイム環境.....	2
Node.js と CICS バンドル.....	2
NODEJSAPP バンドル・パーツのライフサイクル.....	3
<b>第 2 章 Node.js アプリケーションの開発.....</b>	<b>5</b>
Node.js アプリケーション開発のベスト・プラクティス.....	5
Node.js アプリケーション内で使用する環境変数.....	6
CICS サービスの呼び出し.....	7
Node.js パイプラインに関する考慮事項.....	9
<b>第 3 章 Node.js アプリケーションのデプロイ.....</b>	<b>11</b>
Node.js ランタイムのインストールの検査.....	12
<b>第 4 章 Node.js サポートの設定.....</b>	<b>13</b>
Node.js プロファイルの検証およびプロパティ.....	13
Node.js プロファイルのコーディング規則.....	13
Node.js プロファイルおよびコマンド行オプション.....	14
Node.js プロファイルで使用するシンボル.....	17
Node.js アプリケーションでの時間帯の設定.....	18
NODEJSAPP 出力、ログ、トレースの場所の制御.....	19
z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与.....	19
Node.js のメモリー制限の設定.....	21
<b>第 5 章 Node.js のパフォーマンスの改善.....</b>	<b>23</b>
DFHSJNRO による NODEJSAPP のエンクレープの変更.....	23
Node.js アプリケーションのストレージ要件の計算.....	24
<b>第 6 章 Node.js アプリケーションのトラブルシューティング.....</b>	<b>27</b>
Node.js アプリケーションのトレースのアクティブ化と管理.....	28
Node.js アプリケーションの CICS コンポーネント・トレース.....	29
<b>特記事項.....</b>	<b>31</b>
<b>索引.....</b>	<b>37</b>



## この PDF について

---

この PDF では、CICS で Node.js アプリケーションを作成して使用方法について説明します。本書は、CICS の経験がほとんどなく、Node.js プログラムの開発と実行に必要な CICS の知識のみを求めている、経験のある Node.js アプリケーション・プログラマーを対象としています。また、Node.js サポートに必要な CICS 要件について知りたい、経験のある CICS ユーザーやシステム・プログラマーも対象としています。

本書で使用されている用語や表記について詳しくは、IBM Knowledge Center の [CICS 資料で使用されている表記規則および用語](#)を参照してください。

### 本書の作成日

本書は、2020 年 5 月 28 日に作成されました。



# 第 1 章 CICS および Node.js

Node.js は、JavaScript で作成されたアプリケーション用のサーバー・サイド・ランタイムです。

これには次のような特徴があります。

- イベント・ドリブン: HTTP 要求などのイベントを `listen` し、イベントが検出されたときにコールバック関数をトリガーします。
- シングル・スレッド化: 要求を 1 つずつ処理します。
- 非ブロッキング I/O: ファイル・システム、ソケット、データベースなどの入出力装置に対する読み取りと書き込みは、z/OS の基礎となるサポートを使用して非同期に発生し、完了時にコールバック関数を起動します。

Node.js は軽量かつ効率的で、データ集約型アプリケーションに最も適しています。z/OS の基礎となる非同期 I/O サポートを使用し、モジュール・ドリブンのハイ・スケーラブルなアプローチを提供して、アジャイル・プラクティスを促進するアプリケーション設計および開発を行うことができます。

また、企業内での地位を着実に確立し、REST サービスの提供と集約が可能なため、デジタル・トランスフォーメーションの優先的な選択になりつつあります。

Node.js の普及に大きく貢献しているのは、豊富な Node.js モジュールです。これらのモジュールはパブリック・サービス・レジストリで使用可能であり、ノード・パッケージ・マネージャー (NPM) を使用してアクセスできます。モジュールはほとんどのタスクで既に使用可能で、Node.js アプリケーション開発者のかなりの時間を節約できます。

Node.js は、Node.js とその関連モジュールの採用および開発を促進することを目的とする Node.js Foundation によって開発されています。詳しくは、[Node.js Foundation の Web サイト](#)を参照してください。

## Node.js アプリケーションからの CICS サービスの呼び出し

Node.js アプリケーションは、通常は長時間実行され、複数のユーザーからの TCP/IP ソケット要求を処理します。Node.js ランタイムは、アプリケーションごとに開始されます。1 つの CICS 領域に複数のアプリケーションが存在する場合があります。

CICS で実行される Node.js アプリケーションが、既存の CICS アプリケーションを呼び出さなければならない場合もあります。例えば Node.js アプリケーションが、フロントエンド・アプリケーションに単一のサービス・インターフェースを提供するために、既存のビジネス・ロジック機能の呼び出しを集約する場合があります。既存のビジネス・ロジック機能を使用すると、既存のアプリケーションに対する Node.js アプリケーションの近接性を活かせるので、フロントエンド・アプリケーションはネットワーク呼び出しを複数回にわたって行う必要がなくなります。また、Node.js アプリケーションは、外部サービスを呼び出すか NPM モジュールを使用することにより、既存のビジネス・ロジックに機能を追加することもできます。

Node.js アプリケーションは、既存のビジネス・ロジックを呼び出すために、CICS でホストされるサービスを呼び出すことができます。このようなサービスの例として、CICS Web サービス・テクノロジーまたは z/OS Connect を使用して公開される JSON Web サービスおよび SOAP Web サービスがあります。Node.js アプリケーションは、HTTP 要求を発行したり JSON および SOAP Web サービスを利用したりするための NPM モジュールを使用して、CICS サービスを呼び出すことができます。JSON は JavaScript のネイティブ・オブジェクト形式であるため、JSON Web サービスは Node.js アプリケーションにとって利用しやすいサービスです。

もう 1 つの方法として、呼び出す必要がある JSON Web サービスと同じ CICS 領域で Node.js アプリケーションがホストされる場合には、ローカル最適化トランスポートを使用することもできます。この場合は仮想記憶間アプローチを使用してサービスを呼び出すので、ネットワークを介したやりとりの必要がなくなります。ローカル最適化トランスポートを使用して CICS サービスを呼び出すためには、Node.js アプリケーションが `ibm-cics-api` モジュールを使用する必要があります。CICS JSON Web サービス・テクノロジーを使用してサービスが公開される必要があり、適合する PIPELINE リソースと URIMAP リソースが存在しなければなりません。詳しくは、[CICS サービスの呼び出し](#)を参照してください。

## コンポーネント

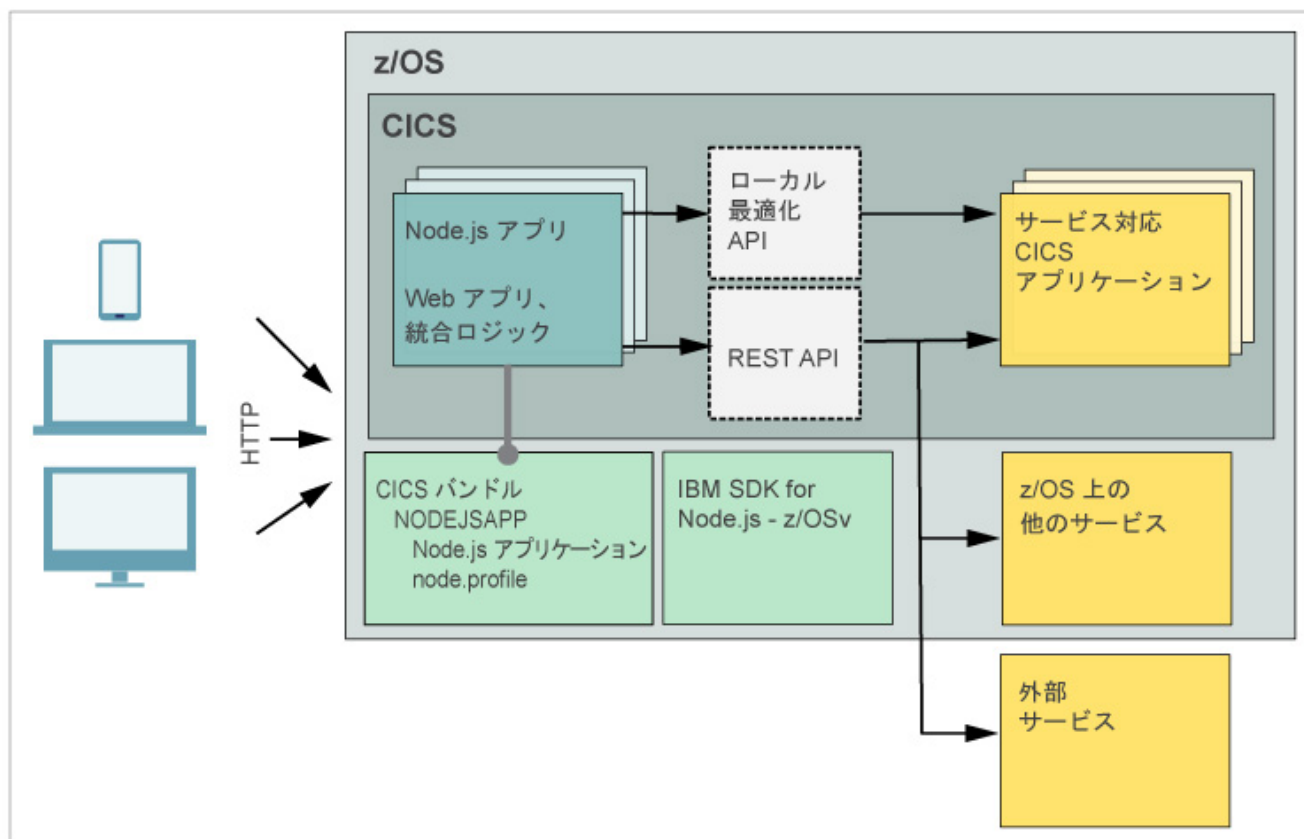


図 1. Node.js アプリケーションの CICS サポート

## Node.js ランタイム環境

CICS で Node.js アプリケーションを使用するには、IBM SDK for Node.js -z/OS をインストールする必要があります。これは、CICS V5.6 で Node.js アプリケーションによって使用される Node.js ランタイムを提供します。

システム要件について詳しくは、[詳細なシステム要件](#)を参照してください。

CICS 内の Node.js ランタイム環境は CICS 領域ユーザー ID のもとで実行され、言語環境 (LE) エンクレーブを伴います。エンクレーブは Node.js プロセスを実行し、Node.js アプリケーションごとにこのようなエンクレーブが 1 つのみ存在します。Node.js アプリケーションのワークロードはエンクレーブ内で実行され、他の Node.js アプリケーション・インスタンスから分離されます。

CICS の Node.js アプリケーションは、NODEJSAPP リソースで表されます。Node.js アプリケーションに必要な構成情報 (IBM SDK for Node.js -z/OS のインストール場所など) は、Node.js アプリケーション・プロファイルに指定されています。

このアプリケーション・プロファイルは、Node.js アプリケーションを構成する他の成果物とともに、CICS バンドルにパッケージ化する必要があります。CICS BUNDLE リソースは、Node.js アプリケーションを CICS に対して表現するもので、これを使用してアプリケーションのライフサイクルを管理することができます。バンドルが有効な場合、関連する Node.js アプリケーションはエンクレーブ内で実行されます。バンドルが無効な場合、関連する Node.js アプリケーションは停止します。

## Node.js と CICS バンドル

Node.js アプリケーションは、JavaScript ソース・コードとサポート・ファイルで構成されます。Node.js アプリケーションは、CICS NODEJSAPP バンドル・パーツを使用して CICS バンドルでデプロイされます。

NODEJSAPP バンドル・パーツは JavaScript コードを参照して、アプリケーションと Node.js アプリケーション・プロファイル (どちらも CICS バンドルにパッケージ化されています) を開始します。Node.js アプリ



ケーションに依存関係がある場合は、バンドルを有効にする前に NPM を使用して依存関係を解決する必要があります。

CICS バンドルが有効な場合、CICS は Node.js アプリケーション・プロファイル内の構成を使用して環境をセットアップし、IBM SDK for Node.js -z/OS によって提供される Node.js ランタイムを開始します。Node.js ランタイムによってアプリケーションが開始されます。

CICS バンドルが無効な場合、CICS は SIGTERM シグナルを送信して Node.js アプリケーションを停止しようとし、その後、Node.js ランタイムは除去されます。

Node.js アプリケーションのデプロイ方法について詳しくは、[Node.js アプリケーションのデプロイ](#)を参照してください。

## NODEJSAPP バンドル・パーツのライフサイクル

CICS では、NODEJSAPP バンドル・パーツによって Node.js アプリケーションのライフサイクルが管理されます。各 NODEJSAPP バンドル・パーツは、単一の Node.js アプリケーションと Node.js ランタイムの単一インスタンスを管理します。

### NODEJSAPP バンドル・パーツの有効化

CICS Explorer® または CEMT SET BUNDLE コマンドを使用して、NODEJSAPP バンドル・パーツを含む CICS BUNDLE を有効にすることができます。

Node.js アプリケーション・バンドル・パーツが有効になると、CICS は Node.js ランタイムを開始し、NODEJSAPP で指定されている初期 JavaScript ファイルを呼び出します。次に、CICS はバンドル・パーツの状況を ENABLED に設定します。アプリケーションで処理を受け入れる準備が整ったことを CICS は認識しません。

Node.js ランタイムの開始時にエラーが発生した場合 (例えば、Node.js プロファイルまたはアプリケーションのエラーなど)、CICS はバンドル・パーツの状況を DISABLED に設定します。

### NODEJSAPP バンドル・パーツの無効化

Node.js アプリケーションは、以下の 2 つの方法で無効にできます。

- アプリケーションが正常に完了し、CICS がバンドル・パーツの状況を DISABLED に設定します。
- CICS バンドルが無効になったために、バンドル・パーツが DISABLING 状態になります。

NODEJSAPP バンドル・パーツが DISABLING 状態になると、CICS は SIGTERM シグナルを Node.js プロセスに送信します。これにより、アプリケーションは、[Node.js アプリケーションの開発](#)で説明されているように、シグナルを受信し、正常にシャットダウンできます。設定された期間 (NODEJSAPP\_DISABLE\_TIMEOUT Node.js プロファイル・オプションで制御) が経過してもアプリケーションが自発的に終了しない場合、CICS は SIGKILL シグナルを Node.js プロセスに送信します。これにより、Node.js ランタイムは直ちに終了します。ただし、終了時にシステム・リソース (Unix メッセージ・キューなど) が割り振り解除されない可能性があります。アプリケーションで SIGTERM 通知に対処することをお勧めします。これに失敗すると、時間の経過とともにシステム・リソース・リークが起こる恐れがあります。

Node.js ランタイムが終了すると、CICS は、ローカル最適化 API を使った呼び出し要求によって開始されたすべてのタスクが終了するまで待機します。その後、NODEJSAPP バンドル・パーツは DISABLED 状態になります。



## 第 2 章 Node.js アプリケーションの開発

Node.js アプリケーションは JavaScript を使用して作成され、COBOL プログラマーや Java™ プログラマーにとって不慣れとも思われる非同期プログラミングの概念を必要とします。経験豊かな Node.js 開発者は、CICS 用アプリケーションの開発は他のプラットフォーム用の Node.js アプリケーションの開発と似たプロセスであるということを知っています。開発とデバッグのスキルが共用され、Node Package Manager (NPM) が同じ方法で使用されます。

CICS には、Node.js アプリケーションから CICS プログラムを呼び出すための API が用意されています。この API は、ネットワークを介して既存の CICS 資産をサービスとして呼び出すのではなく、それらの CICS 資産とやり取りするためのローカル最適化手段を提供します。

トピック 5 ページの『[Node.js アプリケーション開発のベスト・プラクティス](#)』は、Node.js アプリケーションを開発する際に注意する必要があるアプリケーション・プログラミングの側面を取り上げています。

説明付きの z/OS 用サンプル Node.js アプリケーションは、[Github](#) でダウンロードできます。

### CICS における Node.js の制約

共通 Node.js ライブラリーのほぼすべてが CICS 内で使用可能ですが、いくつかの制約を受けます。これらの制約は、ネイティブ・コードと、基礎となるオペレーティング・システムとの相互作用に関係します。サード・パーティー API の実装でプラットフォーム固有のネイティブ・コードが使用されている場合、その実装は z/OS に移植されていない可能性があります。z/OS に移植されていないものを使用したい場合は、そのコードの作成者にサポートを求めなければならない場合があります。新しいオペレーティング・システム・プロセスを API が作成する必要がある場合、その機能は CICS 内で使用できません。このため、以下の Node.js API は CICS と非互換であることが分かっています。

- [子プロセス](#)
- [クラスター](#)
- [process.setegid\(id\)](#)
- [process.seteuid\(id\)](#)
- [process.setgid\(id\)](#)
- [process.setgroups\(groups\)](#)
- [process.setuid\(id\)](#)

## Node.js アプリケーション開発のベスト・プラクティス

Node.js アプリケーションを開発するときは、環境変数を使用してリソースの構成を外部化することを考慮する必要があります。Node.js アプリケーションで SIGTERM シグナルを処理して、正常に終了できるようにすることも考慮する必要があります。

### 環境変数の使用

Node.js アプリケーションで TCP/IP ポートや URI、データベースなどのリソースにアクセスする場合は、環境変数を使用してそのリソースの構成を外部化することをお勧めします。これにより、アプリケーションが、開発環境、テスト環境、実稼働環境にデプロイされるときに、異なる値を指定できます。

例えば、HTTP 要求を listen する TCP/IP ポートは、CICS Node.js アプリケーション・プロファイルで次のように指定できます。

```
PORT=8080
```

Node.js アプリケーションは、[process.env](#) プロパティを次のように使用して値を取得できます。

```
var httpPort = process.env.PORT
```

## 正常終了

Node.js アプリケーションを含む CICS BUNDLE が DISABLED の場合、CICS はシグナル `SIGTERM` を Node.js プロセスに送信します。これにより、Node.js アプリケーションが正常に終了する機会が与えられます。正常に終了するための方法として、例えば、新規接続をこれ以上受け入れないようにして、持続接続を停止し、未処理の要求を完了して、最終的にファイルとデータベース接続をクローズして、終了します。アプリケーションは、オプション `NODEJSAPP_DISABLE_TIMEOUT` で指定された期間内に終了する必要があります。

`SIGTERM` シグナルを処理して、HTTP サーバーを閉じる例を以下に示します。

```
var http = require('http');
var httpPort = process.env.PORT;
var process = require('process');

//create a server object
var server = http.createServer(
  function (request, response) {
    response.write('Hello World! PID:' + process.pid); //write a response to the client
    response.end(); //end the response
  }
);

process.on('SIGTERM',
  function () {
    server.close(
      function () {
        console.log('Received SIGTERM at ' + (new Date()));
      }
    );
  }
);

console.log("Started hello.js at " + (new Date()));
server.listen(httpPort);
```

これらの手法の両方の例が Node.js IVP サンプルで示されています。詳しくは、[Node.js ランタイムのインストールの検査](#)を参照してください。

## Node.js アプリケーション内で使用する環境変数

Node.js アプリケーションは、環境変数を使用して、CICS バンドルと環境に関する情報を見つけることができます。

### **CICS\_APPLID**

CICS 領域アプリケーション ID `APPLID` の SIT パラメーターの値。

### **CICS\_BUNDLE**

Node.js アプリケーションが含まれている CICS バンドルを管理するために使用される `BUNDLE` リソースの名前。

### **CICS\_BUNDLEID**

Node.js アプリケーションが含まれている CICS バンドルの ID。

### **CICS\_LOCAL\_CCSID**

`LOCALCCSID` SIT パラメーターの値。

### **CICS\_LOG**

Node.js アプリケーションの動作中に CICS がログ・メッセージを書き込む先の z/OS UNIX ファイルの名前。

### **CICS\_NODEJSAPP**

CICS バンドル内にある `NODEJSAPP` の名前。

### **CICS\_OUTPUT\_DIR**

`STDOUT`、`STDERR`、`LOG`、および `TRACE` ファイルを格納するディレクトリー。`CICS_WORK_DIR` に対する相対パスで指定します。つまり `<APPLID>/<BUNDLEID>/<NODEJSAPP>`。

**CICS\_PROFILE\_PATH**

Node.js アプリケーション・プロファイルの完全修飾ファイル名。

**CICS\_STDERR**

Node.js 標準エラー (STDERR) の書き込み先となる完全修飾ファイル名。

**CICS\_STDOUT**

Node.js 標準出力 (STDOUT) の書き込み先となる完全修飾ファイル名。

**CICS\_TRACE**

Node.js 標準トレース (TRACE) の書き込み先となる完全修飾ファイル名。

**CICS\_USSCONFIG**

UNIX System Services 構成 USSCONFIG の SIT パラメーターの値。

**CICS\_USSHOME**

UNIX System Services ホーム USSHOME の SIT パラメーターの値。

**CICS\_WORK\_DIR**

Node.js アプリケーションの作業ディレクトリ。

**使用例**

環境変数は、`process.env` グローバル変数を使用して Node.js アプリケーションでアクセスできます。以下に例を示します。

```
console.log("Node.js application " + process.env.CICS_NODEJSAPP +  
  " is running in CICS region " + process.env.CICS_APPLID);
```

**関連リンク**

[Node.js プロファイルおよびコマンド行オプション](#)

## CICS サービスの呼び出し

`ibm-cics-api` モジュールに含まれる呼び出し関数を使用して、CICS サービスを呼び出すことができます。Node.js には、非同期アクティビティーのための 2 つの一般的なメカニズム (コールバック関数と `promise`) があります。どちらの手法も使用できます。

CICS 内で Node.js アプリケーションが実行されている場合、呼び出し要求が行われると CICS はターゲット JSON サービスを実行するための新しいタスクを開始します。アプリケーションと CICS の間でデータが JSON 形式で渡された後、アプリケーションに応答が返されます。次の例には一般的な開始があり、その後に CICS サービスを呼び出すための 2 つのオプションがあります。コールバック関数を使用することも、`promise` を使用することもできます。

```
const cics = require('ibm-cics-api');  
  
let uri = "http://winmvs2c.hursley.ibm.com/exampleApp/json_inquireCatalogWrapper";  
let requestData =  
{  
  "inquireCatalogRequest":  
  {  
    "startItemRef": 10,  
    "itemCount": 774  
  }  
};  
  
//Version using callback  
cics.invoke(uri,requestData,function(err, data)  
{  
  if (err)  
  {  
    ... do something with error ....  
  }  
  else  
  {  
    ... do something with response data  
  }  
});  
  
//Version using promises
```

```
cics.invoke(uri,data).then
(
function(response)
{
... do something with response ...
},
function(err)
{
console.log(err);
}
);
```

CICS サービスを呼び出す際に、以下の 3 つのパラメーターが使用されます。

- URI (ストリング) - ターゲット・サービスを特定します。CICS 内の URIMAP とのマッチングに使用され、URIMAP は WEBSERVICE、PIPELINE、そして最終的にターゲット PROGRAM にマップされます。
- 要求データ (JavaScript オブジェクトまたはストリング) - CICS サービスに要求本体として送信されるデータ。このデータは、CICS 内のターゲット・サービスによって予期される JSON 形式でなければなりません。
- コールバック関数 - この関数は、応答が Node.js アプリケーションで処理される準備ができたときに CICS によって呼び出されます。promise を使用する場合、この関数は適用されません。

コールバック関数に渡されるパラメーターは、以下のとおりです。

- エラー・オブジェクト。要求が成功した場合はヌルです。失敗した場合は、エラーを示す JavaScript Error オブジェクトです。
- 応答オブジェクト。CICS からの応答を表す JavaScript オブジェクトです。

## エラー処理

Node.js アプリケーションからの要求の処理中にエラーが発生すると、CICS は JavaScript Error オブジェクトを作成し、コールバック関数に渡します。httpCode と呼ばれる整数値は、相当する HTTP 状況のエラーを表します。次の例は、stderr に出力されたエラーを示しています。

```
{ Error: CICS error: internal server error
  at Error (native)
  code: 'ERR_CICS_INTERNAL_ERROR',
  httpCode: 500,
  response: '{"Fault":{"faultstring":"Failure interacting
with CICS","detail":{"CICSFault": "DFHPI1007 08\\21\\2018 09:06:17
IYK2ZKE1 CNJW 00121 JSON to data transformation failed because
of incorrect input (UNDEFINED_ELEMENT foo) for WEBSERVICE
json_inquireCatalogWrapper."}}}' }
```

コード	メッセージ	httpCode	説明
ERR_CICS_BAD_REQUEST	CICS エラー: 間違った要求	400	アプリケーションから CICS に渡されたデータにエラーがあります。典型的なエラーとして、誤った形式の URI や無効の URI があります。
ERR_CICS_FORBIDDEN	CICS エラー: 禁止	403	許可エラー。たいていの場合、URIMAP に関連付けられたユーザー ID にターゲット TRANSACTION への接続権限が付与されていません。
ERR_CICS_NOT_FOUND	CICS エラー: リソースが見つかりません	404	構成エラー。処理に関連する CICS リソースの 1 つが見つかりません。欠落リソースは、URIMAP、TRANSACTION、または PIPELINE リソースである場合があります。最も一般的な問題として、CICS にとって不明の URI が使用されていることが挙げられます。

コード	メッセージ	httpCode	説明
ERR_CICS_INTERNAL_ERROR	CICS エラー: 内部サーバー・エラー	500	CICS 内で問題が発生しました。典型的な問題としては、ターゲット・サービスが異常終了したか、データ変換エラーが発生したことが挙げられます。問題を示す診断が CICS によって発行されます。エラー・オブジェクトの「response」属性に説明メッセージがある場合があります。最も一般的な問題として、JSON 要求データと CICS 内で予期されている JSON 形式の不一致があります。
ERR_CICS_UNAVAILABLE	CICS エラー: 使用不可	503	CICS 内のリソースが使用不可になっています。URIMAP、TRANSACTION、または PIPELINE リソースのいずれかが現在使用不可になっています。

## 単体テスト

呼び出し関数は、ローカル CICS 呼び出しとリモート CICS 呼び出しをサポートします。CICS 内で Node.js アプリケーションが実行される場合、要求は、ネットワークを使用しない仮想記憶間システム呼び出しに最適化されます。CICS 外で Node.js アプリケーションが実行される場合、要求は、URI に基づいて HTTP または HTTPS を使用して CICS に送信されます。このリモート機能は単体テスト専用です。

ibm-cics-api モジュールを使用して HTTP で CICS に接続する場合、基本認証などの HTTP ヘッダーを設定したり、クライアント証明書を送信したりすることはできません。そうする必要がある場合は、代わりに他の Node.js モジュールを使用してください。

## Node.js パイプラインに関する考慮事項

Node.js アプリケーションは、呼び出し関数を使用して CICS 内の JSON パイプラインを開始することで、JSON Web サービスとして使用可能になっている既存の CICS プログラムを呼び出すことができます。既存の JSON パイプラインを Node.js アプリケーションで再利用できますが、以下の考慮事項があります。

### パイプラインの選択

ローカル最適化 Node.js アプリケーションからの呼び出し関数は、HTTP プロトコルで CICS に到達しません。HTTP プロトコルを使用するパイプラインは、Node.js での使用に適さない可能性があります。例えば、パイプラインに登録されたハンドラー・プログラムが EXEC CICS WEB API を使用して HTTP 要求のヘッダーとやり取りしようとする、CICS からエラー応答を受け取ります。呼び出し関数について詳しくは、[CICS サービスの呼び出し](#)を参照してください。

パイプライン・トランスポート・ハンドラー・プログラムを使用して特定のトランスポート実装にパイプラインがバインドされている場合、そのパイプラインは Node.js アプリケーションからのローカル最適化呼び出し関数での使用に適さない可能性があります。CICS は、呼び出し関数で指定された URI と URIMAP リソースの照合に基づいて、Node.js 要求用に使用するパイプラインを選択します。関連する URIMAP は USAGE(PIPELINE) を示していなければなりません。USAGE(PIPELINE) が示されていない場合、URIMAP は、Node.js アプリケーションからのローカル最適化呼び出し関数での使用に適していません。

特定の TCPIP SERVICE リソースに URIMAP がバインドされている場合、その URIMAP はその TCPIP SERVICE からの作業のみを受け入れます。Node.js アプリケーションからのローカル最適化呼び出し関数は HTTP を使用せず、TCPIP SERVICE リソースをバイパスします。そのような URIMAP はすべて、Node.js アプリケーション呼び出し関数からのローカル最適化要求での使用に適していません。

### パイプライン・ハンドラー・プログラム

CICS はいくつかのタイプの JSON パイプラインをサポートします。詳しくは、[JSON 要求のサービス・プロバイダーとしての CICS](#)を参照してください。Node.js アプリケーションからのローカル最適化呼び出し関数での使用には、2つの構成が適しています。これらは、CICS Java パイプライン、および CICS 提供端末ハンドラー DFHPIJT と呼ばれます。z/OS Connect for CICS または CICS Liberty JVM サーバーの JAX-RS および JSON フィーチャーを使用して実装された JSON インフラストラクチャーは、使用に適格ではあ



りません。DFHPIJT 端末ハンドラー (非 Java JSON サービス・プロバイダーとも呼ばれる) は、Node.js にとっておそらく最高と思えるパフォーマンス特性を備えています。これらの選択肢のどちらも、ハンドラー・プログラム内でターゲット作業のユーザー ID とトランザクション ID をプログラマチックに選択すること (コンテキスト・スイッチングともいう) はサポートしていません。ユーザー ID とトランザクション ID のカスタマイズについては、[Node.js アプリケーションに関するセキュリティ](#)を参照してください。

Node.js アプリケーションからのローカル最適化呼び出し関数の代わりに呼び出されていることを検出するパイプライン・ハンドラー・プログラムを作成することができます。そのようなハンドラー・プログラムは、要求の処理に関してプログラムによる決定を行えます。例えば、Node.js アプリケーションから到着した作業を拒否したり、HTTP を介して到着した作業のみに関連するロジックを分岐させたりすることもできます。DFHWS-NODEJSAPP と呼ばれる制御コンテナの内容が、現行チャンネル上の CICS によって取り込まれます。この内容から、作業の発生元になった NODEJSAPP リソースを特定できます。ハンドラー・プログラムとチャンネル接続アプリケーションは、このコンテナに照会できます。



## 第 3 章 Node.js アプリケーションのデプロイ

NODEJSAPP バンドル・パーツを作成することによって、Node.js アプリケーションを CICS バンドルでデプロイできます。

### 始める前に

この作業を始める前に、Node.js アプリケーションの場所と必要な資産 (イメージや HTML ファイルなど) を特定する必要があります。

プロファイルを作成して、CICS が Node.js ランタイムを開始するために使用する構成情報を指定します。このプロファイルを使用して、Node.js アプリケーションがその機能を構成するためにアクセスできる環境変数を指定することもできます。

適切な CICS バンドル・プロジェクトを見つけるか、新しいプロジェクトを作成する必要があります。

Node.js アプリケーションを CICS バンドル・プロジェクトにコピーするか、別の zFS の場所で参照する必要があります。

### このタスクについて

以下の手順に従って、CICS バンドルに NODEJSAPP バンドル・パーツを作成できます。

### 手順

1. 「Project Explorer (プロジェクト・エクスプローラー)」ビューの CICS Explorer で、CICS バンドル・プロジェクトを右クリックし、「新規」>「**Node.js アプリケーション (Node.js Application)**」をクリックします。  
「**Node.js アプリケーションの作成 (Create Node.js Application)**」というタイトルのウィザードが開きます。
2. Node.js アプリケーション・バンドル・パーツの名前を入力します。この名前は、バンドルをインストールする CICS 領域内で固有でなければなりません。「次へ」をクリックします。
3. 実行する初期 JavaScript ファイルを選択します。zFS にエクスポートされた JavaScript ファイルの完全修飾パスが 255 文字を超えていないことを確認する必要があります。「次へ」をクリックします。
4. Node.js アプリケーションのプロファイルを選択します。zFS にエクスポートされたプロファイルの完全修飾パスが 255 文字を超えていないことを確認する必要があります。「完了」をクリックします。

### タスクの結果

完了すると、ウィザードによってバンドル・プロジェクト内に NODEJSAPP バンドル・パーツが作成されます。ローカル・ファイル・システムまたは Eclipse ワークスペースからプロファイルを選択した場合、プロファイルは CICS バンドル・プロジェクトにコピーされます。バンドルが使用可能な場合、CICS は Node.js アプリケーションの初期 JavaScript ファイルを実行するために、プロファイルの構成を使用して Node.js ランタイムを開始します。

### 次のタスク

CICS バンドル・プロジェクトをデプロイできるようになりました。

### CICS バンドル・プロジェクトのデプロイ

Node.js アプリケーションは一般的に外部モジュールに依存しているため、アプリケーションの実行前にそれらの依存関係を解決する必要があります。CICS プロジェクトをデプロイした後、IBM SDK for Node.js - z/OS によって提供される npm ツールを z/OS UNIX システム・サービス・シェルから実行する必要があります。例えば、依存関係が Node.js アプリケーションの package.json ファイルで記述されている場合、**npm install** コマンドを使用して、依存関係をリポジトリからダウンロードし、インストールします。

## Node.js ランタイムのインストールの検査

このタスクでは、CICS で Node.js アプリケーションを実行できることを確認するステップと、予期しない結果が生じた場合に役立つ診断フィードバックがある場所について説明します。

### 始める前に

IBM SDK for Node.js -z/OS をインストールし、そのインストール・ディレクトリーに対する読み取り権限と実行権限を CICS 領域ユーザー ID に付与します。

CICS バンドル・ディレクトリー `/usr/lpp/cicsts/cicsts56/samples/nodejs/nodejsivp` が存在していることを確認します。

### 手順

1. CICS バンドル・ディレクトリーとその内容を、選択した新しい場所にコピーします。  
例えば、`cp -R /usr/lpp/cicsts/cicsts56/samples/nodejs/nodejsivp /u/jdoe/` のようにします。
2. CICS バンドルのコピー内で、Node.js プロファイル `profiles/ivp_sample.profile` を編集します。  
以下の環境変数を更新します。
  - `PORT=`: Node.js アプリケーションが Web ブラウザー要求へのサービス提供に使用できる、使用可能な HTTP ポート番号に設定します。ポートを他のアプリケーションと共用することはできません。
3. CICS バンドルの Node.js プロファイルには、システム全体の Node.js 構成データを含む `zFS` ファイルを参照する `%INCLUDE` ステートメントが含まれます。このファイルを `<USSCONFIG>/nodejsprofiles/general.profile` に作成します。ここで、`<USSCONFIG>` はターゲット CICS 領域内の `USSCONFIG SIT` パラメーターの値に置き換えます。  
このファイルには、以下の値を設定する必要があります。
  - `NODE_HOME=`: IBM SDK for Node.js -z/OS インストール・ディレクトリーに設定します。
  - `WORK_DIR=`: 出力診断ファイルが書き込まれるディレクトリーに設定します。値 `.` は、CICS 領域ユーザー ID のホーム・ディレクトリーを意味します。
4. サンプル・リソース定義 `DFHNJIVP` をグループ `DFH$NODJ` から選択したグループにコピーします。
5. `DFHNJIVP` のコピーを編集して、ステップ 12 ページの『1』の CICS バンドルのコピーのディレクトリーを `BUNDLEDIR` 属性を設定します。
6. `DFHNJIVP` のコピーをインストールします。Node.js アプリケーションが開始し、HTTP 要求を `listen` します。
7. Web ブラウザーを使用して、HTTP 要求を Node.js アプリケーションに送信します。  
例えば、`http://hostname:port/` のようにします。ここで、`hostname` は CICS が実行されている `z/OS` 上の TCP/IP スタックの完全修飾ホスト名、`port` はステップ 12 ページの『2』で選択された値です。  
Web ブラウザーに、「おめでとうございます。Node.js IVP アプリケーション `IVPSAMPLE` は正常に実行されました。(Congratulations, you have successfully run Node.js IVP Application `IVPSAMPLE`.)」という応答が表示されます。

### 次のタスク

Web ブラウザーに予期される応答が表示されない場合は、[Node.js アプリケーションのトラブルシューティングの診断情報](#)を確認してください。

### 関連情報

# 第 4 章 Node.js サポートの設定

CICS 領域で Node.js をサポートするための基本的なセットアップ・タスクを実行します。

## Node.js プロファイルの検証およびプロパティ

Node.js プロファイルには、開始時に Node.js ランタイムに渡される一連のオプションと環境変数が含まれています。

Node.js プロファイルは通常、CICS バンドルの一部としてデプロイされますが、zFS 上の絶対ファイル・パスを参照することもできます。

## Node.js プロファイルのコーディング規則

Node.js プロファイルは、USS ファイル・システムに保管されるときに EBCDIC でエンコードされるテキスト・ファイルです。Node.js プロファイルは、CICS バンドル内に作成されると、任意のテキスト・エディターを使用してワークステーション上で編集できます。これらは USS への転送時に EBCDIC に変換する必要があります。CICS Explorer は、CICS バンドル・プロジェクトを USS にエクスポートするときに、この変換を自動的に実行します。Node.js プロファイルをコーディングする際には、以下の規則に従ってください。

### 大/小文字の区別

パラメーター・キーワードおよびオペランドはすべて、大/小文字の区別があり、CICS 環境における JVM のオプション、JVM システム・プロパティ、または Node.js プロファイルおよびコマンド行オプションに示されているとおり正確に指定する必要があります。

### コメント

コメントを追加するか、オプションを削除する代わりにコメント化するには、コメントの各行の先頭に # 記号を付けます。ファイルがランチャーによって読み取られるときに、コメント行は無視されます。

空白行も無視されます。オプション間、またはオプションのグループ間の分離文字として空白行を使用できます。

プロファイル構文解析コードは、インライン・コメントを除去します。インライン・コメントは以下のように定義されます。

- コメントの先頭に # 記号が付いている。
- 前に 1 つ以上のスペース (またはタブ) がある。
- 引用符付きテキストには含まれない。

表 1. インライン・コメントの例	
コード	結果
MYVAR=myValue # Comment	MYVAR=myValue
MYVAR=#myValue # Comment	MYVAR=#myValue
MYVAR=myValue "# Quoted comment" # Comment	MYVAR=myValue "# Quoted comment"

### 継続

オプションの場合、値は、テキスト・ファイル内の行の終わりで区切られます。入力または編集しようとする値が、エディターのウィンドウには長すぎる場合は、スクロールしないですむように改行することができます。次の行に継続するには、次の例のように、現在行の終わりに円記号 (¥) 文字と空白の継続文字を付けます。

```
STDERR=/example/a/long/path/which/you/would/like\  
/to/break/over/a/line
```

同じ行に複数のオプションを置くことはできません。

## ファイルの組み込み

%INCLUDE=<file\_path> を使用して、プロファイルにファイルを組み込みます。ファイルには、プロファイルとは別個に保守できる、システム全体に対する共通の構成を含めることができます。これにより、複数のプロファイルに共通する構成を共用することが可能になるので、プロファイルを制御しやすくして、保守を容易にすることができます。

以下の規則が適用されます。

- <file\_path> は、zFS 内の完全修飾ファイルでなければなりません。
  - <file\_path> の先頭で相対ディレクトリー (. や .. など) を使用しないでください。それらは UNIX System Services では、Language Environment® の現行作業ディレクトリーを基準とした相対ディレクトリーとして解釈されます。この位置は処理中に変更されることがあります。
  - <file\_path> が存在しない場合や CICS 領域のユーザー ID に <file\_path> に対する読み取り権限がない場合は、メッセージ DFHSJ1308 が発行されます。
- <file\_path> には、&USSCONFIG; などのシンボルを含めることができます。
  - シンボルの &DATE; と &TIME; は許可されません。それらのフォーマットが、%INCLUDE ディレクティブの前にも後にも指定できる時間帯オプション (TZ) によって設定されるためです。
- %INCLUDE ディレクティブは、<file\_path> の内容に置き換わります。
- プロファイルには、任意の数の %INCLUDE ディレクティブを含めることができます。
- 循環参照の結果として、Skipping duplicate というメッセージが表示されます。例えば、Profile-A に Profile-B を含めることや Profile-B に Profile-C を含めることは可能ですが、Profile-B に Profile-A が含まれている場合、そのディレクティブは無視されます。

## オプションの複数インスタンス

同じオプションの複数のインスタンスがプロファイルに含まれている場合、最後に検出されるオプションの値が使用され、それより前の値は無視されます。

## UNIX システム・サービスのディレクトリー・パス

プロファイルで zFS ファイルまたはディレクトリーの値を指定する場合は、引用符を使用しないでください。

## Node.js プロファイルに固有の規則

### プロファイルの名前

- 名前は、z/OS UNIX システム・サービス内のファイルに有効な任意の名前にすることができます。DFH で始まる名前を使用しないでください。これらの文字は CICS が使用するために予約されているからです。
- プロファイルは UNIX ファイルであるため、大文字小文字が重要です。CICS で名前を指定する場合は、z/OS UNIX ファイル名にあるのと同じ大文字と小文字の組み合わせを使用して名前を入力する必要があります。

## Node.js プロファイルおよびコマンド行オプション

Node.js プロファイルとコマンド行オプションが、その説明とともにリストされます。

### コマンド行オプション

Node.js プロファイルにコマンド行オプションを指定できます。それは Node.js ランタイムに渡されます。ハイフンで始まるすべての行は、コマンド行オプションとして扱われます。コマンド行オプションは、1 行に 1 つ指定する必要があります。コマンド行オプションの追加パラメーターは、同じ行に指定する必要があります。

以下に、プロファイル内にコマンド行オプションを設定する例を示します。

```
--v8-pool-size=0
--redirect-warnings=/file/path
--require "/path/modules/module.js"
```

## プロファイルのオプションとその説明

デフォルト値 (該当する場合) とは、オプションが指定されていない場合に CICS で使用される値のことです。サンプル Node.js プロファイルで、デフォルト値とは異なる値が指定される場合があります。

**LIBPATH\_PREFIX=pathnames**

**LIBPATH\_SUFFIX=pathnames**

Node.js ランタイムまたはモジュールによって使用されるネイティブ C++ ダイナミック・リンク・ライブラリー (DLL) ファイル (拡張子は z/OS UNIX で .so) の検索対象となるディレクトリー・パスを指定します。これには、アプリケーション・コードまたはサービスによってロードされる Node.js アプリケーションおよび追加のネイティブ・ライブラリーを実行するのに必要なファイルが含まれます。

Node.js ランタイムの基本ライブラリー・パスは、**USSHOME** システム初期設定パラメーターと Node.js プロファイルの **NODE\_HOME** オプションで指定されたディレクトリーを使用して自動的に作成されます。

**LIBPATH\_SUFFIX** オプションを使用すると、このライブラリー・パスを拡張できます。このオプションはディレクトリーを、ライブラリー・パスの最後、基本ライブラリー・パスの後に追加します。このオプションは、アプリケーションに必要な Node.js モジュールで使用する追加のネイティブ・ライブラリーが含まれるディレクトリーを指定するために使用します。

**LIBPATH\_PREFIX** オプションは、ライブラリー・パスの先頭で、基本ライブラリー・パスの前にディレクトリーを追加します。このオプションは注意して使用してください。指定されたディレクトリー内の DLL ファイルが、基本ライブラリー・パス上の DLL ファイルと同じ名前である場合は、提供されたファイルの代わりにロードされます。

パス名に含まれる複数のパスは、コンマではなくコロンを使って区切ってください。

**LOG={&APPLID;.&NODEJSAPP;.&Dyyyymmdd.&Thmmss.log|file\_name}**

Node.js アプリケーションの動作中に CICS がログ・メッセージを書き込む先の z/OS UNIX ファイルの名前を指定します。このファイルに書き込まれるメッセージのレベルは、**LOG\_LEVEL** オプションによって制御されます。

絶対ファイル名を **LOG** で指定した場合、CICS は存在しないディレクトリーをそのパス内に作成します。

ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。Node.js アプリケーションごとに固有の出力ファイルを作成するには、サンプル Node.js プロファイルで示されているように **&NODEJSAPP**; シンボルと **&APPLID**; シンボルをファイル名で使います。

**LOG\_FILES\_MAX={0|number}**

システム上に保持する古いログ・ファイルの数を指定します。デフォルト設定の 0 では、すべての古いバージョンのログ・ファイルが保持されます。この値を変更して、ファイル・システム上に保持する古いログ・ファイルの数を指定できます。

If **STDOUT**、**STDERR**、**STDIN**、**LOG**、および **TRACE** がデフォルト・スキームを使用している場合、またはカスタマイズされている場合は、**&DATE**; **&TIME**; パターンを含めると、各ログ・タイプの最新の nn のみがシステムに保持されます。カスタマイズに出力を固有にする変数が含まれていない場合は、ファイルが追加され、削除の要求はありません。特殊値 0 は、削除しないことを意味します。

**LOG\_LEVEL={INFO|WARNING|ERROR|NONE}**

.log ファイルに入れて戻されるログ記録情報についての制御を提供します。値が **NONE** の場合、すべての出力は抑止され、ファイルは空になります。それ以外の値は、.log ファイルに書き込まれる最低ログ・タイプを示します。例えば、**WARNING** を選択すると、**WARNING** レベル以上のログ項目になります。

## **NODE\_HOME=pathname**

z/OS UNIX で IBM SDK for Node.js -z/OS のインストール場所を指定します。これは必須パラメーターです。

## **NODEJSAPP\_DISABLE\_TIMEOUT={10000|number}**

NODEJSAPP を無効にしようとする際に CICS が待機するタイムアウト値 (ミリ秒単位) を指定します。デフォルトは 10000 (10 秒) です。指定できる最小値は 1000 (1 秒) です。

これは、CICS が SIGTERM シグナルを送信した後に Node.js アプリケーション・プロセスの終了を待機する時間の最小の長さです。このタイムアウト値が満了する前にプロセスが終了しない場合、CICS は SIGKILL シグナルを発行して、プロセスを強制的に終了します。

## **PRINT\_PROFILE={TRUE|FALSE}**

このオプションを TRUE に設定すると、Node.js ランタイムとアプリケーションに渡される Node.js プロファイル内のオプションおよび環境変数は、SYSPRINT に出力されます。

## **STDERR={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.stderr|file\_name}**

STDERR ストリームをリダイレクトする z/OS UNIX ファイルの名前を指定します。このオプションに値を設定しない場合、CICS によって自動的に、Node.js アプリケーションごとに固有のトレース・ファイルが作成されます。

絶対ファイル名を STDERR で指定した場合、CICS は存在しないディレクトリーをそのパス内に作成します。

ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。Node.js アプリケーションごとに固有の出力ファイルを作成するには、サンプル Node.js プロファイルで示されているように &NODEJSAPP; シンボルと &APPLID; シンボルをファイル名で使します。STDERR がデフォルトのままである場合、CICS は &APPLID; と &NODEJSAPP; シンボル、および Node.js アプリケーションが固有の出力ファイルの作成を開始した日時のタイムスタンプを使します。

## **STDIN=file\_name**

STDIN ストリームの読み取り元の z/OS UNIX ファイルの名前を指定します。このオプションに値が指定されない限り、CICS はこのファイルを作成しません。

## **STDOUT={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.stdout|file\_name}**

STDOUT ストリームをリダイレクトする z/OS UNIX ファイルの名前を指定します。このオプションに値を設定しない場合、CICS によって自動的に、Node.js アプリケーションごとに固有のトレース・ファイルが作成されます。

絶対ファイル名を STDOUT で指定した場合、CICS は存在しないディレクトリーをそのパス内に作成します。

ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。Node.js アプリケーションごとに固有の出力ファイルを作成するには、サンプル Node.js プロファイルで示されているように &NODEJSAPP; シンボルと &APPLID; シンボルをファイル名で使します。STDOUT がデフォルトのままである場合、CICS は &APPLID; と &NODEJSAPP; シンボル、および Node.js アプリケーションが固有の出力ファイルの作成を開始した日時のタイムスタンプを使します。

## **TRACE={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.trace|file\_name}**

Node.js アプリケーションの稼働中に Node.js トレースを書き込む z/OS UNIX ファイルの名前を指定します。このオプションに値を設定しない場合、CICS によって自動的に、Node.js アプリケーションごとに固有のトレース・ファイルが作成されます。

絶対ファイル名を TRACE で指定した場合、CICS は存在しないディレクトリーをそのパス内に作成します。

ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。Node.js アプリケーションごとに固有の出力ファイルを作成するには、サンプル Node.js プロファイルで示されているように &NODEJSAPP; シンボルと &APPLID; シンボルをファイル名で使します。



## WORK\_DIR={./|/tmp|pathname}

CICS 領域が Node.js アプリケーションに関連した STDOUT、STDERR、TRACE などの出力ファイルに使用する、z/OS UNIX 上のディレクトリーを指定します。提供された Node.js プロファイルではピリオド (.) が定義されています。これは、CICS 領域ユーザー ID のホーム・ディレクトリーが作業ディレクトリーとして使われることを示します。このディレクトリーは、CICS インストール時に作成されます。このディレクトリーが存在しないか、WORK\_DIR が省略されている場合には、z/OS UNIX ディレクトリー名として /tmp が使用されます。

作業ディレクトリーの絶対パスまたは相対パスを指定できます。作業ディレクトリーの相対パスは、CICS 領域ユーザー ID のホーム・ディレクトリーから見た相対パスです。ホーム・ディレクトリーを Node.js に関連した活動の作業ディレクトリーとして使用しない場合や、CICS 領域が z/OS ユーザー ID (UID) を共用しているために同じホーム・ディレクトリーを持っている場合には、CICS 領域ごとに異なる作業ディレクトリーを作成できます。

&APPLID; シンボル (これは CICS によって実際の CICS 領域 APPLID に置換されます) を使用するディレクトリー名を指定すると、すべての CICS 領域で Node.js プロファイルのセットを共用する場合であっても、領域ごとに固有の作業ディレクトリーを作成できます。例えば、次のように指定するとします。

```
WORK_DIR=/u/&APPLID;/nodeoutput
```

その Node.js プロファイルを使用する各 CICS 領域は、それぞれ独自の作業ディレクトリーを持ちます。該当するディレクトリーが z/OS UNIX 上に作成されていること、およびそれに対する読み取り/書き込み/実行アクセス権限が CICS 領域に与えられていることを確認してください。

また、作業ディレクトリーの固定名を指定することもできます。この場合は、適切なディレクトリーが z/OS UNIX に作成され、アクセス権限が正しい CICS 領域に付与されていることを確認する必要があります。作業ディレクトリーに固定名を使用する場合は、Node.js プロファイルを共用する CICS 領域内のすべての Node.js アプリケーションからの出力ファイルが、そのディレクトリーに作成されます。ご使用の出力ファイルに固定のファイル名を使用する場合には、こうした CICS 領域内のすべての Node.js アプリケーションからの出力は同じ z/OS UNIX ファイルに追加されます。同じファイルに追加されないようにするには、&NODEJSAPP; シンボルと &APPLID; シンボルを使用して、Node.js アプリケーションごとに固有の出力およびダンプ・ファイルを生成します。

**USSHOME** システム初期設定パラメーターで定義された CICS ファイルのホーム・ディレクトリーである、z/OS UNIX 上の CICS インストール・ディレクトリーで、作業ディレクトリーを定義しないでください。

## 関連リンク

[Node.js アプリケーション内で使用する環境変数](#)

## Node.js プロファイルで使用されるシンボル

Node.js プロファイルに指定する変数では、置換シンボルを使用できます。これらのシンボルの値は実行時に決定されるため、多くの Node.js アプリケーションおよび CICS 領域で共通のプロファイルを使用できます。

以下のシンボルがサポートされています。

### &APPLID;

このシンボルを使用すると、シンボルは、**APPLID** システム初期設定パラメーターの値に置き換えられます。これは CICS 領域のアプリケーション ID です。この方法ですべての領域に同じプロファイルを使用でき、同時に領域固有の作業ディレクトリーまたは出力宛先を用いることができます。APPLID は常に大文字です。

### &BUNDLE;

このシンボルを使用すると、シンボルは、NODEJSAPP のインストール元となっている CICS バンドルの名前に置き換えられます。

### &BUNDLEID;

このシンボルを使用すると、シンボルは、NODEJSAPP のインストール元となっている CICS バンドルの ID に置き換えられます。

**&CONFIGROOT;**

このシンボルを使用すると、NODEJSAPP のインストール元であるバンドルのルート・ディレクトリーが実行時に置換されます。

**&DATE;**

このシンボルを使用すると、シンボルは実行時に *Dyymmdd* 形式の現在日付で置き換えられます。

**&NODEJSAPP;**

このシンボルを使用する際、NODEJSAPP リソースの名前は実行時に置換されます。Node.js アプリケーションごとに固有の出力またはダンプ・ファイルを作成する場合に、このシンボルを使用します。

**&TIME;**

このシンボルを使用すると、シンボルは実行時に *Thhmmss* 形式の NODEJSAPP 開始時刻で置き換えられます。

**&USSCONFIG;**

このシンボルを使用すると、シンボルは、[USSCONFIG](#) システム 初期設定パラメーターの値に置換されます。これは CICS 構成ファイルのディレクトリーです。

**&USSHOME;**

このシンボルを使用すると、シンボルは、[USSHOME](#) システム 初期設定パラメーターの値に置換されます。これは CICS Transaction Server 製品ファイルのディレクトリーです。

## Node.js アプリケーションでの時間帯の設定

TZ 環境変数はシステムの「ローカル」時間を指定します。この変数を JVM プロファイルに追加することで、JVM サーバーのローカル時間を設定できます。TZ 変数を設定しないと、システムによってデフォルトで UTC に設定されます。TZ 変数が設定されると、JVM は必要に応じて夏時間調整を自動的に行います。このとき、再始動や追加の介入が生じることはありません。

JVM サーバーまたは Node.js アプリケーションの時間帯を設定する際には、以下の点に注意する必要があります。

- JVM または Node.js プロファイルの TZ 変数が GMT を基準にしたローカル MVS™ システムのオフセットと一致するようにします。ローカル MVS システム・オフセットを表示および設定する方法については、「[z/OS Communications Server: IP 構成解説書](#)」の『[TIMEZONE ステートメント](#)』および『[z/OS MVS シスプレックスのセットアップ](#)』の『[地方時の調整](#)』を参照してください。
- 時間帯のカスタマイズはサポートされていないため、カスタマイズすると UTC にフェイルオーバーするか、または時間帯が混在した状態で JVMTRACE ファイル (JVM サーバーの場合) または TRACE ファイル (Node.js アプリケーションの場合) に出力されます。
- LOCALTIME を時間帯ストリングとして表示すると、構成に不整合が生じます。この不整合は、ローカル MVS 時間と設定する TZ の間、またはローカル MVS 時間と JVM または Node.js プロファイルのデフォルト設定の間に生じる可能性があります。各項目は正しくても、時間帯が混在した状態で出力されます。

### POSIX タイム・ゾーン形式の使用

POSIX タイム・ゾーン形式には、省略形と完全形があります。どちらも TZ 環境変数の設定に使用できますが、省略形を使用すると、入力ミスの可能性を減らせます。

完全形のサンプル:

```
TZ=GMT0BST,M3.5.0,M10.5.0
```

省略形のサンプル:

```
TZ=GMT0BST
```

システムが実行されているタイム・ゾーンを調べるには、USS にログオンし、`echo $TZ` と入力します。結果は、TZ 環境変数が設定されている値の完全形になります。

```
/u/user:>echo $TZ
GMT0BST,M3.5.0,M10.4.0
```



POSIX タイム・ゾーン形式の詳細については、IBM developerWorks® サイトの [POSIX および Olson のタイム・ゾーンの形式](#)を参照してください。

## NODEJSAPP 出力、ログ、トレースの場所の制御

Node.js アプリケーション、Node.js ランタイム、および CICS からの出力は、z/OS UNIX ファイルに書き込まれます。z/OS UNIX ファイルは、CICS Node.js プロファイルの STDOUT、STDERR、LOG、および TRACE オプションを使用して指定されます。

デフォルトでは、Node.js アプリケーションからの出力は、z/OS UNIX ファイル・システムに書き込まれます。デフォルトのファイル名規則は WORK\_DIR/APPLID/BUNDLEID/NODEJSAPP/D<date>.T<time>.<stdxxx> です。ここで、WORK\_DIR、APPLID、BUNDLEID、NODEJSAPP、date、および time は、トピック [Node.js プロファイルの検証およびプロパティ](#)で詳しく説明されている値に置き換えられるシンボルです。

デフォルトをオーバーライドする場合は、STDOUT、STDERR、LOG、および TRACE の各プロファイル・オプションに対して zFS ファイル名を指定できます。既に存在しているファイル名を使用すると、出力はそのファイルに付加されます。この状態のときは、問題判別に役立つように、Node.js アプリケーション・インスタンスを識別するためのヘッダーを、出力の各行の前に付けることをお勧めします。

## z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与

CICS では、z/OS UNIX 内のディレクトリーとファイルへのアクセスが必要です。各 CICS 領域には、インストール時に z/OS UNIX ユーザー ID (UID) が割り当てられます。これらの領域は、z/OS UNIX グループ ID (GID) を割り当てられた ESM グループに接続されます。この UID および GID を使用して、CICS 領域が z/OS UNIX 内のディレクトリーおよびファイルにアクセスする権限を付与します。

### 始める前に

自分が z/OS UNIX のスーパーユーザーであるか、ディレクトリーおよびファイルの所有者であることを確認してください。ディレクトリーおよびファイルの所有者は、最初は、製品をインストールしたシステム・プログラマーの UID として設定されます。ディレクトリーおよびファイルの所有者は、インストール時に GID が割り当てられた ESM グループに接続されなければなりません。所有者は、この ESM グループをデフォルト・グループ (DFLTGRP) として指定するか、補足グループの 1 つとして接続することができます。

### このタスクについて

z/OS UNIX システム・サービスでは、個々の CICS 領域が単一の UNIX ユーザーとして扱われます。z/OS UNIX ディレクトリーおよびファイルにアクセスするユーザー権限をさまざまな方法で付与することができます。例えば、CICS 領域の接続先の ESM グループに対して、ディレクトリーまたはファイルに関する適切なグループ権限を付与できます。このオプションは、実稼働環境に最適な場合があり、以下の手順で説明しています。

### 手順

1. CICS 領域からのアクセスが必要となる z/OS UNIX 内のディレクトリーおよびファイルを識別します。

構成	デフォルト・ディレクトリー	アクセス権	説明
<a href="#">BUNDLE リソースの BUNDLEDIR 属性</a>	ユーザーが設定。	読み取りおよび実行	CICS バンドルが含まれるディレクトリー。
<a href="#">Node.js プロファイル中の NODE_HOME オプション</a>	ユーザーが設定。 注：Node.js のデフォルト・インストール・ディレクトリーは /usr/lpp/IBM/cnj/v8r0/IBM/node-latest-os390-s390x です。	読み取りおよび実行	IBM SDK for Node.js -z/OS インストール・ディレクトリー。

構成	デフォルト・ディレクトリー	アクセス権	説明
<a href="#">USSHOME SIT パラメーター</a>	/usr/lpp/cicsts/ cicsts56	読み取りおよび実行	z/OS UNIX 上の CICS ファイルのインストール・ディレクトリー。このディレクトリー内のファイルには、サンプル・プロファイルが含まれます。
<a href="#">Node.js プロファイルの中の WORK DIR オプション</a>	/u/CICS region userid	読み取り、書き込み、および実行	CICS 領域の作業ディレクトリー。このディレクトリーには、Node.js アプリケーションからの入力、出力、およびメッセージが入っています。

2. ディレクトリーおよびファイルをリストして、権限を表示します。

開始したいディレクトリーに移動し、次の UNIX コマンドを発行します。

```
ls -la
```

現行ディレクトリーが CICSHT## のホーム・ディレクトリーであるときに、このコマンドが z/OS UNIX システム・サービスのシェル環境で発行されると、次の例のようなリストが表示される場合があります。

```
/u/cicsht##:>ls -la
total 256
drwxr-xr-x  2 CICSHT## CICS56      8192 Mar 15  2008 .
drwx----- 4 CICSHT## CICS56      8192 Jul  4 16:14 ..
-rw----- 1 CICSHT## CICS56      2976 Dec  5 2010 Snap0001.trc
-rw-r--r--  1 CICSHT## CICS56      1626 Jul 16 11:15 stderr
-rw-r--r--  1 CICSHT## CICS56         0 Mar 15  2010 stdin
-rw-r--r--  1 CICSHT## CICS56      458 Oct  9 14:28 stdout
/u/cicsht##:>
```

3. グループ権限を使用してアクセス権限を付与する場合は、各ディレクトリーおよびファイルのグループ権限が、CICS がリソースに対して必要とするアクセス・レベルを認可するものであることを確認します。

r、w、x、および - という文字を使用して 3 組の権限が指定されます。これらの文字は、「読み取り」、「書き込み」、「実行」、および「なし」を表し、コマンド行の左側の列の 2 文字目から表示されます。最初の組は所有者権限、2 番目の組はグループ権限、3 番目の組はその他の権限です。

前述の例では、所有者には stderr、stdin、および stdout に対する読み取りおよび書き込み権限がありますが、グループおよびその他すべてには、読み取り権限しかありません。

4. 特定のリソースのグループ権限を変更したい場合は、UNIX コマンド chmod を使用します。

次の例では、指定されたディレクトリーおよびそのサブディレクトリーとファイルのグループ権限を、読み取り、書き込み、および実行に設定します。-R は、すべてのサブディレクトリーおよびファイルに権限を再帰的に適用します。

```
chmod -R g=rwx directory
```

次の例では、指定されたファイルのグループ権限を読み取りおよび実行に設定します。

```
chmod g+rx filename
```

次の例では、指定された 2 つのファイルでグループに対する書き込み権限をオフにします。

```
chmod g-w filename filename
```

上記のすべての例で、g はグループ権限を指定します。その他に訂正したい権限がある場合、u はユーザー (所有者) 権限を指定し、o はその他の権限を指定します。

5. CICS 領域から z/OS UNIX にアクセスするために選択された RACF® グループに対して、リソースごとにグループ権限を割り当てます。個々のディレクトリーとそのサブディレクトリー、およびそこに含まれるファイルに対して、グループ権限を割り当てる必要があります。

次の UNIX コマンドを入力します。

```
chgrp -R GID directory
```

GID は ESM グループの GID の数値であり、*directory* は、CICS 領域の権限を付与する対象となるディレクトリーの絶対パスです。

例えば、`/usr/lpp/cicsts/cicsts56` ディレクトリーにグループ権限を割り当てるには、次のコマンドを使用します。

```
chgrp -R GID /usr/lpp/cicsts/cicsts56
```

CICS 領域のユーザー ID は ESM グループに接続されるため、CICS 領域は、これらすべてのディレクトリーおよびファイルに関する適切な権限を持つことになります。

### タスクの結果

これで、CICS には、Node.js アプリケーションを実行するために z/OS UNIX 内のディレクトリーとファイルにアクセスする適切な権限があることが確実にになりました。

ファイルの移動または新規ファイルの作成などにより、セットアップ中の CICS 機能を変更した場合は、新規のファイルまたは移動したファイルに関する CICS 領域のアクセス権限が保持されるように、この手順を忘れずに繰り返してください。

### 次のタスク

[Node.js ランタイムのインストールの検査](#)にあるサンプルを使用することにより、Node.js サポートが正しくセットアップされていることを確認します。

## Node.js のメモリ制限の設定

通常、Node.js アプリケーションには、コンパイル言語で作成されたものより多くのメモリが必要です。その理由の 1 つは、Node.js アプリケーションに、同時に多数のクライアント要求を処理する機能があるということです。Node.js アプリケーションを実行するために使用できる、十分なストレージとメモリが CICS および Node.js にあることを確認する必要があります。

### このタスクについて

Node.js アプリケーションに必要なストレージは、DSA、EDSA、GDSA などの CICS 管理対象ストレージ域からは割り振られません。代わりに、使用可能な MVS 専用域からのストレージが使用されます。24 ビット、31 ビット、64 ビットの各アドレッシング域に、十分な量の未割り振りの専用領域ストレージが存在することが重要です。専用領域ストレージが不足している場合、CICS はストレージ不足メカニズムを使用できません。

### 手順

1. z/OS **MEMLIMIT** パラメーターが適切な値に設定されていることを確認します。

このパラメーターは、CICS アドレス・スペースが使用できる 64 ビット・ストレージの量を制限します。

Node.js は 64 ビット・バージョンのストレージを使用するので、**MEMLIMIT** が、CICS 領域で 64 ビット・ストレージを使用するこの機能やその他の機能の両方に十分な大きい値に設定されていることを確認する必要があります。

2. 開始ジョブ・ストリームの **REGION** パラメーターに、Node.js の実行に十分な大きさの値が指定されていることを確認します。  
このパラメーターは、CICS アドレス・スペースが使用できる 31 ビット・ストレージの量を制限します。
3. 各 Node.js アプリケーションに必要な 24 ビット・ストレージが使用可能であることを確認してください。



## 第 5 章 Node.js のパフォーマンスの改善

Node.js アプリケーションのパフォーマンスを改善するために、さまざまなアクションを実行することができます。

### DFHSJNRO による NODEJSAPP のエンクレープの変更

DFHSJNRO は、Node.js アプリケーションが実行される Language Environment エンクレープにデフォルトの実行時オプション・セットを提供するサンプル・プログラムです。例えば、Node.js プロセスにストレージ割り振りパラメーターを定義します。すべてのワークロード用に最適化されたデフォルトのランタイム・オプションを提供することは不可能です。

#### このタスクについて

このサンプル・プログラムを更新して Language Environment エンクレープを調整するか、サンプルに基づいて独自のプログラムを作成することができます。

アセンブリ言語でこのプログラムを作成する必要があり、CICS 変換プログラムでこれを変換してはなりません。オプションは文字ストリングとして指定され、2 バイトのストリングの長さで、それに続く実行時オプションで構成されます。すべての Language Environment 実行時オプションの最大長は 255 バイトです。

#### 手順

1. DFHSJNRO プログラム・ソースを CICSTS56.CICS.SDFHSAMP ライブラリーから新しい場所にコピーします。  
CICS 領域に保守が適用される場合、プログラムの変更を反映したい場合があります。
2. 実行時オプションの省略形を使用して、各オプションを編集し、変更内容を全体で 200 バイト未満に制限します。  
「[z/OS Language Environment プログラミング・ガイド](#)」に、Language Environment 実行時オプションに関する詳細情報が記載されています。
  - CICS はいくつかのオプションをこのリストに追加するため、高速処理のためにはオプションのリストのサイズを最小限に保つ必要があります。
  - HEAP64 オプションを使用して、初期のヒープ割り振りを指定します。
  - ALL31 オプション、POSIX オプション、および XPLINK オプションは、CICS によって強制的にオンにされます。ABTERMENC オプションは、CICS によって (ABEND) に設定され、TRAP オプションは (ON,NOSPIE) に設定されます。
  - RPTO および RPTS オプションによって生成される出力は、CESE 一時データ・キューに書き込まれます。
  - 出力を生成するオプションがそれを行うのは、各 Node.js アプリケーションの終了時です。生成後に CESE に送信される出力のボリュームを考慮に入れてください。
3. DFHASMVS プロシージャを使用して、プログラムをコンパイルします。
4. コンパイルしたプログラムを CICS で使用可能なロード・ライブラリーにコピーします。
5. DFHSJNRO 以外の名前を使用する場合は、NODEJSAPP CICS バンドル・パーツを編集し、属性 `lerunopts="PROGRAM"` を指定します。

#### タスクの結果

NODEJSAPP が含まれる BUNDLE リソースを有効にすると、CICS は、DFHSJNRO プログラムに指定された実行時オプションを使用して Language Environment エンクレープを作成します。CICS は実行時オプションを Language Environment に渡す前に、それらの長さをチェックします。この長さが 255 バイトより長い場合、CICS は NODEJSAPP の始動を試みず、CSMT にエラー・メッセージを書き込みます。指定した値は、Language Environment に渡される前に CICS によってチェックされません。

## Node.js アプリケーションのストレージ要件の計算

CICS® 領域で 1 つ以上の Node.js アプリケーションを始動する場合は、各 Node.js アプリケーションで利用できるストレージ容量が十分であることを確認する必要があります。同じ領域で実行される CICS とその他の製品には、大量の z/OS ストレージが必要になる場合があります。**DSALIM** や **EDSALIM** などの CICS の割り振りパラメーターは、ストレージの可用性に影響を与えます。また、これらはピーク所要量と比較して過剰に割り振られることがあります。

### このタスクについて

Node.js アプリケーションに必要なストレージは、CICS DSA や EDSA ストレージから割り振られません。C コードで発行される **malloc()** などの要求を処理する Language Environment によって管理されるストレージもあれば、**IARV64** や **STORAGE OBTAIN** などの z/OS ストレージ管理要求を使用する Node.js アプリケーションによって直接管理されるストレージもあります。Language Environment は z/OS ストレージ・サービスを使用します。

Node.js ランタイムで使用される各スレッドには、4 KB の 24 ビット・ストレージが必要です。使用されるスレッド数は、Node.js ランタイムが開始されると固定され、**UV\_THREADPOOL\_SIZE** 環境変数を設定しないかぎり、通常は 8 から 12 の間です。さらに、UNIX System Services では、各スレッドの作成処理時に、連続する 24 ビット・ストレージが 256 KB 必要です。

Node.js ランタイムは、JavaScript オブジェクトおよび JIT コンパイル済みコードにヒープを割り振ります。z/OS 2.2 では、このヒープは 31 ビット・ストレージに割り振られます。z/OS 2.3 では、このヒープは 31 ビットおよび 64 ビット・ストレージの両方にわたります。また、ヒープは複数のスペースからなり、それぞれのサイズは個別に異なります。テスト環境でヒープ使用量を測定することが、ヒープ・サイズ所要量を見積もる最も単純な方法です。Node.js ランタイム用に UNIX System Services ダイナミック・リンク・ライブラリー (DLL) ファイルをロードするためには、31 ビット・ストレージも必要です。

さらに、C++ のヒープおよびスタック用に Language Environment により、64 ビット・ストレージが割り振られます。これは内部で Node.js ランタイム・コードにより、そしてネイティブ・モジュールによって使用されます。

Node.js アプリケーションに必要な総ストレージ量を見積もるには、次の手順を実行します。

### 手順

1. サンプル統計プログラム DFH0STAT を使用して、24 ビット、31 ビット、および 64 ビットのメモリー使用量を測定します。
  - **1** Private Area storage available below 16 Mb の値をメモします。これは、領域内で現在使用可能な 24 ビット専用ストレージです。
  - **2** Private Area storage available above 16 Mb の値をメモします。これは、領域内で現在使用可能な 31 ビット専用ストレージです。
  - **3** MEMLIMIT minus usable within Private Memory Objects の値をメモします。これは、領域内で現在使用可能な 64 ビット専用ストレージです。

#### Storage BELOW 16MB

```
-----
Private Area Region size below 16Mb . . . . . : 10,216K
Max LSQA/SWA storage allocated below 16Mb (SYS) . . : 660K
Max User storage allocated below 16Mb (VIRT). . . : 5,460K
System Use. . . . . : 20K
RTM . . . . . : 250K
```

```
-----
Private Area storage available below 16Mb . . . . . : 3,826K 1
```

#### Storage ABOVE 16MB

```
-----
Private Area Region size above 16Mb . . . . . : 1,417,216K
Max LSQA/SWA storage allocated above 16Mb (SYS) . . : 84,500K
Max User storage allocated above 16Mb (EXT) . . . : 987,936K
```

```
-----
Private Area storage available above 16Mb . . . . . : 344,780K 2
```

```

Storage ABOVE 2GB
-----
MEMLIMIT Size. . . . . : 200G
MEMLIMIT Set By. . . . . : JCL

Current Address Space active (bytes) : 3,780,116,480
Current Address Space active . . . . : 3,605M
Peak Address Space active. . . . . :
3,661M

MEMLIMIT minus Current Address Space active. . . . : 201,195M
MEMLIMIT minus usable within Private Memory Objects: 196,408M 3
Number of Private Memory Objects . . . . . : 728
....minus Current GDSA extents . . . . . : 727
Bytes allocated to Private Memory Objects. . . . . : 8,392M
....minus Current GDSA allocated . . . . . : 7,368M
Bytes hidden within Private Memory Objects . . . . : 4,787M
....minus Current GDSA hidden. . . . . : 4,786M
....minus CICS Internal Trace Table hidden . . : 3,794M
Bytes usable within Private Memory Objects . . . . : 3,605M
Peak bytes usable within Private Memory Objects. . : 3,681M
Current GDSA Allocated . . . . . : 1,024M
Peak GDSA Allocated. . . . . : 1,024M

```

2. NODEJSAPP を有効にして、代表的なワークロードを実行します。使用可能な各専用ストレージ域の値の変化を監視し、専用ストレージ域が制約を受けていないことを確認します。





## 第 6 章 Node.js アプリケーションのトラブルシューティング

Node.js アプリケーションに問題がある場合、CICS および Node.js で提供される診断を使用して、問題の原因を判別できます。

CICS は、Node.js に関連した問題の診断に役立ついくつかの統計、メッセージ、およびトレースを提供します。Node.js で提供される診断ツールおよびインターフェースにより、Node.js ランタイムおよびアプリケーションの実行に関する詳細情報を取得できます。

Node.js アプリケーションの分析をリアルタイムとオフラインで実行する無料のツール (例えば、IBM Health Center や Appmetrics) を使用できます。詳細については、[IBM モニターおよび診断ツール - ヘルプ・センター](#)または [Node アプリケーションのメトリック](#)を参照してください。

ログ・ファイルが存在する場所の詳細については、[NODEJSAPP 出力、ログおよびトレースの場所の制御](#)を参照してください。

**重要:** 問題の原因を確定できない場合は、IBM サポートにお問い合わせください。Node.js の問題を報告するための [MustGather](#) にリストされているとおりに、必要な情報を確実に提供してください。

IBM SDK for Node.js - z/OS に関連したトラブルシューティング情報については、[IBM SDK for Node.js - z/OS トラブルシューティング](#)を参照してください。

### インストール検査プログラム (IVP) の実行が失敗した場合:

1. MSGUSR ログを確認します。CICS バンドルおよび NODEJSAPP バンドル・パーツがインストールされ、使用可能になると、CICS メッセージがここに書き込まれます。
2. SYSPRINT ログを確認します。Node.js プロファイルが処理されると、CICS メッセージがここに書き込まれます。
3. WORK\_DIR/APPLID/DFHJNIVP/IVPSAMPLE ディレクトリーを確認します。Node.js ランタイム・メッセージおよびアプリケーション・メッセージは、CURRENT.STDOUT と CURRENT.STDERR の各ファイルに書き込まれます。CICS トレースが使用可能な場合は、CURRENT.TRACE に書き込まれます。

### npm のインストールで、Node.js アプリケーションの依存関係をダウンロードするために必要なサイトに到達できない場合

エラー `getaddrinfo ENOTFOUND nodejs.org nodejs.org:443` が表示される場合があります。

1. `npm -verbose install` によって返されたメッセージを調べて、サイト TCP/IP アドレスを識別するエラー (Error: connect ETIMEDOUT 2400:cb00:2048:1::6812:5e60:443 など) があるかを確認します。
2. サイトに代替 TCP/IP アドレスを使用してみてください。代替 TCP/IP IPv6 および IPv4 アドレスをリストするには、コマンド `dig registry.npmjs.org -t any` を使用します。TCP/IP アドレスを使用するように npm を変更するには、最初にコマンド `npm adduser --registry=https://<ipaddress>` を使用してから、コマンド `npm install` を再試行します。
3. ネットワーク・チームに連絡して、TCP/IP およびファイアウォールの構成を調べてください。

### NODEJSAPP がすぐに無効になる場合

`stderr` でメッセージ「CEE5207E The signal SIGABRT was received」を受け取った場合は、LPAR 上の共用メッセージ・キューの限度に達した可能性があります。SIGKILL シグナルによって Node.js アプリケーションが終了すると、共用メッセージ・キューは割り振り解除されない場合があります。これを回避するには、アプリケーションが SIGTERM シグナルへの応答としてタイムリーに終了するようする必要があります。詳しくは、[Node.js アプリケーションの開発](#)を参照してください。

z/OS コンソール・コマンド D OMVS,L を使用して共用メッセージ・キューの数を確認し、IPCMSGNIDS を検索することができます。共用メッセージ・キューを削除するには、`ipcrm` コマンドを使用します。詳しくは、[ipcrm - メッセージ・キュー、セマフォ・セット、または共用メモリー ID の除去](#)を参照してください。

次のようなメッセージを受け取った場合

- CEE0374C CONDITION=CEE3561S TOKEN=00030DE9 59C3C5C5 00000000\_00000001 WHILE RUNNING PROGRAM static-initial (CICS ジョブ・ログ)
- CEE3501S The module libnode.so was not found in stderr
- DFHSJ1313 E CICSUSER CNJL NODEJSAPP CICSJSON was disabled because an unsupported version of IBM SDK for Node.js - z/OS was used (MSGUSR)

CICS でサポートされている最小レベルの Node.js を使用しているかどうかを確認します。Node.js ランタイムのパスは、Node.js プロファイルの NODE\_HOME オプションで指定されます。

## Node.js アプリケーションのトレースのアクティブ化と管理

SJ コンポーネント・トレースをオンにすることにより、Node.js アプリケーションのトレースをアクティブにすることができます。少量のトレースが内部トレース・テーブルに書き込まれますが、Node.js はまた、ロギング情報を Node.js アプリケーションごとに zFS の固有のファイルに書き込みます。このファイルはラップしないので、そのサイズを zFS で管理する必要があります。

### このタスクについて

Node.js アプリケーションのトレースは、補助トレースも GTF トレースも使用しません。CICS は、内部トレース・テーブルに一部の情報を書き込みます。しかし、ほとんどの診断情報は Node.js によって記録され、zFS 内のファイルに書き込まれます。このファイルには、各 Node.js アプリケーションに由来する一意的な名前が付けられます。デフォルトのファイル名の形式は `&DATE;.&TIME;.trace` です。

NODEJSAPP リソースを使用可能にすると、このファイルが `$WORK_DIR/&APPLID;/&NODEJSAPP;` ディレクトリーに CICS によって作成されます。TRACE プロファイル・オプションを使用して、このトレース・ファイルの名前と場所を変更できます。プロファイルに対する変更は、NODEJSAPP リソースを有効にしたときに適用されます。Node.js アプリケーションの実行時にトレース・ファイルを削除または名前変更すると、CICS はそのファイルを再作成せず、ロギング情報が別のファイルに書き込まれることもありません。

### 手順

1. CETR トランザクションを使用して、Node.js アプリケーションのトレースを活動化します。Node.js アプリケーションを始動および停止するために CICS が行うアクションをトレースするには、SJ コンポーネントを選択します。Node.js は、診断情報を zFS ファイルに記録します。
2. SJ コンポーネントのトレース・レベルを設定します。
  - SJ のレベル 0 は、Node.js アプリケーションの初期化時のエラーなど、例外のみのトレースを作成します。SJ のレベル 1 とレベル 2 は、SJ ドメインからさらに CICS トレースを作成します。このトレースは、内部トレース・テーブルに書き込まれます。
  - SJ のレベル 3 は、Node.js から追加のログを作成します。例えば、警告メッセージや情報メッセージです。この情報は、zFS のトレース・ファイルに書き込まれます。
  - SJ のレベル 4、5 は、CICS および Node.js からデバッグ情報を作成します。これは、Node.js アプリケーション処理に関するより詳細な情報を提供します。この情報は、zFS のトレース・ファイルに書き込まれます。
3. 各トレース項目には、日時のタイム・スタンプがあります。[TRACE](#) プロファイル・オプションを使用して、このトレース・ファイルの名前と場所を変更できます。
4. デフォルトの TRACE 設定を使用している場合、NODEJSAPP リソースを有効にすると、CICS は Node.js アプリケーションの存続期間にわたって新しい固有のトレース・ファイルを作成します。  
NODEJSAPP リソースを無効にすると、トレース・ファイルを削除することができます。情報を別個に保持する場合はファイルの名前を変更することができます。

5. ファイルの数を管理するには、LOG\_FILES\_MAX オプションを設定することで、Node.js アプリケーション始動時に保持される古いトレース・ファイルの数を制御できます。

## Node.js アプリケーションの CICS コンポーネント・トレース

Node.js によって作成されるログに加えて、CICS は、SJ (JVM および Node.js ランタイム) ドメインで、トレース・レベル 0、1、および 2 において、いくつかの標準トレース・ポイントを提供します。これらのトレース・ポイントは、CICS が Node.js アプリケーションのセットアップと管理を行う際に取るアクションをトレースします。

CETR トランザクションを使用して、SJ ドメイン・トレース・ポイントをレベル 0、1 および 2 で活動化できます。SJ ドメインのすべての標準トレース・ポイントの詳細については、JVM および Node.js ランタイム・ドメインのトレース・ポイントを参照してください。

### SJ コンポーネントのトレース

SJ コンポーネントは、SJ ドメインでの例外および処理をトレースし、内部トレース・テーブルに書き込みます。SJ のレベル 3、レベル 4、およびレベル 5 のトレースは、Node.js ログを作成し、zFS のトレース・ファイルに書き込みます。トレース・ファイルの名前と場所は、Node.js プロファイル内の TRACE オプションによって決まります。トレース・ファイルのために十分なスペースが zFS 内にあることを確認してください。トレースの起動および管理について詳しくは、Node.js アプリケーションのトレースのアクティブ化と管理を参照してください。



## 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。この資料の他の言語版を IBM から入手できる場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。IBM 製品、プログラムまたはサービスに代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができません。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510

東京都中央区日本橋箱崎町 19 番 21 号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様自身の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

*IBM Director of Licensing*

*IBM Corporation*

*North Castle Drive, MD-NC119 Armonk,*

*NY 10504-1785*

*United States of America*

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関す

る実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態を提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

## プログラミング・インターフェース情報

CICS には、プログラミング・インターフェースと見なすことのできる資料と、プログラミング・インターフェースと見なすことのできない資料があります。

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが含まれています。

- [アプリケーションの開発](#)
- [Developing system programs](#)
- [CICS TS セキュリティ](#)
- [外部インターフェースに向けた開発](#)
- [アプリケーション開発のリファレンス](#)
- [リファレンス: システム・プログラミング](#)
- [リファレンス: 接続](#)

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が含まれています。

- [Troubleshooting and support](#)
- [CICS TS 診断リファレンス](#)

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが以下のマニュアルに含まれています。

- [アプリケーション・プログラミング・ガイドおよびアプリケーション・プログラミング・リファレンス](#)
- [Business Transaction Services](#)
- [Customization Guide](#)
- [C++ OO Class Libraries](#)
- [Debugging Tools Interfaces Reference](#)
- [Distributed Transaction Programming Guide](#)
- [External Interfaces Guide](#)
- [Front End Programming Interface Guide](#)



- IMS Database Control Guide
- インストール・ガイド
- セキュリティー・ガイド
- Supplied Transactions
- CICSplex® SM Managing Workloads
- CICSplex SM Managing Resource Usage
- CICSplex SM アプリケーション・プログラミング・ガイドおよび CICSplex SM アプリケーション・プログラミング・リファレンス
- Java Applications in CICS

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が以下のマニュアルに含まれています。

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSplex SM Problem Determination Guide

## 商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、<http://www.ibm.com/legal/copytrade.shtml> をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Apache、Apache Axis2、Apache Maven、Apache Ivy、Apache Software Foundation (ASF) ロゴ、および ASF feather ロゴは、Apache Software Foundation の商標です。

Gradle および Gradlephant ロゴは、Gradle, Inc. およびその子会社の米国およびその他の国における登録商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Intel Centrino、Intel Centrino ロゴ、Celeron、Intel Xeon、Intel SpeedStep、Itanium、および Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

登録商標 Linux® は、世界中で商標の所有者である Linux Torvalds の独占的ライセンシーである Linux Foundation のサブライセンスに従って使用されています。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Red Hat®、および Hibernate® は、Red Hat, Inc. またはその子会社の米国およびその他の国における商標または登録商標です。

Spring Boot は、Pivotal Software, Inc. の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Zowe™、Zowe ロゴ、および Open Mainframe Project™ は、Linux Foundation の商標です。

## 製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

## 適用範囲

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

## 個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布（頒布、送信を含む）または表示（上映を含む）することはできません。

## 商用使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

## 権利

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

## IBM オンラインでのプライバシー・ステートメント

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品（ソフトウェア・オフファリング）では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オフファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オフファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オフファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

### CICSplex SM Web ユーザー・インターフェース（メイン・インターフェース）の場合:

このソフトウェア・オフファリングは、展開される構成に応じて、セッション管理、認証、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの Cookie および持続的な Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

### CICSplex SM Web ユーザー・インターフェース（データ・インターフェース）の場合:

このソフトウェア・オフファリングは、展開される構成に応じて、セッション管理、認証、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名またはその他の個人情報を、セッションごとの Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

### CICSplex SM Web ユーザー・インターフェース（「Hello World」ページ）の場合:

このソフトウェア・オフファリングは、展開される構成に応じて、個人情報を収集しないセッションごとの Cookie を使用する場合があります。これらの Cookie を無効にすることはできません。

### CICS Explorer の場合:

このソフトウェア・オフファリングは、展開される構成に応じて、セッション管理、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの設定および持続的な設定を使用して収集する場合があります。これらの設定を無効にすることはできませんが、ユーザー・パスワードの暗号化形式でのディスクへの保管は、サインオン中にチェック・ボックスにチェック・マークを付けることによるユーザーの明示的な操作によってのみ有効化することができます。



この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、[『IBM プライバシー・ポリシー』](#) および [『IBM オンラインでのプライバシー・ステートメント』](#) の『クッキー、Web ビーコン、その他のテクノロジー』および [『IBM ソフトウェア製品および Software as a Service のプライバシー・ステートメント』](#) を参照してください。



---

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。  
なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]

アクセス制御リスト (ACL) [19](#)  
エンクレーブの変更  
JVM サーバー [23](#)

## [カ行]

グループ ID (GID) [19](#)

## [サ行]

時間帯  
記号 [18](#)

## [タ行]

調整  
Java [23](#)

## [ハ行]

パフォーマンス  
Java [23](#)

## [ヤ行]

ユーザー ID (UID) [19](#)

## D

DFHAXRO [23](#)

## G

GID [19](#)

## J

Java  
パフォーマンス [23](#)  
JVM サーバー  
エンクレーブの変更 [23](#)

## L

Language Environment エンクレーブ  
JVM サーバー [23](#)

## N

Node.js

Node.js (続き)  
インストールの検査 [12](#)  
Node.js のオプション  
記号 [17](#)  
Node.js プロファイル  
規則 [13](#)  
Node.js プロファイルに対する規則 [13](#)

## T

TZ [18](#)

## U

UID [19](#)  
UNIX システム・サービスのアクセス [19](#)  
UNIX ファイル・アクセス [19](#)





