

z/OS용 CICS Transaction
Server버전 5 릴리스 6

CICS에서 Node.js 사용



참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, [제품 법적 주의사항](#)에 있는 정보를 확인하십시오.

이 개정판은 새 개정판에 별도로 명시하지 않는 한 IBM® CICS® Transaction Server for z/OS®, 버전 5 릴리스 6 (제품 번호 5655-Y305655-BTA) 및 모든 후속 릴리스와 수정에 적용됩니다.

© Copyright International Business Machines Corporation 1974, 2020.

목차

| | |
|---|-----------|
| 이 PDF 정보..... | v |
| 제 1 장 CICS 및 Node.js..... | 1 |
| Node.js 런타임 환경..... | 2 |
| Node.js 및 CICS 번들..... | 2 |
| NODEJSAPP 번들 파트의 라이프사이클..... | 3 |
| 제 2 장 Node.js 애플리케이션 개발..... | 5 |
| Node.js 애플리케이션 개발을 위한 우수 사례..... | 5 |
| Node.js 애플리케이션에 사용할 환경 변수..... | 6 |
| CICS 서비스 호출..... | 7 |
| Node.js 파이프라인 고려사항..... | 9 |
| 제 3 장 Node.js 애플리케이션 배치..... | 11 |
| Node.js 런타임 설치 확인..... | 11 |
| 제 4 장 Node.js 지원 설정..... | 13 |
| Node.js 프로파일 유효성 검증 및 특성..... | 13 |
| Node.js 프로파일을 코딩하는 규칙..... | 13 |
| Node.js 프로파일 및 명령행 옵션..... | 14 |
| Node.js 프로파일에서 사용된 기호..... | 17 |
| Node.js 애플리케이션의 시간대 설정..... | 17 |
| NODEJSAPP 출력, 로그 및 추적 위치 제어..... | 18 |
| CICS 리전에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공..... | 18 |
| Node.js의 메모리 한계 설정..... | 20 |
| 제 5 장 Node.js 성능 개선..... | 21 |
| DFHJSNRO로 NODEJSAPP의 인클레이브 수정..... | 21 |
| Node.js 애플리케이션에 대한 저장영역 요구사항 계산..... | 21 |
| 제 6 장 Node.js 애플리케이션 문제점 해결..... | 25 |
| Node.js 애플리케이션에 대한 추적 활성화 및 관리..... | 26 |
| Node.js 애플리케이션에 대한 CICS 컴포넌트 추적..... | 26 |
| 주의사항..... | 29 |
| 색인..... | 33 |

이 PDF 정보

이 PDF는 CICS에서 실행하는 애플리케이션에서 Node.js를 개발하고 사용하는 방법을 설명합니다. CICS에 대한 경험이 거의 없는 숙련된 Node.js 애플리케이션 프로그래머를 위한 것으로, Node.js 프로그램을 개발하고 실행하는 데 필요한 정도의 CICS 정보만 제공합니다. Node.js 지원에 대한 CICS 요구사항을 파악해야 하는 숙련된 CICS 사용자 및 시스템 프로그래머에 필요한 정보도 제공합니다.

사용된 용어 및 표기법에 대한 세부사항은 IBM Knowledge Center에서 [CICS 문서에서 사용하는 규칙과 용어](#)의 내용을 참조하십시오.

이 PDF의 날짜

이 PDF는 2020년 5월 28일에 작성되었습니다.

제 1 장 CICS 및 Node.js

Node.js는 JavaScript에서 작성된 애플리케이션의 서버 측 런타임입니다.

특성은 다음과 같습니다.

- 이벤트 기반 - HTTP 요청과 같은 이벤트를 청취하고 이벤트가 발견될 때 콜백 기능을 트리거합니다.
- 단일 스레드 - 한 번에 하나의 요청을 처리합니다.
- 비블록 I/O - I/O 디바이스(예: 파일 시스템, 소켓 및 데이터베이스)에 대한 읽기 및 쓰기는 z/OS에서의 기본 지원을 사용하여 비동기식으로 발생하며 콜백 기능이 완료될 때 이를 트리거합니다.

Node.js는 경량이고 효율적이며 데이터 중심 애플리케이션에 가장 적합합니다. 이는 z/OS에서 기본 비동기 I/O 지원을 사용할 수 있으며 Agile 사례를 권장하는 애플리케이션 디자인 및 개발에 대한 확장성이 높은 모듈 기반의 접근 방법을 제공합니다.

이는 지속적으로 엔터프라이즈 내의 입지를 굳히고 있으며, REST 서비스를 제공하고 집계하는 기능으로 인해 디지털 변환에 대해 선호되는 선택사항이 되고 있습니다.

Node.js가 인기를 끄는 주요 요인은 공용 서비스 레지스트리에서 사용이 가능하며 NPM(Node Package Manager)을 사용하여 액세스되는 풍부한 Node.js 모듈에서 찾을 수 있습니다. 모듈은 이미 대부분의 태스크에 사용 가능하며 Node.js 애플리케이션 개발자를 위해 상당한 시간이 절약됩니다.

Node.js는 Node.js 및 관련 모듈의 채택과 개발을 촉진하는 목표를 갖고 있는 Node.js Foundation에서 개발됩니다. 자세한 정보는 [Node.js Foundation 웹 사이트](#)를 참조하십시오.

Node.js 애플리케이션에서 CICS 서비스 호출

Node.js 애플리케이션은 일반적으로 장시간 실행되며 많은 사용자로부터의 TCP/IP 소켓 요청을 처리합니다. Node.js 런타임은 각 애플리케이션마다 시작됩니다. CICS 리전에 여러 애플리케이션이 존재할 수 있습니다.

CICS에서 실행되는 Node.js 애플리케이션은 기존 CICS 애플리케이션을 호출해야 할 수 있습니다. 예를 들어, Node.js 애플리케이션은 프론트 엔드 애플리케이션에 대한 단일 서비스 인터페이스를 제공하기 위해 기존 비즈니스 로직 기능에 대한 호출을 집계할 수 있습니다. 기존 비즈니스 로직 기능을 사용하면 기존 애플리케이션에 대한 Node.js 애플리케이션의 근접성을 최대한 활용하므로 프론트 엔드 애플리케이션이 다수의 네트워크 호출을 작성할 필요가 없습니다. Node.js 애플리케이션은 외부 서비스를 호출하거나 NPM 모듈을 사용하여 기존 비즈니스 로직에 기능을 추가할 수도 있습니다.

Node.js 애플리케이션은 기존 비즈니스 로직을 호출하기 위해 CICS에 호스팅된 서비스를 호출할 수 있습니다. 이는 CICS 웹 서비스 기술을 사용하거나 z/OS Connect를 사용하여 공개된 JSON 또는 SOAP 웹 서비스일 수 있습니다. Node.js 애플리케이션은 JSON 및 SOAP 웹 서비스를 이용하고 HTTP 요청을 작성하기 위해 사용되는 NPM 모듈을 사용하여 CICS 서비스를 호출할 수 있습니다. JSON이 JavaScript의 고유 오브젝트 형식이므로, JSON 웹 서비스는 Node.js 애플리케이션에서 이용하기가 매우 쉽습니다.

또는 Node.js 애플리케이션이 호출이 필요한 JSON 웹 서비스와 동일한 CICS 리전에서 호스팅된 경우에는 로컬로 최적화된 전송이 사용될 수 있습니다. 이는 교차 메모리(cross-memory) 접근 방법을 사용하여 서비스를 호출하며, 네트워크 상에서 상호작용이 필요 없습니다. 로컬로 최적화된 전송을 사용하여 CICS 서비스를 호출하려면, Node.js 애플리케이션이 `ibm-cics-api` 모듈을 사용해야 합니다. 서비스는 CICS JSON 웹 서비스 기술을 사용하여 공개되어야 하며, 적합한 PIPELINE 및 URIMAP 자원이 존재해야 합니다. 자세한 정보는 [CICS 서비스 호출](#)의 내용을 참조하십시오.

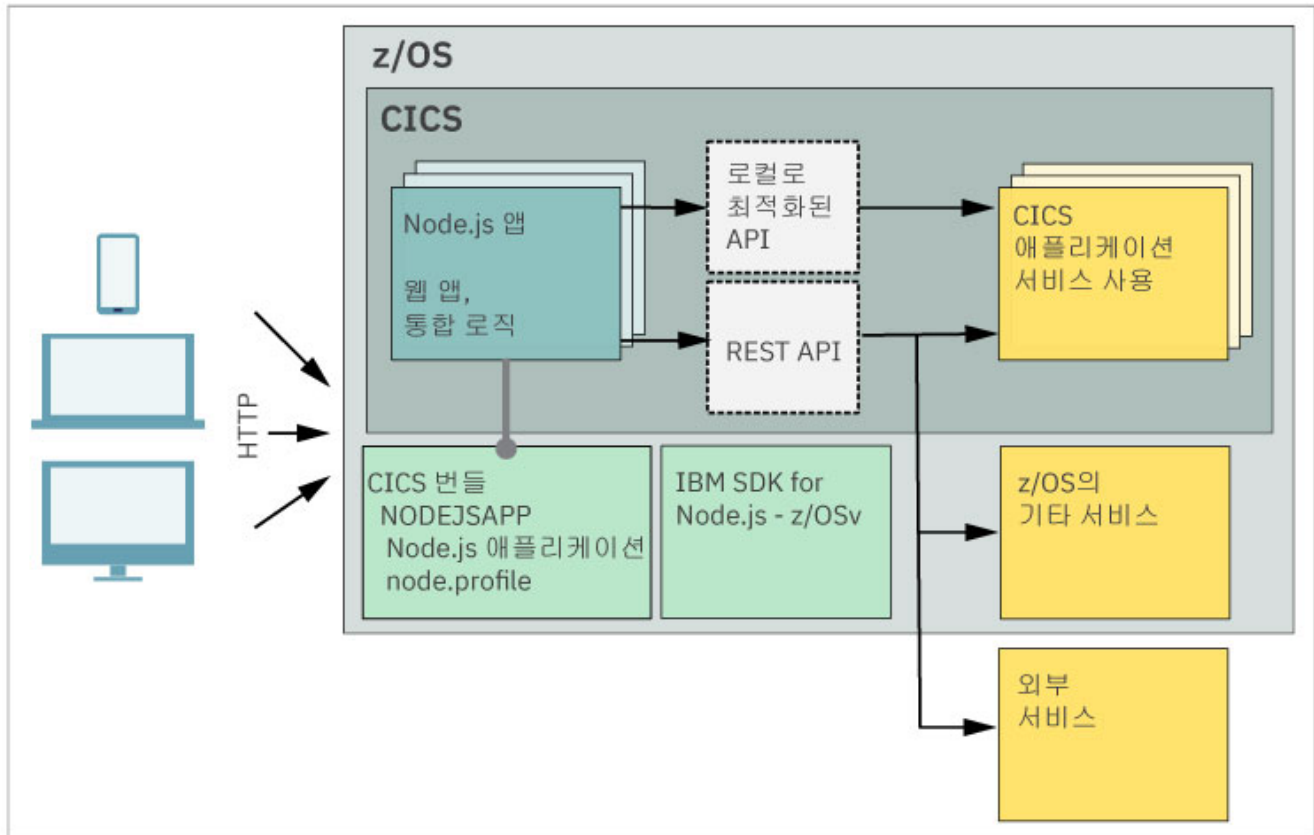


그림 1. Node.js 애플리케이션에 대한 CICS 지원

Node.js 런타임 환경

CICS와 함께 Node.js 애플리케이션을 사용하려면 Node.js -z/OS용 IBM SDK를 설치해야 합니다. CICS V5.6에서 Node.js 애플리케이션으로 사용되는 Node.js 런타임을 제공합니다.

시스템 요구사항에 대한 자세한 정보는 [자세한 시스템 요구사항](#)의 내용을 참조하십시오.

CICS의 Node.js 런타임 환경은 CICS 리전 사용자 ID 하에서 실행되며, 이에는 LE(Language Environment®) 인클레이브가 포함됩니다. 인클레이브는 Node.js 프로세스를 실행하고 이러한 단일 인클레이브는 각 Node.js 애플리케이션에 존재합니다. Node.js 애플리케이션의 워크로드를 인클레이브 내에서 실행되며 다른 Node.js 애플리케이션 인스턴스로부터 격리됩니다.

CICS의 Node.js 애플리케이션은 NODEJSAPP 자원으로 표시됩니다. Node.js 애플리케이션(예: Node.js -z/OS용 IBM SDK의 설치 위치)에 필요한 구성 정보는 Node.js 애플리케이션 프로파일에 지정됩니다.

이 애플리케이션 프로파일은 Node.js 애플리케이션을 구성하는 기타 아티팩트와 함께 CICS 번들에 함께 패키징되어야 합니다. CICS BUNDLE 자원은 Node.js 애플리케이션을 CICS에 표시하고 이를 사용하여 애플리케이션의 라이프사이클을 관리할 수 있습니다. 번들이 사용으로 설정되면 연관된 Node.js 애플리케이션은 인클레이브 내에서 실행됩니다. 번들이 사용 안함으로 설정되면 연관된 Node.js 애플리케이션은 중지됩니다.

Node.js 및 CICS 번들

Node.js 애플리케이션은 JavaScript 소스 코드 및 지원 파일로 구성됩니다. Node.js 애플리케이션은 CICS NODEJSAPP 번들 파트를 사용하여 CICS 번들에 배치됩니다.

NODEJSAPP 번들 파트는 JavaScript 코드를 참조하여 모두 CICS 번들에 패키징된 애플리케이션 및 Node.js 애플리케이션 프로파일을 시작합니다. Node.js 애플리케이션에 종속성이 있는 경우 이 종속성은 번들이 사용으로 설정되기 전에 NPM을 사용하여 분석되어야 합니다.

CICS 번들이 사용으로 설정되는 경우 CICS는 Node.js 애플리케이션 프로파일의 구성을 사용하여 환경을 설정하고 Node.js -z/OS용 IBM SDK에서 제공된 Node.js 런타임을 시작합니다. Node.js 런타임은 애플리케이션을 시작합니다.

CICS 번들이 사용으로 설정되지 않는 경우 CICS는 Node.js 애플리케이션을 SIGTERM 신호로 전송하여 이를 중지합니다. 그런 다음 Node.js 런타임이 제거됩니다.

Node.js 애플리케이션 배치에 대한 자세한 정보는 [Node.js 애플리케이션 배치](#)의 내용을 참조하십시오.

NODEJSAPP 번들 파트의 라이프사이클

CICS에서 NODEJSAPP 번들 파트는 Node.js 애플리케이션의 라이프사이클을 관리합니다. 각 NODEJSAPP 번들 파트는 단일 Node.js 애플리케이션과 Node.js 런타임의 단일 인스턴스를 관리합니다.

NODEJSAPP 번들 파트 사용

CICS Explorer® 또는 CEMT SET BUNDLE 명령을 사용하여 NODEJSAPP 번들 파트가 포함된 CICS BUNDLE을 사용으로 설정할 수 있습니다.

Node.js 애플리케이션 번들 파트가 사용으로 설정되는 경우 CICS는 Node.js 런타임을 시작하고 NODEJSAPP으로 지정된 초기 JavaScript 파일을 호출합니다. 그런 다음 CICS는 번들 파트의 상태를 ENABLED로 설정합니다. CICS는 애플리케이션이 작업을 허용할 준비가 되는 시점을 알고 있지 않습니다.

Node.js 런타임 시작 시 오류(예: Node.js 프로파일 또는 애플리케이션의 오류)가 발생하는 경우 CICS는 번들 파트의 상태를 DISABLED로 설정합니다.

NODEJSAPP 번들 파트 사용 안함

Node.js 애플리케이션은 다음 두 가지 방법으로 사용 안함으로 설정됩니다.

- 애플리케이션은 정상적으로 완료되고 CICS는 번들 파트의 상태를 DISABLED로 설정합니다.
- CICS 번들이 사용 안함으로 설정되었으므로 번들 파트는 DISABLING 상태로 설정됩니다.

NODEJSAPP 번들 파트가 DISABLING 상태로 설정되는 경우 CICS는 SIGTERM 신호를 Node.js 프로세스로 전송합니다. 이를 통해 [Node.js 애플리케이션 개발](#)에 설명된 대로 애플리케이션이 신호를 수신하고 정상적으로 종료됩니다. 설정된 기간(NODEJSAPP_DISABLE_TIMEOUT Node.js 프로파일 옵션으로 제어됨) 이후에 애플리케이션이 자체적으로 종료되지 않으면 CICS는 SIGKILL 신호를 Node.js 프로세스로 전송합니다. 이는 Node.js 런타임이 즉시 종료되도록 강제 실행합니다. 그러나 시스템 자원(예: Unix 메시지 큐)은 종료 시에 할당 해제되지 않을 수 있습니다. 애플리케이션은 SIGTERM 알림에 반응하도록 요청을 받으며, 그러지 못하면 시간이 지나면서 시스템 자원이 고갈될 수 있습니다.

일단 Node.js 런타임이 종료되면, CICS는 로컬로 최적화된 API를 사용하여 호출 요청에 의해 시작된 모든 태스크가 종료될 때까지 대기합니다. 그리고 NODEJSAPP 번들 파트는 DISABLED 상태에 진입합니다.

제 2 장 Node.js 애플리케이션 개발

Node.js 애플리케이션은 JavaScript를 사용하여 작성되며 COBOL 및 Java™ 프로그래머에 생소할 수 있는 비동기 프로그래밍 개념을 포함합니다. 경험이 풍부한 Node.js 개발자는 CICS용 애플리케이션 개발이 다른 플랫폼용 Node.js 애플리케이션 개발과 유사한 프로세스라는 것을 압니다. 개발 및 디버깅 스킬이 공유되고 노드 패키지 관리자(NPM)가 동일한 방식으로 사용됩니다.

CICS는 Node.js 애플리케이션에서 CICS 프로그램을 호출하기 위한 API를 제공합니다. API는 기존 CICS 자산을 네트워크를 통해 서비스로 호출하지 않고 상호작용할 수 있도록 로컬로 최적화된 방법을 제공합니다.

주제 5 페이지의 『Node.js 애플리케이션 개발을 위한 우수 사례』에서는 Node.js 애플리케이션을 개발할 때 인식해야 하는 애플리케이션 프로그래밍 측면을 다룹니다.

[Github](#)의 지시사항에 따라 z/OS용 샘플 Node.js 애플리케이션을 다운로드할 수 있습니다.

CICS에서 Node.js의 제한사항

CICS 내에서 거의 모든 공통 Node.js 라이브러리를 사용할 수 있지만 몇 가지 제한사항이 있습니다. 해당 제한사항에는 원시 코드와 기본 운영 체제와의 상호작용이 모두 포함됩니다. 써드파티 API의 구현에 플랫폼 특정 고유 코드가 포함된 경우 z/OS로 포팅되지 않았을 수 있습니다. z/OS로 포팅되지 않은 것을 사용하려면 해당 코드 작성자의 지원을 받아야 할 수 있습니다. API가 새로운 운영 체제 프로세스를 생성해야 하는 경우, CICS 내에서 해당 기능을 사용할 수 없습니다. 다음 Node.js API는 이러한 이유로 CICS와 호환되지 않는 것으로 알려져 있습니다.

- [하위 프로세스](#)
- [클러스터](#)
- [process.setegid\(id\)](#)
- [process.seteuid\(id\)](#)
- [process.setgid\(id\)](#)
- [process.setgroups\(groups\)](#)
- [process.setuid\(id\)](#)

Node.js 애플리케이션 개발을 위한 우수 사례

Node.js 애플리케이션을 개발할 때 자원의 구성을 구체화하도록 환경 변수 사용을 고려해야 합니다. 또한 정상적으로 종료될 수 있도록 Node.js 애플리케이션에서 SIGTERM 신호를 처리하는 것도 고려해야 합니다.

환경 변수 사용

Node.js 애플리케이션이 TCP/IP 포트 또는 URI 또는 데이터베이스와 같은 자원에 액세스하는 경우 환경 변수를 사용하여 해당 자원의 구성을 구체화하는 것이 좋습니다. 이를 통해 개발, 테스트 및 프로덕션 환경에서 애플리케이션을 배치할 때 다른 값을 지정할 수 있습니다.

예를 들어, HTTP 요청을 청취하기 위한 TCP/IP 포트를 CICS Node.js 애플리케이션 프로파일에서 지정할 수 있습니다.

```
PORT=8080
```

Node.js 애플리케이션은 [process.env](#) 특성을 사용하여 값을 가져올 수 있습니다.

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

정상적으로 종료

Node.js 애플리케이션을 포함하는 CICS BUNDLE이 DISABLED로 설정된 경우 CICS는 신호 SIGTERM을 Node.js 프로세스로 보냅니다. 이를 통해 Node.js 애플리케이션이 정상적으로 종료될 수 있습니다. 예를 들어,

더 이상 새 연결을 허용하지 않고, 지속적 연결을 중지하고, 미해결 요청을 완료하고, 마지막으로 파일과 데이터 베이스에 대한 연결을 닫고 종료합니다. The 옵션 `NODEJSAPP_DISABLE_TIMEOUT`으로 지정된 기간 내에 애플리케이션을 종료해야 합니다.

다음은 SIGTERM 신호를 처리하고 HTTP 서버를 종료하는 예입니다.

```
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);
```

이러한 기술 모두의 예는 Node.js IVP 샘플에서 설명됩니다. 자세한 정보는 [Node.js 런타임 설치 확인](#)의 내용을 참조하십시오.

Node.js 애플리케이션에 사용할 환경 변수

Node.js 애플리케이션은 환경 변수를 사용하여 CICS 번들 및 환경에 대한 정보를 찾을 수 있습니다.

CICS_APPLID

CICS 리전 애플리케이션 ID `APPLID` SIT 매개변수의 값.

CICS_BUNDLE

Node.js 애플리케이션이 포함된 CICS 번들을 관리하는 데 사용되는 `BUNDLE` 자원의 이름.

CICS_BUNDLEID

Node.js 애플리케이션이 포함된 CICS 번들의 ID.

CICS_LOCAL_CCSID

`LOCALCCSID` SIT 매개변수의 값.

CICS_LOG

CICS가 Node.js 애플리케이션의 작동중에 로그 메시지를 작성하는 z/OS UNIX 파일의 이름.

CICS_NODEJSAPP

CICS 번들 내 `NODEJSAPP`의 이름.

CICS_OUTPUT_DIR

`CICS_WORK_DIR`, 즉 `<APPLID>/<BUNDLEID>/<NODEJSAPP>`에 관련된 `STDOUT`, `STDERR`, `LOG` 및 `TRACE` 파일이 포함되는 디렉토리.

CICS_PROFILE_PATH

Node.js 애플리케이션 프로파일에 대한 완전한 파일 이름.

CICS_STDERR

Node.js 표준 오류(`STDERR`)가 기록되는 완전한 파일 이름.

CICS_STDOUT

Node.js 표준 출력(`STDOUT`)이 기록되는 완전한 파일 이름.

CICS_TRACE

Node.js 추적(TRACE)이 기록되는 완전한 파일 이름.

CICS_USSCONFIG

UNIX 시스템 서비스 구성 USSCONFIG SIT 매개변수의 값.

CICS_USSHOME

UNIX 시스템 서비스 홈 USSHOME SIT 매개변수의 값.

CICS_WORK_DIR

Node.js 애플리케이션의 작업 디렉토리.

사용법 예

환경 변수는 `process.env` 글로벌 변수를 사용하여 Node.js 애플리케이션에서 액세스될 수 있습니다. 예를 들면 다음과 같습니다.

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

관련 링크

[Node.js 프로파일 및 명령행 옵션](#)

CICS 서비스 호출

`ibm-cics-api` 모듈에서 호출 함수를 사용하여 CICS 서비스를 호출할 수 있습니다. Node.js에는 비동기 활동을 위한 인기있는 두 가지 메커니즘, 콜백 함수 및 약속이 있습니다. 어느 기술이든 사용할 수 있습니다.

Node.js 애플리케이션이 CICS 내에서 실행 중인 경우 호출 요청은 CICS가 대상 JSON 서비스를 실행하기 위한 새 태스크를 시작하게 합니다. 애플리케이션과 CICS 간에 JSON 형식으로 데이터가 전달되면 응답이 애플리케이션으로 리턴됩니다. 다음 예제에는 공통 시작이 있으며 CICS 서비스를 호출하는 두 가지 옵션이 있습니다. 콜백 함수를 사용하거나 약속을 사용할 수 있습니다.

```
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 서비스를 호출하는 데 사용되는 세 개의 매개변수가 있습니다.

- URI(문자열) - 대상 서비스를 식별합니다. CICS에서 URIMAP를 일치시키는 데 사용되며, 그런 다음 WEBSERVICE, PIPELINE, 최종적으로 대상 PROGRAM에 매핑됩니다.
- 요청 데이터(JavaScript 오브젝트 또는 문자열) - 요청 본문으로서 CICS 서비스에 전송되는 데이터입니다. 이 데이터는 CICS의 해당 서비스에 의해 예상된 JSON 형식이어야 합니다.
- 콜백 함수 - 응답이 Node.js 애플리케이션에 의해 처리할 준비가 되면 CICS에 의해 이 함수가 호출됩니다. 약속을 사용하는 경우 이 함수는 적용되지 않습니다.

콜백 함수에 전달되는 매개변수는 다음과 같습니다.

- 오류 오브젝트. 요청이 성공하면 null입니다. 실패하는 경우, 이는 오류를 설명하는 JavaScript 오류 오브젝트입니다.
- 응답 오브젝트. CICS의 응답을 표시하는 JavaScript 오브젝트입니다.

오류 처리

Node.js 애플리케이션에서 요청을 처리하는 동안 오류가 발생하면 CICS는 JavaScript 오류 오브젝트를 작성하여 콜백 함수에 전달합니다. `httpCode`라는 정수 값은 동등한 HTTP 상태의 오류를 표시합니다. 다음 예제에서는 `stderr`로 인쇄될 때 오류가 발생했습니다.

```
{ 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."}}}' }
```

| 코드 | 메시지 | httpCode | 설명 |
|-------------------------|----------------------|----------|--|
| ERR_CICS_BAD_REQUEST | CICS 오류: 잘못된 요청 | 400 | 애플리케이션에서 CICS에 전달된 데이터의 오류. 일반 오류에는 잘못된 형식이나 올바르지 않은 URI가 포함됩니다. |
| ERR_CICS_FORBIDDEN | CICS 오류: 금지 | 403 | 권한부여 오류. 일반적으로 URIMAP와 연관된 사용자 ID에는 대상 TRANSACTION을 첨부할 권한이 없습니다. |
| ERR_CICS_NOT_FOUND | CICS 오류: 자원을 찾을 수 없음 | 404 | 구성 오류. 처리에 관련된 CICS 자원 중 하나를 찾을 수 없습니다. 누락된 자원은 URIMAP, TRANSACTION 또는 PIPELINE 자원일 수 있습니다. 가장 일반적인 문제점은 CICS에 알려지지 않은 URI를 사용하는 것입니다. |
| ERR_CICS_INTERNAL_ERROR | CICS 오류: 내부 서버 오류 | 500 | CICS 내에서 문제가 발생했습니다. 일반적인 문제로는 대상 서비스가 이상 종료되었거나 데이터 변환 오류가 발생한 경우를 들 수 있습니다. CICS는 문제점을 설명하기 위해 진단을 발행하며 오류 오브젝트의 '응답' 속성에서 설명 메시지를 사용할 수 있습니다. 가장 일반적인 문제점에는 JSON 요청 데이터와 CICS 내에서 예상되는 JSON 형식 간의 불일치가 포함됩니다. |
| ERR_CICS_UNAVAILABLE | CICS 오류: 사용 불가능 | 503 | CICS의 자원이 사용 안함으로 설정되었습니다. URIMAP, TRANSACTION 또는 PIPELINE 자원 중 하나를 현재 사용할 수 없습니다. |

단위 테스트

호출 함수는 로컬 및 원격 CICS 호출을 지원합니다. Node.js 애플리케이션이 CICS에서 실행될 때 요청은 네트워크를 사용하지 않는 교차 메모리 시스템 호출로 최적화됩니다. Node.js 애플리케이션이 CICS 외부에서 실행될 때 요청은 URI를 기반으로 HTTP 또는 HTTPS를 사용하여 CICS로 전송됩니다. 이 원격 기능은 단위 테스트 용도로만 사용됩니다.

ibm-cics-api 모듈을 사용하여 HTTP를 통해 CICS에 연결하는 경우 기본 인증과 같은 HTTP 헤더를 설정하거나 클라이언트 인증서를 보낼 수 없습니다. 이 작업을 수행해야 하는 경우 다른 Node.js 모듈을 대신 사용하십시오.

Node.js 파이프라인 고려사항

Node.js 애플리케이션은 호출 함수를 사용하여 CICS의 JSON 파이프라인을 시작하여 JSON 웹 서비스로 사용할 수 있는 기존 CICS 프로그램을 호출할 수 있습니다. Node.js 애플리케이션에서 기존 JSON 파이프라인을 재사용할 수 있으며 다음과 같은 고려사항이 있습니다.

파이프라인 선택

로컬로 최적화된 Node.js 애플리케이션의 호출 함수는 HTTP 프로토콜을 통해 CICS에 도달하지 않습니다. HTTP 프로토콜에 종속적인 모든 파이프라인은 Node.js와 함께 사용하기에 적합하지 않을 수 있습니다. 예를 들어, 파이프라인에 등록된 핸들러 프로그램이 EXEC CICS WEB API를 사용하여 HTTP 요청의 헤더와 상호작용하려고 시도하면 CICS에서 오류 응답이 수신됩니다. 호출 함수에 대한 자세한 정보는 [CICS 서비스 호출](#)의 내용을 참조하십시오.

파이프라인이 파이프라인 전송 핸들러 프로그램을 사용하여 특정 전송 구현에 바인드된 경우 해당 파이프라인은 Node.js 애플리케이션에서 로컬로 최적화된 호출 함수와 함께 사용하기에 적합하지 않을 수 있습니다. CICS는 호출 함수에 지정된 URI를 URIMAP 자원과 일치시키는 것을 기반으로 Node.js 요청에 사용할 파이프라인을 선택합니다. 연관된 URIMAP는 USAGE(PIPELINE)를 표시해야 합니다. USAGE(PIPELINE)가 표시되지 않으면 URIMAP는 Node.js 애플리케이션에서 로컬로 최적화된 호출 함수와 함께 사용하기에 적합하지 않습니다.

URIMAP가 특정 TCIPSERVICE 자원에 바인드된 경우 해당 URIMAP는 해당 TCIPSERVICE의 작업만 허용합니다. Node.js 애플리케이션에서 로컬로 최적화된 호출 함수는 HTTP를 사용하지 않으며 TCIPSERVICE 자원을 무시합니다. 이러한 URIMAP는 Node.js 애플리케이션 호출 함수에서 로컬로 최적화된 요청과 함께 사용하기에 적합하지 않습니다.

파이프라인 핸들러 프로그램

CICS는 몇몇 유형의 JSON 파이프라인을 지원합니다. 자세한 정보는 [JSON 요청에 대한 서비스 제공자로서의 CICS](#)의 내용을 참조하십시오. Node.js 애플리케이션에서 로컬로 최적화된 호출 함수와 함께 사용하기에 적합한 두 가지 구성이 있습니다. 이는 CICS Java 파이프라인 및 CICS 제공 터미널 핸들러 DFHPIJT라고 합니다. CICS용 z/OS Connect 또는 CICS Liberty JVM 서버의 JAX-RS 및 JSON 기능을 사용하여 구현된 JSON 인프라는 사용할 수 없습니다. DFHPIJT 터미널 핸들러(비Java JSON 서비스 제공자라고도 함)는 Node.js에 대해 최고의 성능 특성을 제공할 수 있습니다. 이러한 옵션 중 어느 것도 컨텍스트 전환이라고 하는 핸들러 프로그램 내에서 대량 작업의 사용자 및 트랜잭션 ID를 프로그래밍 방식으로 선택하는 것을 지원하지 않습니다. 사용자 ID 및 트랜잭션 ID의 사용자 정의에 대한 정보는 [Node.js 애플리케이션에 대한 보안](#)의 내용을 참조하십시오.

Node.js 애플리케이션에서 로컬로 최적화된 호출 함수를 대신하여 호출되고 있음을 감지하는 파이프라인 핸들러 프로그램을 작성할 수 있습니다. 이러한 핸들러 프로그램은 요청 처리에 대해 프로그래밍 방식으로 의사결정할 수 있습니다. 예를 들어, Node.js 애플리케이션에서 도달하는 작업을 거부하거나 HTTP를 통해 도달하는 작업에만 관련된 로직을 분기할 수 있습니다. DFHWS-NODEJSAPP라는 제어 컨테이너는 현재 채널에서 CICS에 의해 채워지며 해당 콘텐츠는 작업이 시작된 NODEJSAPP 자원을 이름 지정합니다. 핸들러 프로그램과 애플리케이션에 접속된 채널은 이 컨테이너를 조회할 수 있습니다.

제 3 장 Node.js 애플리케이션 배치

NODEJSAPP 번들 파트를 작성하여 CICS 번들에서 Node.js 애플리케이션을 배치할 수 있습니다.

시작하기 전에

이 작업을 시작하기 전에 필요한 Node.js 애플리케이션 및 자산(예: 이미지 또는 HTML 파일)의 위치를 식별해야 합니다.

CICS가 Node.js 런타임을 시작하는 데 사용할 구성 정보를 지정하도록 프로파일을 작성하십시오. 이 프로파일을 사용하여 Node.js 애플리케이션이 해당 기능을 구성하기 위해 액세스할 수 있는 환경 변수를 지정할 수도 있습니다.

적절한 CICS 번들 프로젝트를 찾거나 새 프로젝트를 작성해야 합니다.

CICS 번들 프로젝트로 Node.js 애플리케이션을 복사하거나 별도의 zFS 위치에서 참조해야 합니다.

이 태스크 정보

이러한 단계를 수행하여 CICS 번들에서 NODEJSAPP 번들 파트를 작성할 수 있습니다.

프로시저

1. 프로젝트 탐색기 보기의 CICS Explorer에서 CICS 번들 프로젝트를 마우스 오른쪽 단추로 클릭하고 **새로 작성 > Node.js 애플리케이션**을 클릭하십시오.
제목이 **Node.js 애플리케이션** 작성인 마법사가 열립니다.
2. Node.js 애플리케이션 번들 파트의 이름을 입력하십시오. 이 이름은 번들이 설치될 CICS 리전 내에서 고유해야 합니다. **다음**을 클릭하십시오.
3. 실행할 초기 JavaScript 파일을 선택하십시오. zFS로 내보낸 경우 JavaScript 파일에 대한 완전한 경로가 255자를 넘지 않아야 합니다. **다음**을 클릭하십시오.
4. Node.js 애플리케이션의 프로파일을 선택하십시오. zFS로 내보낼 때 프로파일의 완전한 경로는 255자를 넘지 않아야 합니다. **완료**를 클릭하십시오.

결과

완료되면 마법사는 번들 프로젝트 내에 NODEJSAPP 번들 파트를 작성합니다. 프로파일을 로컬 파일 시스템 또는 Eclipse 작업공간에서 선택한 경우, 프로파일은 CICS 번들 프로젝트로 복사됩니다. 번들을 사용하는 경우 CICS는 Node.js 애플리케이션 초기 JavaScript 파일을 실행하기 위해 Node.js 런타임을 시작하려면 프로파일에서 구성을 사용합니다.

다음에 수행할 작업

이제 CICS 번들 프로젝트를 배치할 수 있습니다.

CICS 번들 프로젝트 배치

Node.js 애플리케이션에는 일반적으로 애플리케이션을 실행하기 전에 분석해야 하는 외부 모듈에 대한 종속성이 있습니다. CICS 프로젝트를 배치한 후 Node.js용 IBM SDK에서 제공하는 npm 도구를 실행해야 합니다(z/OS UNIX 시스템 서비스 셸에서의 z/OS). 예를 들어, 종속성이 Node.js 애플리케이션의 package.json 파일에서 설명된 경우 **npm install** 명령을 사용하여 저장소에서 종속 항목을 다운로드하여 설치하십시오.

Node.js 런타임 설치 확인

이 태스크는 CICS에서 Node.js 애플리케이션을 실행할 수 있는지 검증하는 단계와 예상치 못한 결과가 표시되는 경우 유용한 진단 피드백을 찾을 수 있는 위치를 자세히 설명합니다.

시작하기 전에

Node.js -z/OS용 IBM SDK를 설치하고 CICS 리전 사용자 ID의 해당 설치 디렉토리에 대한 읽기 및 실행 권한을 부여하십시오.

CICS 번들 디렉토리 /usr/lpp/cicsts/cicsts56/samples/nodejs/nodejsivp가 있는지 확인하십시오.

프로시저

1. CICS 번들 디렉토리와 해당 콘텐츠를 선택사항의 새 위치로 복사하십시오.
예: `cp -R /usr/lpp/cicsts/cicsts56/samples/nodejs/nodejsivp /u/jdoe/`
2. CICS 번들의 사본에서 Node.js 프로파일 `profiles/ivp_sample.profile`을 편집하십시오.
다음 환경 변수를 업데이트하십시오.
 - `PORT`= Node.js 애플리케이션이 웹 브라우저 요청을 서비스하는 데 사용할 수 있는 사용 가능한 HTTP 포트 번호로 설정하십시오. 포트는 다른 애플리케이션과 공유할 수 없습니다.
3. CICS 번들의 Node.js 프로파일에는 시스템 전체 Node.js 구성 데이터를 포함하는 zFS 파일을 참조하는 `%INCLUDE` 명령문이 있습니다. 다음 위치에 이 파일을 작성하십시오. 대상 CICS 리전에서 `<USSCONFIG>`를 `USSCONFIG SIT` 매개변수의 값으로 대체하는 `<USSCONFIG>/nodejsprofiles/general.profile`. 이 파일은 다음 값을 설정해야 합니다.
 - `NODE_HOME`= Node.js -z/OS용 IBM SDK 설치 디렉토리로 설정하십시오.
 - 출력 진단 파일이 작성되는 디렉토리로 설정된 `WORK_DIR`= . . 값은 CICS 리전 사용자 ID의 홈 디렉토리를 의미합니다.
4. 샘플 자원 정의 `DFHNIIVP`를 그룹 `DFH$NODJ`에서 선택한 그룹으로 복사하십시오.
5. `DFHNIIVP`의 사본을 편집하여 `BUNDLEDIR` 속성을 [12 페이지의 『1』](#) 단계에서 CICS 번들 사본의 디렉토리로 설정하십시오.
6. `DFHNIIVP`의 사본을 설치하십시오. Node.js 애플리케이션이 시작되며 HTTP 요청을 청취합니다.
7. HTTP 요청을 Node.js 애플리케이션에 전송하도록 웹 브라우저를 사용하십시오.
예를 들면 다음과 같습니다. `http://hostname:port/`, 여기서 `hostname`은 CICS가 실행 중인 z/OS에서 TCP/IP 스택의 완전한 호스트 이름이며 `port`는 [12 페이지의 『2』](#) 단계에서 선택된 값입니다.
웹 브라우저에는 다음과 같은 응답이 표시됩니다. "축하합니다. Node.js IVP 애플리케이션 IVPSAMPLE을 성공적으로 실행했습니다."

다음에 수행할 작업

웹 브라우저에 예상된 응답이 표시되지 않으면 [Node.js 애플리케이션 문제점 해결](#)의 진단 정보를 검토하십시오.

관련 정보

제 4 장 Node.js 지원 설정

CICS 리전에서 Node.js를 지원하는 기본 설정 태스크를 수행합니다.

Node.js 프로파일 유효성 검증 및 특성

Node.js 프로파일에는 시작 시 Node.js 런타임으로 전달되는 옵션 및 환경 변수 세트가 포함됩니다.

Node.js 프로파일은 일반적으로 CICS 번들의 일부로 배치되지만 zFS의 절대 파일 경로도 참조할 수 있습니다.

Node.js 프로파일을 코딩하는 규칙

Node.js 프로파일은 USS 파일 시스템에 저장될 때 EBCDIC으로 인코딩된 텍스트 파일입니다. CICS 번들에 Node.js 프로파일이 작성되면 텍스트 편집기를 사용하여 워크스테이션에서 편집할 수 있습니다. USS로 전송할 때는 EBCDIC으로 변환해야 합니다. CICS 번들 프로젝트를 USS로 내보낼 때 CICS Explorer에서 이 변환을 자동으로 수행합니다. Node.js 프로파일을 코딩할 때 이 규칙을 따르십시오.

대소문자 구분

모든 매개변수 키워드와 피연산자는 대소문자를 구분하므로 CICS 환경의 JVM에 대한 옵션, JVM 시스템 특성 또는 Node.js 프로파일 및 명령행 옵션에 표시된 대로 정확하게 지정해야 합니다.

설명

설명을 추가하거나 옵션을 삭제하지 않고 주석 처리하려면 설명의 각 행을 # 기호로 시작하십시오. 실행 프로그램이 파일을 읽을 때 설명 행이 무시됩니다.

빈 행 또한 무시됩니다. 빈 행을 옵션 또는 옵션 그룹 간의 분리 문자로 사용할 수 있습니다.

프로파일 구분 분석 코드에서 인라인 주석을 제거합니다. 인라인 설명은 다음과 같이 정의됩니다.

- 설명은 # 기호로 시작됩니다.
- 하나 이상의 공백(탭)이 앞에 추가됩니다.
- 따옴표로 묶은 텍스트에 포함되지 않습니다.

| 표 1. 인라인 설명 예제 | |
|--|----------------------------------|
| 코드 | 결과 |
| MYVAR=myValue # Comment | MYVAR=myValue |
| MYVAR=#myValue # Comment | MYVAR=#myValue |
| MYVAR=myValue "# Quoted comment" # Comment | MYVAR=myValue "# Quoted comment" |

연속

옵션의 경우에는 텍스트 파일에서 행 끝으로 값이 구분됩니다. 입력 또는 편집하는 값이 편집기 창에 너무 긴 경우 화면 이동 방지를 위해 행을 구분할 수 있습니다. 다음 행에서 계속 진행하려면 다음 예제와 같이 백슬래시 문자와 공백 연속 문자로 현재 행을 종료하십시오.

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

동일한 행에 여러 옵션을 삽입하지 마십시오.

파일 포함

%INCLUDE=<file_path>를 사용하여 프로파일에 파일을 포함할 수 있습니다. 이 파일에는 프로파일과 별개로 유지보수할 수 있는 공통 시스템 전체 구성이 있을 수 있습니다. 이 파일을 사용하면 여러 프로파일에 공통인 구성을 공유할 수 있으므로, 프로파일의 제어 기능이 강화되고 더 쉽게 유지보수할 수 있습니다.

다음 규칙이 적용됩니다.

- <file_path>는 zFS의 완전한 파일이어야 합니다.
 - <file_path> 시작 부분에서 상대 디렉토리를 사용하지 마십시오(예: . 및 ..). 이 디렉토리는 UNIX System Services에서 처리 중에 변경될 수 있는 Language Environment 현재 작업 디렉토리의 상대적인 것으로 해석됩니다.
 - <file_path>가 없거나 CICS 리전 사용자 ID에 <file_path>에 대한 읽기 액세스 권한이 없으면 DFHSJ1308 메시지가 발행됩니다.
- <file_path>에는 기호가 포함될 수 있습니다(예: &USSCONFIG;).
 - &DATE; 및 &TIME; 기호는 %INCLUDE 지시문 앞뒤에 올 수 있는 시간대 옵션(TZ)을 통해 설정되는 형식 때문에 허용되지 않습니다.
- <file_path>의 콘텐츠가 %INCLUDE 지시문을 대체합니다.
- 프로파일은 임의의 수의 %INCLUDE 지시문을 포함할 수 있습니다.
- 순환 참조를 수행하면 Skipping duplicate 메시지가 표시됩니다. 예를 들어 Profile-A에는 Profile-B가 포함되고 Profile-B에는 Profile-C가 포함될 수 있습니다. 그러나 Profile-B에 Profile-A가 포함되면 지시문이 무시됩니다.

옵션의 복수 인스턴스

동일한 옵션의 여러 인스턴스가 프로파일에 포함되는 경우 마지막으로 찾은 옵션의 값을 사용하고 이전 값은 무시됩니다.

UNIX System Services 디렉토리 경로

프로파일에서 zFS 파일 또는 디렉토리 값을 지정할 때 따옴표를 사용하지 마십시오.

Node.js 프로파일의 고유 규칙

프로파일의 이름

- 이름은 z/OS UNIX 시스템 서비스에서 파일에 유효한 모든 이름이 가능합니다. DFH로 시작되는 이름은 사용하지 마십시오. 이러한 문자는 CICS에서 사용하도록 예약되어 있기 때문입니다.
- 프로파일은 UNIX 파일이므로 대소문자 구분이 중요합니다. CICS에서 이름을 지정하는 경우 z/OS UNIX 파일 이름과 동일한 대소문자 조합을 사용하여 입력해야 합니다.

Node.js 프로파일 및 명령행 옵션

Node.js 프로파일 및 명령행 옵션이 해당 설명과 함께 나열됩니다.

명령행 옵션

Node.js 런타임에 전달된 Node.js 프로파일에 명령행 옵션을 지정할 수 있습니다. 하이픈으로 시작하는 행은 명령행 옵션으로 처리됩니다. 명령행 옵션은 행당 하나씩 지정해야 합니다. 명령행 옵션의 추가 매개변수는 같은 행에 표시되어야 합니다.

다음은 프로파일에 명령행 옵션을 설정하는 예입니다.

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

프로파일 옵션 및 해당 설명

해당되는 경우 기본값은 옵션이 지정되지 않을 때 CICS가 사용하는 값입니다. 샘플 Node.js 프로파일에서 기본 값과 다른 값을 지정할 수 있습니다.

LIBPATH_PREFIX=pathnames

LIBPATH_SUFFIX=pathnames

Node.js 런타임 또는 모듈에서 사용하고 z/OS UNIX에서 확장자가 .so인 고유 C++ 동적 링크 라이브러리(DLL) 파일을 검색할 디렉토리 경로를 지정합니다. 여기에는 Node.js 애플리케이션을 실행하는 데 필요한 파일과 애플리케이션 코드 또는 서비스로 로드되는 추가 고유 라이브러리가 포함됩니다.

Node.js 런타임의 기본 라이브러리 경로는 Node.js 프로파일에 **USSHOME** 시스템 초기화 매개변수 및 **NODE_HOME** 옵션으로 지정된 디렉토리를 사용하여 자동으로 빌드됩니다.

LIBPATH_SUFFIX 옵션을 사용하여 라이브러리 경로를 확장할 수 있습니다. 이 옵션은 라이브러리 경로 끝에서 기본 라이브러리 경로 다음에 디렉토리를 추가합니다. 이 옵션을 사용하면 애플리케이션에 필요한 Node.js 모듈에서 사용하는 추가 고유 라이브러리를 포함하는 디렉토리를 지정할 수 있습니다.

LIBPATH_PREFIX 옵션은 라이브러리 경로 시작에서 기본 라이브러리 경로 이전에 디렉토리를 추가합니다. 이 옵션 사용에 주의하십시오. 지정된 디렉토리의 DLL 파일이 기본 라이브러리 경로에서 DLL 파일과 이름이 같은 경우 제공된 파일 대신 해당 파일이 로드됩니다.

경로 이름에서 경로를 구분하는 데 쉼표가 아니라 콜론을 사용하십시오.

LOG={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.log|file_name}

Node.js 애플리케이션 조작 중에 CICS에서 로그 메시지를 쓸 z/OS UNIX 파일의 이름을 지정합니다. 이 파일에 작성된 메시지 레벨은 **LOG_LEVEL** 옵션을 통해 제어합니다.

LOG에 절대 파일 이름이 지정되면 CICS에서 존재하지 않는 경로에 디렉토리를 작성합니다.

이 파일이 있으면 파일 끝에 출력이 추가됩니다. 각 Node.js 애플리케이션의 고유한 출력 파일을 작성하려면 샘플 Node.js 프로파일에 설명된 대로 파일 이름에 **&NODEJSAPP;** 및 **&APPLID;** 기호를 사용하십시오.

LOG_FILES_MAX={0|number}

시스템에서 보관하는 이전 로그 파일 수를 지정합니다. 기본 설정인 0은 로그 파일의 이전 버전이 모두 보존됨을 나타냅니다. 이 값을 변경하여 파일 시스템에서 남겨두려는 로그 파일 수를 지정할 수 있습니다.

STDOUT, **STDERR**, **STDIN**, **LOG** 및 **TRACE**에서 기본 스킴을 사용하거나 사용자 정의된 경우 **&DATE;**, **&TIME;** 패턴을 포함한 다음 각 로그 유형의 최신 nn만 시스템에 보관합니다. 사용자 정의에 출력을 고유하게 작성하는 변수가 포함되지 않으면 파일이 추가되고 삭제에 대한 요구사항은 없습니다. 특수 값인 0은 삭제하지 않음을 나타냅니다.

LOG_LEVEL={INFO|WARNING|ERROR|NONE}

.log 파일에 리턴되는 로깅된 정보에 대한 제어 기능을 제공합니다. 값이 **NONE**이면 모든 출력이 비활성화되고 파일이 비어 있습니다. 다른 모든 값은 .log 파일에 쓴 가장 낮은 로그 유형을 표시합니다. 예를 들어 **WARNING**을 선택하면 **WARNING** 레벨 이상의 로그 항목을 제공합니다.

NODE_HOME=pathname

z/OS UNIX에서 Node.js -z/OS용 IBM SDK의 설치 위치를 지정합니다. 이것은 필수 매개변수입니다.

NODEJSAPP_DISABLE_TIMEOUT={10000|number}

NODEJSAPP를 사용 안함으로 설정할 때 CICS에서 대기하는 제한시간 값(밀리초)을 지정합니다. 기본값은 10000(10초)입니다. 지정할 수 있는 최소값은 1000(1초)입니다.

SIGTERM 신호를 보낸 다음 Node.js 애플리케이션 프로세스가 종료될 때까지 CICS에서 대기할 최소 시간입니다. 이 제한시간 값이 만료되기 전에 프로세스가 종료되지 않으면 CICS에서 **SIGKILL** 신호를 발행하여 프로세스를 강제로 종료합니다.

PRINT_PROFILE={TRUE|FALSE}

이 옵션이 **TRUE**로 설정되어 있으면, Node.js 런타임 및 애플리케이션에 전달된 Node.js 프로파일의 옵션과 환경 변수가 **SYSPRINT**에 출력됩니다.

STDERR={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.stderr|file_name}

STDERR 스트림의 경로가 재지정될 z/OS UNIX 파일의 이름을 지정합니다. 이 옵션의 값을 설정하지 않으면 CICS가 각 Node.js 애플리케이션에 고유한 추적 파일을 자동으로 작성합니다.

STDERR에 절대 파일 이름이 지정되면 CICS에서 존재하지 않는 경로에 디렉토리를 작성합니다.

이 파일이 있으면 파일 끝에 출력이 추가됩니다. 각 Node.js 애플리케이션의 고유한 출력 파일을 작성하려면 샘플 Node.js 프로파일에 설명된 대로 파일 이름에 **&NODEJSAPP;** 및 **&APPLID;** 기호를 사용하십시오.

STDERR이 비어 있으면 CICS에서 **&APPLID;** 및 **&NODEJSAPP;** 기호와 Node.js 애플리케이션에서 고유 출력 파일을 작성하려고 시작한 날짜와 시간소인을 사용합니다.

STDIN=*file_name*

STDIN 스트림을 읽는 z/OS UNIX 파일의 이름을 지정합니다. 이 옵션의 값을 지정하지 않으면 CICS가 이 파일을 작성하지 않습니다.

STDOUT={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.stdout|*file_name*}

STDOUT 스트림의 경로가 재지정될 z/OS UNIX 파일의 이름을 지정합니다. 이 옵션의 값을 설정하지 않으면 CICS가 각 Node.js 애플리케이션에 고유한 추적 파일을 자동으로 작성합니다.

STDOUT에 절대 파일 이름이 지정되면 CICS에서 존재하지 않는 경로에 디렉토리를 작성합니다.

이 파일이 있으면 파일 끝에 출력이 추가됩니다. 각 Node.js 애플리케이션의 고유한 출력 파일을 작성하려면 샘플 Node.js 프로파일에 설명된 대로 파일 이름에 &NODEJSAPP; 및 &APPLID; 기호를 사용하십시오.

STDOUT이 비어 있으면 CICS에서 &APPLID; 및 &NODEJSAPP; 기호와 Node.js 애플리케이션에서 고유 출력 파일을 작성하려고 시작한 날짜와 시간소인을 사용합니다.

TRACE={&APPLID;.&NODEJSAPP;.Dyyyymmdd.Thhmmss.trace|*file_name*}

Node.js 애플리케이션의 조작 중에 Node.js 추적을 쓸 z/OS UNIX 파일의 이름을 지정합니다. 이 옵션의 값을 설정하지 않으면 CICS가 각 Node.js 애플리케이션에 고유한 추적 파일을 자동으로 작성합니다.

TRACE에 절대 파일 이름이 지정되면 CICS에서 존재하지 않는 경로에 디렉토리를 작성합니다.

이 파일이 있으면 파일 끝에 출력이 추가됩니다. 각 Node.js 애플리케이션의 고유한 출력 파일을 작성하려면 샘플 Node.js 프로파일에 설명된 대로 파일 이름에 &NODEJSAPP; 및 &APPLID; 기호를 사용하십시오.

WORK_DIR={./|*tmp|pathname*}

CICS 리전에서 STDOUT, STDERR, TRACE, Node.js 애플리케이션과 관련된 다른 출력 파일에 사용할 z/OS UNIX의 디렉토리를 지정합니다. 마침표(.)가 제공된 Node.js 프로파일에 정의되며 이는 CICS 리전 사용자 ID의 홈 디렉토리가 작업 디렉토리로 사용됨을 나타냅니다. 이 디렉토리는 CICS 설치 중에 작성될 수 있습니다. 이 디렉토리가 없거나 WORK_DIR이 생략되면 /tmp가 z/OS UNIX 디렉토리 이름으로 사용됩니다.

작업 디렉토리의 절대 경로 또는 상대 경로를 지정할 수 있습니다. 상대 작업 디렉토리는 CICS 리전 사용자 ID의 홈 디렉토리에 상대적입니다. 홈 디렉토리를 Node.js와 연관된 활동의 작업 디렉토리로 사용하지 않거나 CICS 리전에서 z/OS 사용자 ID(UID)를 공유하므로 동일한 홈 디렉토리가 있는 경우 CICS 리전마다 다른 작업 디렉토리를 작성할 수 있습니다.

&APPLID; 기호(이 기호를 사용하여 CICS에서 실제 CICS 리전 APPLID 대체)를 사용하는 디렉토리 이름을 지정하면 모든 CICS 리전에서 Node.js 프로파일 세트를 공유하는 경우에도 리전마다 고유한 작업 디렉토리가 있습니다. 예를 들어, 다음을 지정하는 경우

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

해당 Node.js 프로파일을 사용하는 각 CICS 리전이 고유한 작업 디렉토리를 갖습니다. 관련 디렉토리가 z/OS UNIX에서 작성되고 CICS 리전에 읽기, 쓰기, 실행 액세스 권한이 부여되었는지 확인하십시오.

작업 디렉토리에 고정된 이름을 지정할 수도 있습니다. 이러한 경우 또한 관련 디렉토리가 z/OS UNIX에서 작성되고 올바른 CICS 리전에 액세스 권한이 부여되었는지 확인해야 합니다. 작업 디렉토리에 고정된 이름을 사용하는 경우 CICS 리전에서 Node.js 프로파일을 공유하는 모든 Node.js 애플리케이션의 출력 파일이 해당 디렉토리에서 작성됩니다. 출력 파일에 고정된 파일 이름을 사용하는 경우 해당 CICS 리전에 있는 모든 Node.js 애플리케이션의 출력이 동일한 z/OS UNIX 파일에 추가됩니다. 동일한 파일에 추가되지 않게 하려면 &NODEJSAPP; 기호와 &APPLID; 기호를 사용하여 각 Node.js 애플리케이션의 고유 출력 및 덤프 파일을 생성하십시오.

z/OS UNIX의 경우 **USSHOME** 시스템 초기화 매개변수로 정의된 CICS 파일의 홈 디렉토리인 CICS 설치 디렉토리에 작업 디렉토리를 정의하지 마십시오.

관련 링크

[Node.js 애플리케이션에 사용할 환경 변수](#)

Node.js 프로파일에서 사용된 기호

Node.js 프로파일에 지정된 변수에 대체 기호를 사용할 수 있습니다. 이 기호의 값은 런타임 시 판별되므로, 여러 Node.js 애플리케이션과 CICS 리전에 공통 프로파일을 사용할 수 있습니다.

지원되는 기호는 다음과 같습니다.

&APPLID;

이 기호를 사용하면 CICS 리전 애플리케이션 ID인 APPLID 시스템 초기화 매개변수의 값으로 기호가 대체됩니다. 이러한 방식으로 모든 영역에 동일한 프로파일을 사용할 수 있으며 영역에 고유한 작업 디렉토리 또는 출력 대상을 계속 갖습니다. APPLID는 항상 대문자입니다.

&BUNDLE;

이 기호를 사용하면 NODEJSAPP를 설치 중인 CICS 번들의 이름으로 기호가 대체됩니다.

&BUNDLEID;

이 기호를 사용하면 NODEJSAPP를 설치 중인 CICS 번들의 ID로 기호가 대체됩니다.

&CONFIGROOT;

이 기호를 사용하면 NODEJSAPP가 설치되는 번들의 루트 디렉토리가 런타임 시 대체됩니다.

&DATE;

이 기호를 사용하는 경우 기호는 런타임 시 *Dyyymmdd* 형식의 현재 날짜로 바뀝니다.

&NODEJSAPP;

이 기호를 사용하면 NODEJSAPP 자원의 이름이 런타임 시 대체됩니다. 각 Node.js 애플리케이션에 고유한 출력 또는 덤프 파일을 작성하려면 이 기호를 사용하십시오.

&TIME;

이 기호를 사용하는 경우 기호는 런타임 시 *Thhmmss* 형식의 NODEJSAPP 시작 시간으로 바뀝니다.

&USSCONFIG;

이 기호를 사용하면 CICS 구성 파일의 디렉토리인 USSCONFIG 시스템 초기화 매개변수의 값으로 기호가 대체됩니다.

&USSHOME;

이 기호를 사용하면 CICS Transaction Server 제품 파일의 디렉토리인 USSHOME 시스템 초기화 매개변수의 값으로 기호가 대체됩니다.

Node.js 애플리케이션의 시간대 설정

TZ 환경 변수는 시스템의 '로컬' 시간을 지정합니다. JVM 프로파일에 추가하여 JVM 서버에 맞게 설정할 수 있습니다. TZ 변수를 설정하지 않으면 시스템은 UTC를 기본값으로 사용합니다. TZ 변수가 설정되고 나면 다시 시작하거나 추가 개입 없이 JVM에서 필요한 대로 일광 절약 시간으로(으로부터) 자동 전환합니다.

JVM 서버 또는 Node.js 애플리케이션의 시간대를 설정하는 경우 다음 문제를 인식해야 합니다.

- JVM 또는 Node.js 프로파일의 TZ 변수는 GMT와의 로컬 MVS™ 시스템 오프셋과 일치해야 합니다. 로컬 MVS 시스템 오프셋을 표시하고 설정하는 방법에 대한 자세한 정보는 z/OS Communications Server: IP 구성 참조서의 TIMEZONE 명령문 및 z/OS MVS Setting Up a Sysplex에서 sysplex의 로컬 시간 조정의 내용을 참조하십시오.
- 사용자 정의된 시간대는 지원되지 않으며 UTC 또는 JVMTRACE 파일(JVM 서버용)이나 TRACE 파일(Node.js 애플리케이션용)의 혼합 시간대 출력으로 장애 복구가 수행됩니다.
- 시간대 문자열로 LOCALTIME이 보이면 구성에 불일치가 존재하는 것입니다. 이는 로컬 MVS 시간과 사용자가 설정한 TZ 사이 또는 로컬 MVS 시간과 JVM 또는 Node.js 프로파일의 기본 설정 사이에서 나타날 수 있습니다. 각 항목이 올바른 경우에도 출력은 혼합 시간대입니다.

POSIX 시간대 형식 사용

POSIX 시간대 형식은 짧은 양식과 긴 양식이 있습니다. 둘 중 하나를 사용하여 TZ 환경 변수를 설정할 수 있지만, 짧은 양식을 사용하면 입력 오류의 가능성이 줄어듭니다.

긴 양식의 예:

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

짧은 양식의 예:

TZ=GMT0BST

시스템이 실행 중인 시간대를 알아내려면 USS로 로그인하여 `echo $TZ`를 입력하십시오. 결과적으로 TZ 환경 변수가 설정되어야 하는 값의 긴 양식이 표시됩니다.

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

POSIX 시간대 형식에 대한 자세한 분류는 IBM developerWorks® 사이트에 있는 [POSIX와 Olson 시간대 형식의 내용을 참조하십시오](#).

NODEJSAPP 출력, 로그 및 추적 위치 제어

Node.js 애플리케이션, Node.js 런타임 및 CICS의 출력이 z/OS UNIX 파일에 작성됩니다. z/OS UNIX 파일은 CICS Node.js 프로파일의 STDOUT, STDERR, LOG, TRACE 옵션을 사용하여 지정합니다.

기본적으로 Node.js 애플리케이션의 출력은 z/OS UNIX 파일 시스템에 작성됩니다. 기본 파일 이름 변환은 `WORK_DIR/APPLID/BUNDLEID/NODEJSAPP/D<date>.T<time>.<stdxxx>`입니다. 여기서 `WORK_DIR`, `APPLID`, `BUNDLEID`, `NODEJSAPP`, `date` 및 `time`은 [Node.js 프로파일 유효성 검증 및 특성 주제](#)에 자세히 설명된 값으로 대체되는 기호입니다.

기본값을 덮어쓰려면 각 STDOUT, STDERR, LOG 및 TRACE 프로파일 옵션의 zFS 파일 이름을 지정할 수 있습니다. 이미 있는 파일 이름을 쓰는 경우 출력이 추가됩니다. 이 경우 문제점 판별에 도움이 되도록 각 출력 라인의 접두부로 헤더를 사용하여 Node.js 애플리케이션 인스턴스를 식별하면 좋습니다.

CICS 리전에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공

CICS에는 z/OS UNIX의 디렉토리 및 파일에 대한 액세스 권한이 필요합니다. 설치 중에 각 CICS 리전에 z/OS UNIX 사용자 ID(UID)가 지정됩니다. 리전은 z/OS UNIX 그룹 ID(GID)에 지정된 ESM 그룹에 연결됩니다. UID 및 GID를 사용하여 CICS 리전이 z/OS UNIX의 디렉토리와 파일에 액세스할 수 있는 권한을 부여할 수 있습니다.

시작하기 전에

본인이 z/OS UNIX의 슈퍼유저 또는 디렉토리와 파일의 소유자인지 확인하십시오. 디렉토리와 파일의 소유자는 처음에 제품을 설치하는 시스템 프로그래머의 UID로 설정됩니다. 디렉토리와 파일의 소유자는 설치 중에 GID가 지정된 ESM 그룹에 연결되어야 합니다. 소유자는 ESM 그룹을 기본 그룹(DFLTGRP)으로 갖거나 보조 그룹 중 하나로 그룹에 연결될 수 있습니다.

이 태스크 정보

z/OS UNIX 시스템 서비스는 각 CICS 리전을 UNIX 사용자로 간주합니다. 다양한 방법으로 사용자에게 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한을 부여할 수 있습니다. 예를 들어, 디렉토리 또는 파일에 적절한 그룹 권한을 CICS 리전이 연결되는 ESM 그룹에 제공할 수 있습니다. 이 옵션은 프로덕션 환경에 가장 적합하며 다음 단계에서 설명합니다.

프로시저

1. z/OS UNIX에서 CICS 리전에 액세스 권한이 필요한 디렉토리와 파일을 식별하십시오.

| 구성 | 기본 디렉토리 | 권한 | 설명 |
|--|--|---------|------------------------------------|
| BUNDLE 자원의 BUNDLEDIR 속성 | 사용자가 설정합니다. | 읽기 및 실행 | CICS 번들을 포함하는 디렉토리입니다. |
| Node.js 프로파일의 NODE_HOME 옵션 | 사용자가 설정합니다. 참고: Node.js의 기본 설치 디렉토리는 <code>/usr/lpp/IBM/cnj/v8r0/IBM/node-latest-os390-s390x</code> 입니다. | 읽기 및 실행 | Node.js -z/OS용 IBM SDK 설치 디렉토리입니다. |

| 구성 | 기본 디렉토리 | 권한 | 설명 |
|---|------------------------------|-------------|---|
| USSHOME SIT 매개변수 | /usr/lpp/cicsts/ cicsts56 | 읽기 및 실행 | z/OS UNIX에서 CICS 파일의 설치 디렉토리입니다. 이 디렉토리의 파일에는 샘플 프로파일이 포함되어 있습니다. |
| Node.js 프로파일의 WORK_DIR 옵션 | /u/CICS region userid | 읽기, 쓰기 및 실행 | CICS 리전의 작업 디렉토리입니다. 이 디렉토리에는 Node.js 애플리케이션의 입력, 출력, 메시지가 포함됩니다. |

2. 권한을 표시하기 위해 디렉토리와 파일을 나열합니다.

시작하려는 디렉토리로 이동하고 다음 UNIX 명령을 실행하십시오.

```
ls -la
```

현재 디렉토리가 CICSHT##의 홈 디렉토리일 때 z/OS UNIX 시스템 서비스 셸 환경에서 이 명령이 실행되면 다음 예와 같은 목록이 나타날 수 있습니다.

```
/u/cicsht##:>ls -la
total 256
drwxr-xr-x  2 CICSHT## CICSTS56      8192 Mar 15  2008 . drwx-----  4 CICSHT##
CICSTS56    8192 Jul  4 16:14 .. -rw-----  1 CICSHT## CICSTS56    2976 Dec  5  2010
Snap0001.trc
-rw-r--r--  1 CICSHT## CICSTS56      1626 Jul 16 11:15 stderr
-rw-r--r--  1 CICSHT## CICSTS56         0 Mar 15  2010 stdin
-rw-r--r--  1 CICSHT## CICSTS56      458 Oct  9 14:28 stdout
/u/cicsht##:>
```

3. 그룹 권한을 사용하여 액세스 권한을 제공하는 경우, 각 디렉토리 및 파일에 대한 그룹 권한이 자원에 대해 CICS에 필요한 액세스 레벨을 제공하는지 확인하십시오.

권한은 문자 **r**, **w**, **x**, **-**로 세 개 세트에 표시됩니다. 이러한 문자는 읽기, 쓰기, 실행, 없음을 나타내며 명령 행 왼쪽 옆에 두 번째 문자부터 표시됩니다. 첫 번째 세트는 소유자 권한이고 두 번째 세트는 그룹 권한이며 세 번째 세트는 기타 권한입니다.

이전 예제에서는 소유자가 **stderr**, **stdin** 및 **stdout**에 대해 읽기 및 쓰기 권한을 갖지만 그룹과 다른 모두는 읽기 권한만 갖습니다.

4. 자원에 대한 그룹 권한을 변경하려면 UNIX 명령 **chmod**를 사용하십시오.

다음 예제는 읽기, 쓰기, 실행을 수행할 이름 지정된 디렉토리와 그 서브디렉토리, 파일에 대한 그룹 권한을 설정합니다. **-R**은 모든 서브디렉토리와 파일에 반복적으로 권한을 적용합니다.

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

다음 예제는 읽고 실행할 이름 지정된 파일에 대해 그룹 권한을 설정합니다.

```
chmod g+rx filename
```

다음 예제는 이름 지정된 두 개 파일에서 그룹에 대한 쓰기 권한을 끕니다.

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

이들 예제 모두에서 **g**는 그룹 권한을 지정합니다. 다른 권한을 정정하려는 경우 **u**가 사용자(소유자) 권한을 지정하고 **o**가 다른 권한을 지정합니다.

5. 각 자원에 대한 그룹 권한을 CICS 리전이 z/OS UNIX에 액세스하기 위해 선택한 RACF® 그룹에 지정하십시오. 각 디렉토리와 해당 서브디렉토리 및 디렉토리 내 파일에 대한 그룹 권한을 지정해야 합니다.

다음 UNIX 명령을 입력하십시오.

```
chgrp -R GID directory
```

GID는 ESM 그룹의 숫자 **GID**이고 **directory**는 CICS 리전 권한을 지정할 디렉토리의 전체 경로입니다.

예를 들어, /usr/lpp/cicsts/cicsts56 디렉토리에 대한 그룹 권한을 지정하려면 다음 명령을 사용하십시오.

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

CICS 리전 사용자 ID는 ESM 그룹에 연결되므로 CICS 리전은 이들 모든 디렉토리와 파일에 적절한 권한을 갖습니다.

결과

CICS에 Node.js 애플리케이션을 실행하기 위해 z/OS UNIX의 디렉토리와 파일에 액세스할 수 있는 적절한 권한이 있음을 확인했습니다.

파일 이동 또는 새 파일 작성과 같이 설정하려는 CICS 기능을 변경하는 경우, 이 프로시저를 반복하여 CICS 리전에 새 파일 또는 이동한 파일에 액세스할 수 있는 권한이 있는지 확인해야 합니다.

다음에 수행할 작업

[Node.js 런타임 설치 확인](#)의 샘플을 사용하여 Node.js 지원이 올바르게 설정되었는지 확인하십시오.

Node.js의 메모리 한계 설정

일반적으로 Node.js 애플리케이션에는 컴파일된 언어로 작성된 경우보다 많은 메모리가 필요합니다. 부분적으로 Node.js 애플리케이션에서 동시에 여러 클라이언트 요청에 서비스를 제공하는 기능 때문입니다. CICS 및 Node.js에 Node.js 애플리케이션을 실행할 수 있는 충분한 스토리지와 메모리가 있는지 확인해야 합니다.

이 태스크 정보

Node.js 애플리케이션에 필요한 스토리지는 DSA, EDSA 또는 GDSA와 같은 CICS 관리 스토리지에서 할당되지 않습니다. 대신, 사용 가능한 MVS 개인용 영역에서 사용 가능한 스토리지를 사용합니다. 24비트, 31비트 및 64비트 주소 지정 영역에 할당되지 않은 개인용 영역 리전 스토리지가 충분히 사용 가능한지 확인하는 것이 중요합니다. 개인용 영역 리전의 스토리지가 부족해지면 CICS에서 스토리지 부족 메커니즘을 사용할 수 없습니다.

프로시저

1. z/OS **MEMLIMIT** 매개변수가 적절한 값으로 설정되었는지 확인하십시오.

이 매개변수는 CICS 주소 공간이 사용할 수 있는 64비트 스토리지의 크기를 제한합니다.

Node.js에서는 64비트 스토리지를 사용하므로 CICS 리전에서 이러한 용도 및 다른 용도로 64비트 스토리지를 사용하는 데 충분히 큰 값으로 **MEMLIMIT**를 설정해야 합니다.

2. 시동 작업 스트림의 **REGION** 매개변수가 Node.js를 실행할 수 있을 정도로 충분한지 확인하십시오.

이 매개변수는 CICS 주소 공간이 사용할 수 있는 31비트 스토리지의 크기를 제한합니다.

3. 각 Node.js 애플리케이션에 필요한 24비트 스토리지를 사용할 수 있는지 확인하십시오.

제 5 장 Node.js 성능 개선

Node.js 애플리케이션의 성능을 개선하기 위해 다양한 조치를 취할 수 있습니다.

DFHSJNRO로 NODEJSAPP의 인클레이브 수정

DFHSJNRO는 Node.js 애플리케이션이 실행되는 Language Environment® 인클레이브의 기본 런타임 옵션 세트를 제공하는 샘플 프로그램입니다. 예를 들어 Node.js 프로세스에 대한 스토리지 할당 매개변수를 정의합니다. 모든 워크로드에 최적화된 기본 런타임 옵션을 제공할 수 없습니다.

이 태스크 정보

샘플 프로그램을 갱신하여 Language Environment 인클레이브를 조정하거나 샘플을 기반으로 고유 프로그램을 만들 수 있습니다.

프로그램은 어셈블리 언어로 작성해야 하며 CICS 변환기로 변환되어서는 안 됩니다. 옵션은 문자열로 지정되며, 뒤에 런타임 옵션이 오는 2바이트 문자열 길이로 구성됩니다. 모든 Language Environment 런타임 옵션의 최대 길이는 255바이트입니다.

프로시저

1. DFHSJNRO 프로그램 소스를 CICSTS56.CICS.SDFHSAMP 라이브러리에서 새 위치로 복사하십시오.
CICS 리전에 유지보수가 적용되는 경우 프로그램에 변경사항을 반영할 수 있습니다.
2. 각 옵션에 대한 약어를 사용하여 런타임 옵션을 편집하고 변경사항은 총 200 바이트 이내로 제한하십시오.
z/OS Language Environment 프로그래밍 안내서는 Language Environment 런타임 옵션에 대한 모든 정보를 제공합니다.
 - CICS는 이 목록에 일부 옵션을 추가하므로 빠른 처리를 위해 옵션 목록의 크기를 최소값으로 유지하십시오.
 - HEAP64 옵션을 사용하여 초기 힙 할당을 지정하십시오.
 - ALL31 옵션, POSIX 옵션, XPLINK 옵션은 CICS에서 강제 실행됩니다. CICS에서는 ABTERMENC 옵션이 (ABEND)로 설정되고 TRAP 옵션은 (ON,NOSP)로 설정됩니다.
 - RPTO 및 RPTS 옵션으로 생성되는 출력은 CESE 트랜지언트 데이터 큐에 기록됩니다.
 - 출력을 생성하는 옵션은 각 Node.js 애플리케이션 종료 시 생성합니다. 생성되어 CESE로 방향이 지정될 수 있는 출력 볼륨을 고려하십시오.
3. DFHASMVS 프로시저를 사용하여 프로그램을 컴파일하십시오.
4. 컴파일된 프로그램을 CICS에서 사용 가능한 로드 라이브러리로 복사하십시오.
5. DFHSJNRO 이외의 이름을 사용하는 경우 NODEJSAPP CICS 번들 파트를 편집하고 `lerunopts="PROGRAM"` 속성을 지정하십시오.

결과

NODEJSAPP를 포함하는 BUNDLE 자원을 사용하는 경우, CICS는 DFHSJNRO 프로그램에서 지정하는 런타임 옵션을 사용하여 Language Environment 인클레이브를 작성합니다. CICS는 런타임 옵션을 Language Environment로 전달하기 전에 그 길이를 확인합니다. 길이가 255바이트보다 크면 CICS는 NODEJSAPP을 시작하려 하지 않고 CSMT에 오류 메시지를 기록합니다. 사용자가 지정하는 값은 Language Environment로 전달되기 전에 CICS가 확인하지 않습니다.

Node.js 애플리케이션에 대한 저장영역 요구사항 계산

CICS® 리전에서 하나 이상의 Node.js 애플리케이션을 시작하려면 사용할 각 Node.js 애플리케이션에 대해 사용 가능한 충분한 저장영역이 있는지 확인해야 합니다. 동일한 리전에서 실행 중인 CICS 및 기타 제품에서는 상당한 양의 z/OS 스토리지가 필요할 수 있습니다. **DSALIM** 및 **EDSALIM**과 같은 CICS 스토리지 할당 매개변수는 스토리지 가용성에 영향을 주고 최고 요구사항에 비해 과도하게 할당될 수 있습니다.

이 태스크 정보

Node.js 애플리케이션에 필요한 스토리지는 CICS DSA 또는 EDSA 스토리지에서 할당되지 않습니다. 일부 스토리지는 C 코드로 실행되는 **malloc()**과 같은 요청을 처리하는 Language Environment에서 관리하며, 일부는 **IARV64** 및 **STORAGE OBTAIN**과 같은 z/OS 스토리지 관리 요청을 사용하는 Node.js 애플리케이션에서 직접 관리됩니다. Language Environment에서는 z/OS 스토리지 서비스를 사용합니다.

Node.js 런타임에서 사용하는 각 스레드에 대해 24비트 스토리지 4KB가 필요합니다. 사용되는 스레드 수는 일단 Node.js 런타임이 시작되면 고정되며, **UV_THREADPOOL_SIZE** 환경 변수를 설정하는 경우가 아니면 보통 8과 12 사이입니다. 또한 UNIX System Services에서는 각 스레드를 작성하는 프로세스 도중 연속 24비트 스토리지 256KB가 필요합니다.

Node.js 런타임은 JavaScript 오브젝트 및 JIT 컴파일된 코드의 힙을 할당합니다. z/OS 2.2에서 힙은 31비트 스토리지에서 할당됩니다. z/OS 2.3에서 힙은 31비트 및 64비트 스토리지 모두에 걸쳐 수행될 수 있습니다. 또한 힙은 여러 공간으로 구성되고 해당 크기는 서로 다릅니다. 테스트 환경에서 힙 사용을 측정하는 것은 힙 크기 요구사항을 추정하는 가장 간단한 방법입니다. 31비트 스토리지는 Node.js 런타임의 UNIX System Services 동적 링크 라이브러리(DLL) 파일을 로드할 때에도 필요합니다.

또한 64비트 스토리지는 C++ 힙 및 스택에 대해 Language Environment에서 할당됩니다. 이는 Node.js 런타임 코드에 의해 내부적으로 그리고 기본 모듈에 의해 사용됩니다.

Node.js 애플리케이션에 필요한 스토리지 총계를 추적하려면 다음 프로시저를 수행하십시오.

프로시저

1. 샘플 통계 프로그램 DFHOSTAT를 사용하여 24비트, 31비트 및 64비트 메모리 사용을 측정하십시오.

- 리전에서 현재 사용 가능한 24비트 개인 스토리지인 **1** Private Area storage available below 16Mb의 값을 기록하십시오.
- 리전에서 현재 사용 가능한 31비트 개인 스토리지인 **2** Private Area storage available above 16Mb의 값을 기록하십시오.
- 리전에서 현재 사용 가능한 64비트 개인 스토리지인 **3** MEMLIMIT minus usable within Private Memory Objects의 값을 기록하십시오.

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-----
MEMLIMIT Set By. . . . . : 200G
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. NODEJSAPP을 사용으로 설정하고 대표 워크로드를 실행하십시오. 사용 가능한 각 개인 스토리지 영역 값이 변경되는 방법을 관찰하고 개인 스토리지 영역이 제한되지 않도록 하십시오.

제 6 장 Node.js 애플리케이션 문제점 해결

Node.js 애플리케이션 관련 문제점이 있는 경우 CICS 및 Node.js에서 제공되는 진단을 사용하여 문제점의 원인을 판별할 수 있습니다.

CICS는 Node.js와 관련된 문제점을 진단하는 데 도움이 되는 일부 통계, 메시지 및 추적을 제공합니다. Node.js와 함께 제공되는 진단 도구 및 인터페이스는 Node.js 런타임 및 애플리케이션 실행에 대한 자세한 정보를 제공할 수 있습니다.

Node.js 애플리케이션의 실시간 및 오프라인 분석을 수행하며 무료로 제공되는 도구(예: IBM Health Center 또는 Appmetrics)를 사용할 수 있습니다. 자세한 정보는 [IBM 모니터링 및 진단 도구 - Health Center](#) 또는 [Node Application Metrics](#)의 내용을 참조하십시오.

로그 파일을 찾을 수 있는 위치에 대한 자세한 정보는 [NODEJSAPP 출력, 로그 및 추적에 대한 위치 제어](#)의 내용을 참조하십시오.

중요사항: 문제점의 원인을 수정할 수 없으면 IBM 지원 센터에 문의하십시오. Node.js 문제점을 보고하기 위해서는 [MustGather](#)에 나열된 필수 정보를 제공해야 합니다.

Node.js -z/OS용 IBM SDK와 관련된 문제점 해결 정보는 [IBM SDK for Node.js - z/OS 문제점 해결](#)의 내용을 참조하십시오.

설치 검증 프로그램(IVP)을 실행하는 데 실패하는 경우

1. MSGUSR 로그를 확인하십시오. CICS 번들 및 NODEJSAPP 파트가 설치되고 사용으로 설정된 경우 CICS 메시지가 여기에 기록됩니다.
2. SYSPRINT 로그를 확인하십시오. Node.js 프로파일이 처리될 때 CICS 메시지가 여기에 기록됩니다.
3. WORK_DIR/APPLID/DFHJNIVP/IVPSAMPLE 디렉토리를 확인하십시오. Node.js 런타임 및 애플리케이션 메시지가 CURRENT.STDOUT 및 CURRENT.STDERR 파일에 기록됩니다. CICS 추적이 사용으로 설정된 경우 CURRENT.TRACE에 기록됩니다.

npm 설치가 Node.js 애플리케이션 종속 항목을 다운로드하는 데 필요한 사이트에 도달하지 못한 경우

getaddrinfo ENOTFOUND nodejs.org nodejs.org:443 오류가 표시될 수 있습니다.

1. `npm -verbose install`에서 리턴된 메시지에서 사이트 TCP/IP 주소를 식별하는 오류가 있는지 확인하십시오(예: `Error: connect ETIMEDOUT 2400:cb00:2048:1::6812:5e60:443`).
2. 사이트에 대체 TCP/IP 주소를 사용해 보십시오. 대체 TCP/IP IPv6 및 IPv4 주소를 나열하려면 `dig registry.npmjs.org -t any` 명령을 사용하십시오. TCP/IP 주소를 사용하도록 npm을 변경하려면 먼저 `npm adduser --registry=https://<ipaddress>` 명령을 사용한 후 `npm install` 명령을 재시도하십시오.
3. 네트워킹 팀에 문의하여 TCP/IP 및 방화벽 구성을 조사하십시오.

NODEJSAPP가 즉시 사용 안함으로 설정되는 경우

stderr에서 CEE5207E The signal SIGABRT was received 메시지를 수신하는 경우 LPAR에서 공유 메시지 큐에 대한 한계에 도달했을 수 있습니다. Node.js 애플리케이션이 SIGKILL 신호로 종료되면 공유 메시지 큐가 할당 해제되지 않을 수 있습니다. 이를 방지하려면 SIGTERM 신호에 대한 응답으로 적시에 애플리케이션이 종료되어야 합니다. 자세한 정보는 [Node.js 애플리케이션 개발](#)의 내용을 참조하십시오.

z/OS 콘솔 명령 D OMVS,L을 사용하고 IPCMSGNIDS를 찾아서 공유 메시지 큐의 수를 확인할 수 있습니다. 공유 메시지 큐를 삭제하려면 ipcrm 명령을 사용하십시오. 자세한 정보는 [ipcrm — 메시지 큐, 세마포어 세트 또는 공유 메모리 ID 제거](#)의 내용을 참조하십시오.

다음과 같은 메시지를 수신하는 경우

- CICS 작업 로그의 CEE0374C CONDITION=CEE3561S TOKEN=00030DE9 59C3C5C5 00000000_00000001 WHILE RUNNING PROGRAM static-initial

- stderr의 CEE3501S The module libnode.so was not found
- 또는 MSGUSR의 DFHSJ1313 E CICSUSER CNJL NODEJSAPP CICSJSON was disabled because an unsupported version of IBM SDK for Node.js - z/OS was used

CICS에서 지원되는 Node.js의 최소 레벨을 사용 중인지 확인하십시오. Node.js 런타임에 대한 경로는 Node.js 파일의 NODE_HOME 옵션에서 지정됩니다.

Node.js 애플리케이션에 대한 추적 활성화 및 관리

SJ 컴포넌트 추적을 커서 Node.js 애플리케이션 추적을 활성화할 수 있습니다. 내부 추적 테이블에는 적은 양의 추적이 기록되지만 Node.js는 zFS에서 Node.js 애플리케이션마다 고유한 파일에 로깅 정보를 기록합니다. 이 파일은 줄 바꾸기가 되지 않습니다. 따라서, zFS에서 해당 크기를 관리해야 합니다.

이 태스크 정보

Node.js 애플리케이션 추적은 보조 또는 GTF 추적을 사용하지 않습니다. CICS는 내부 추적 테이블에 일부 정보를 기록합니다. 그러나 대부분의 진단 정보가 Node.js에서 로깅되며 zFS의 파일에 기록됩니다. 이 파일에는 Node.js 애플리케이션마다 고유한 이름이 지정됩니다. 기본 파일 이름은 `&DATE;.&TIME;.trace` 형식이며 NODEJSAPP 자원을 사용으로 설정할 때 CICS에서 `$WORK_DIR/&APPLID;/&NODEJSAPP;` 디렉토리에 작성합니다. TRACE 프로파일 옵션을 사용하여 이 추적 파일의 이름 및 위치를 변경할 수 있습니다. 프로파일의 변경 사항은 NODEJSAPP 자원이 사용으로 설정될 때 적용됩니다. Node.js 애플리케이션이 실행되는 동안 추적 파일을 삭제하거나 이름을 바꾸는 경우 CICS가 파일을 다시 작성하지 않고 로깅 정보가 다른 파일에 기록되지 않습니다.

프로시저

1. CETR 트랜잭션을 사용하여 Node.js 애플리케이션에 대한 추적을 활성화하십시오. CICS에서 Node.js 애플리케이션을 시작하고 중지하기 위해 수행하는 조치를 추적하려면 SJ 컴포넌트를 선택하십시오. Node.js는 zFS 파일에 진단 정보를 로깅합니다.
2. SJ 컴포넌트에 대한 추적 레벨을 설정하려면 다음을 수행하십시오.
 - SJ 레벨 0은 Node.js 애플리케이션 초기화 중에 발생한 오류와 같은 예외 추적만 생성합니다. SJ 레벨 1 및 레벨 2는 SJ 도메인으로부터 추가 CICS 추적을 생성합니다. 이 추적은 내부 추적 테이블에 기록됩니다.
 - SJ 레벨 3은 Node.js로부터 추가 로깅을 생성합니다(예: 경고 및 정보 메시지). 이 정보는 zFS에서 추적 파일에 작성됩니다.
 - SJ 레벨 4, 5는 Node.js 애플리케이션 처리에 대한 자세한 정보를 제공하는 디버그 정보를 CICS 및 Node.js에서 생성합니다. 이 정보는 zFS에서 추적 파일에 작성됩니다.
3. 각 추적 항목은 날짜와 시간 소인을 갖습니다. TRACE 프로파일 옵션을 사용하여 이 추적 파일의 이름 및 위치를 변경할 수 있습니다.
4. 기본 TRACE 설정을 사용 중인 경우 NODEJSAPP 자원을 사용으로 설정하면 CICS가 Node.js의 수명 동안 새로운 고유 추적 파일을 작성합니다. NODEJSAPP 자원을 사용 안함으로 설정하면 추적 파일을 삭제하거나 정보를 별도로 보관하려는 경우에 파일 이름을 바꿀 수 있습니다.
5. 파일의 수를 관리하기 위해 LOG_FILES_MAX 옵션을 설정하여 Node.js 애플리케이션 시작 시 보존되는 이전 추적 파일의 수를 제어할 수 있습니다.

Node.js 애플리케이션에 대한 CICS 컴포넌트 추적

CICS는 Node.js에서 생성된 로깅뿐만 아니라 0, 1, 2 추적 레벨에 대한 SJ(JVM 및 Node.js 런타임) 도메인에 일부 표준 추적점을 제공합니다. 이러한 추적점은 CICS가 Node.js 애플리케이션을 설정하고 관리하기 위해 수행하는 조치를 추적합니다.

CETR 트랜잭션을 사용하여 0, 1, 2 레벨에서 SJ 도메인 추적점을 활성화할 수 있습니다. SJ 도메인의 모든 표준 추적점에 대한 자세한 정보는 [JVM 및 Node.js 런타임 도메인 추적점](#)의 내용을 참조하십시오.

SJ 컴포넌트 추적

SJ 구성요소는 SJ 도메인의 예외와 처리를 내부 추적 테이블로 추적합니다. SJ 레벨 3, 4 및 5 추적은 zFS의 추적 파일에 기록되는 Node.js 로깅을 생성합니다. 추적 파일의 이름과 위치는 Node.js 프로파일의 TRACE 옵션으로 판별됩니다. 추적 파일에 대한 충분한 공간이 zFS에 있는지 확인하십시오. 추적 활성화 및 관리에 대한 자세한 정보는 [Node.js 애플리케이션에 대한 추적 활성화 및 관리](#)의 내용을 참조하십시오.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. 본 자료는 다른 언어로도 제공될 수 있습니다. 그러나 자료에 접근하기 위해서는 해당 언어로 된 제품 또는 제품 버전의 사본이 필요할 수 있습니다.

IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산을 침해하지 않는 한, 기능상 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

07326

서울특별시 영등포구

국제금융로 10, 3IFC

한국 아이.비.엠 주식회사

대표전화서비스: 02-3781-7114

2바이트(DBCS) 정보에 관한 라이선스 문의는 한국 IBM에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이선스 사용자는 다음 주소로 문의하십시오.

07326

서울특별시 영등포구

국제금융로 10, 3IFC

한국 아이.비.엠 주식회사

대표전화서비스: 02-3781-7114

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이선스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이선스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 인물 또는 기업의 이름과 유사하더라도 이는 전적으로 우연입니다.

저작권 라이선스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 애플리케이션이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 애플리케이션 프로그래밍 인터페이스(API)에 부합하는 애플리케이션을 개발, 사용, 판매 또는 배포할 목적으로 IBM에 추가 비용을 지불하지 않고 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 진술하지 않습니다. 본 샘플 프로그램은 일체의 보증 없이 "현상태대로" 제공됩니다. IBM은 귀하의 샘플 프로그램 사용과 관련되는 손해에 대해 책임을 지지 않습니다.

프로그래밍 인터페이스 정보

CICS에서는 프로그래밍 인터페이스로 간주될 수 있는 몇 가지 문서와 프로그래밍 인터페이스로 간주될 수 없는 몇 가지 문서를 제공합니다.

고객이 CICS Transaction Server for z/OS, 버전 5 릴리스 6 의 서비스를 얻는 프로그램을 작성하는 데 사용할 수 있는 프로그래밍 인터페이스는 온라인 제품 문서의 다음 절에 포함되어 있습니다.

- [애플리케이션 개발](#)
- [시스템 프로그램 개발](#)
- [CICS TS 보안](#)
- [외부 인터페이스용으로 개발](#)
- [애플리케이션 개발 참조서](#)
- [참조: 시스템 프로그래밍](#)
- [참조: 연결성](#)

CICS Transaction Server for z/OS, 버전 5 릴리스 6 의 프로그래밍 인터페이스로 사용하지 않아야 하지만 프로그래밍 인터페이스로 오해할 수 있는 정보는 온라인 제품 문서의 다음 절에 포함되어 있습니다.

- [문제점 해결 및 지원](#)
- [CICS TS 진단 참조](#)

PDF 형식 매뉴얼의 CICS 문서에 액세스하려는 경우 고객이 CICS Transaction Server for z/OS, 버전 5 릴리스 6 의 서비스를 얻는 프로그램을 작성하는 데 사용할 수 있는 프로그래밍 인터페이스는 다음 매뉴얼에 포함되어 있습니다.

- 애플리케이션 프로그래밍 안내서 및 애플리케이션 프로그래밍 참조서
- Business Transaction Services
- 사용자 정의 안내서
- C++ OO 클래스 라이브러리
- 디버깅 도구 인터페이스 참조
- 분산 트랜잭션 프로그래밍 안내서
- 외부 인터페이스 안내서
- Front End Programming Interface 안내서
- IMS 데이터베이스 제어 안내서
- 설치 안내서
- 보안 안내서
- 제공 트랜잭션
- CICSplex® SM 워크로드 관리
- CICSplex SM 관리 자원 사용법
- CICSplex SM 애플리케이션 프로그래밍 안내서 및 애플리케이션 프로그래밍 참조서

- CICS의 Java 애플리케이션

PDF 형식 매뉴얼의 CICS 문서에 액세스하는 경우 CICS Transaction Server for z/OS, 버전 5 릴리스 6 의 프로 그래밍 인터페이스로 사용하지 않아야 하지만 프로그래밍 인터페이스로 오해할 수 있는 정보는 다음 매뉴얼에 포함되어 있습니다.

- 데이터 영역
- 진단 참조
- 문제점 판별 안내서
- CICSplex SM 문제점 판별 안내서

상표

IBM, IBM 로고 및 ibm.com®은 전세계 여러 국가에 등록된 International Business Machines Corp.의 상표 또 는 등록상표입니다. 기타 제품 및 서비스 이름은 IBM 또는 타사의 상표입니다. 현재 IBM 상표 목록은 웹 [저작권 및 상표 정보](http://www.ibm.com/legal/copytrade.shtml)(www.ibm.com/legal/copytrade.shtml)에 있습니다.

Adobe, Adobe 로고, PostScript 및 PostScript 로고는 미국 또는 기타 국가에서 사용되는 Adobe Systems Incorporated의 등록상표 또는 상표입니다.

Apache, Apache Axis2, Apache Maven, Apache Ivy, Apache Software Foundation(ASF) 로고 및 ASF 기능 로고는 Apache Software Foundation의 상표입니다.

Gradle 및 Gradlephant 로고는 미국 또는 기타 국가에서 사용되는 Gradle, Inc.의 등록상표입니다.

Intel, Intel 로고, Intel Inside Inside, Intel Inside 로고, Intel Centrino, Intel Centrino 로고, Celeron, Intel Xeon, Intel SpeedStep SpeedStep, Itanium 및 Pentium은 미국 또는 기타 국가에서 사용되는 Intel Corporation 또는 그 계열사의 상표 또는 등록상표입니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

Red Hat® 및 Hibernate®는 미국 또는 기타 국가에서 사용되는 Red Hat, Inc.의 상표 또는 등록상표입니다.

Spring Boot는 미국 또는 기타 국가에서 사용되는Pivotal Software, Inc.의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

Zowe™, Zowe 로고 및 Open Mainframe Project™는 The Linux Foundation의 상표입니다.

제품 문서의 이용 약관

다음 이용 약관에 따라 이 책을 사용할 수 있습니다.

적용성

본 이용 약관은 IBM 웹 사이트의 모든 이용 약관에 추가됩니다.

개인적 사용

모든 소유권 사항을 표시하는 경우에 한하여 귀하는 이 책을 개인적, 비상업적 용도로 복제할 수 있습니다. 귀 하는 IBM의 명시적 동의 없이 본 발행물 또는 그 일부를 배포 또는 전시하거나 2차적 저작물을 만들 수 없습 니다.

상업적 사용

모든 소유권 사항을 표시하는 경우에 한하여 귀하는 이 책을 귀하 기업집단 내에서만 복제, 배포 및 전시할 수 있습니다. 귀하는 귀하의 기업집단 외에서는 IBM의 명시적 동의 없이 이 책의 2차적 저작물을 만들거나 이 책 또는 그 일부를 복제, 배포 또는 전시할 수 없습니다.

권한

본 허가에서 명시적으로 부여된 경우를 제외하고, 이 책이나 이 책에 포함된 정보, 데이터, 소프트웨어 또는 기타 지적 재산권에 대한 어떠한 허가나 라이선스 또는 권한도 명시적 또는 묵시적으로 부여되지 않습니다.

IBM은 본 발행물의 사용이 IBM의 이익을 해친다고 판단되거나 위에서 언급된 지시사항이 준수되지 않는다고 판단하는 경우 언제든지 이 사이트에서 부여한 허가를 철회할 수 있습니다.

귀하는 미국 수출법 및 관련 규정을 포함하여 모든 적용 가능한 법률 및 규정을 철저히 준수하는 경우에만 본 정보를 다운로드, 송신 또는 재송신할 수 있습니다.

IBM은 이 책의 내용과 관련하여 아무런 보장을 하지 않습니다. 타인의 권리 침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 현 상태대로 제공합니다.

IBM 온라인 개인정보처리방침

서비스 솔루션 소프트웨어를 비롯한 IBM 소프트웨어 제품(소프트웨어 오퍼링)은 제품 사용 정보 수집, 일반 사용자 편의성 향상, 일반 사용자와의 상호작용 조정 및 기타 목적을 위해 쿠키 또는 기타 기술을 사용할 수 있습니다. 많은 경우에 있어서, 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않습니다. IBM의 일부 소프트웨어 오퍼링은 귀하가 개인 식별 정보를 수집하도록 도울 수 있습니다. 본 소프트웨어 오퍼링이 쿠키를 사용하여 개인 식별 정보를 수집할 경우, 본 오퍼링의 쿠키 사용에 대한 특정 정보가 다음에 규정되어 있습니다:

CICSplex SM 웹 사용자 인터페이스(기본 인터페이스)의 경우:

이 소프트웨어 오퍼링은 배치된 구성에 따라 세션 관리, 인증, 사용자 편리성 개선, 기타 사용량의 추적이나 기능적인 용도로 각 사용자의 이름과 개인 정보를 수집하는 세션 및 지속적 쿠키를 사용할 수 있습니다. 이러한 쿠키를 사용하지 못하도록 할 수는 없습니다.

CICSplex SM 웹 사용자 인터페이스(데이터 인터페이스)의 경우:

본 소프트웨어 오퍼링은 배치된 구성에 따라 세션 관리, 인증 또는 기타 사용량 추적이나 기능의 용도로 각 사용자의 사용자 이름 및 개인 식별 정보를 수집하는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키를 사용하지 못하도록 할 수는 없습니다.

CICSplex SM 웹 사용자 인터페이스("hello world" 페이지)의 경우:

배치된 구성에 따라 이 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키를 사용하지 못하도록 할 수는 없습니다.

CICS Explorer의 경우:

배치된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증 및 싱글 사인온 구성을 위해 사용자의 사용자 이름 및 비밀번호를 수집하는 세션 및 지속적 환경 설정을 사용할 수 있습니다. 사인온 중에 선택란을 선택하여 사용자 비밀번호를 암호화된 양식으로 저장하면 사용자의 명시적인 조치에 의해서만 사용으로 설정할 수 있지만 이러한 환경 설정은 사용 안함으로 설정할 수 없습니다.

이 소프트웨어 오퍼링에 대해 배치된 구성이 고객님의 귀하에게 쿠키 및 기타 기술을 통해 일반 사용자로부터 개인적으로 식별 가능한 정보를 수집하는 기능을 제공하는 경우에는 공지사항 및 동의에 대한 요구사항을 포함하여 해당 데이터 콜렉션에 적용할 수 있는 법률에 대한 자체 법률 자문을 구해야 합니다.

해당 용도로 쿠키를 비롯한 다양한 기술을 사용하는 데 관한 자세한 정보는 [IBM 개인정보처리방침](#) 및 [IBM 온라인 개인정보처리방침](#)의 쿠키, 웹 비콘 및 기타 기술 절과 [IBM 소프트웨어 제품 및 SaaS\(Software-as-a-Service\) 개인정보 보호정책](#)을 참조하십시오.

색인

D

DFHAXRO [21](#)

G

GID [18](#)

J

Java

성능 [21](#)

JVM 서버

인클레이브 수정 [21](#)

L

Language Environment 인클레이브

JVM 서버 [21](#)

N

Node.js

설치 확인 [11](#)

Node.js 옵션

기호 [17](#)

Node.js 프로파일

규칙 [13](#)

Node.js 프로파일의 규칙 [13](#)

T

TZ [17](#)

U

UID [18](#)

UNIX 시스템 서비스 액세스 [18](#)

UNIX 파일 액세스 [18](#)

가

그룹 ID(GID) [18](#)

사

사용자 ID(UID) [18](#)

성능

Java [21](#)

시간대

기호 [17](#)

아

액세스 제어 목록(ACL) [18](#)

인클레이브 수정

JVM 서버 [21](#)

타

튜닝

Java [21](#)

