
# OCPP 2.0.1

## 第 4 部分 - JSON over WebSockets 实现指南

FINAL， 2020-03-31

Git Address: https://gitee.com/leven9/OCPP-Documentation

# 免責聲明

# 版本历史

| Version | Date | Author | Description |
|---|---|---|---|
| 2.0.1 | 2020-03-31 | Franc Buve (*OCA*)<br>Milan Jansen<br>(*OCA*)<br>Robert de Leeuw (*ihomer*) | Final version of OCPP 2.0.1 |
| 2.0 | 2018-04-11 | Milan Jansen (*OCA*)<br>Robert de Leeuw (*IHomer*) | OCPP 2.0 April 2018<br>Update for 2.0<br>Section added for OCPP Routing and Signed Messages |
| 1.6 | 2015-10-08 | Patrick Rademakers (*IHomer*)<br>Reinier Lamers (*The New Motion*)<br>Robert de Leeuw (*IHomer*) | Updated to 1.6<br>Asciidoc formatting, remove JSON schema for 1.5Some clarification<br>Added 1.6 json schema |

| | |
|---|---|
| 中文翻译文档版本 | v2023.07.13 |
| 中文翻译文档状态 | 持续修订中 |
| 中文翻译文档发布日期 | 2023-07-13 |

# 1. 介绍

## 1.1. 本文档的目的

本文档的目的是为读者提供创建正确的可互操作 OCPP JSON 实现 （OCPP-J） 所需的信息。我们将根据自己的经验，尝试解释什么是强制性的，什么是好的做法，以及不应该做什么。毫无疑问，误解或模棱两可现象仍将存在，但通过本文件，我们旨在尽可能防止它们。

## 1.2. 目标受众

本文档面向希望以正确且可互操作的方式理解和/或实现 OCPP JSON 的开发人员。假设在服务器或嵌入式设备上实现 Web 服务的基本知识 。

## 1.3. OCPP-S 和 OCPP-J

随着 OCPP 1.6 的引入，OCPP 有两种不同的风格：SOAP 和 JSON。为了避免在实现类型的通信中出现混淆，我们建议使用不同的后缀 -J 和 -S 来表示 JSON 或 SOAP。一般而言，这将是用于 JSON 的 OCPP-J 和用于 SOAP 的 OCPP-S。 特定于版本的术语将是 OCPP1.6J 或 OCPP1.2S。如果没有为 OCPP 1.2 或 1.5 指定后缀，则必须假定 SOAP 实现。从 1.6 版开始，这不再是隐式的，应始终明确说明。如果系统同时支持 JSON 和 SOAP 变体，则标记此 OCPP1.6JS 而不仅仅是 OCPP1.6 被认为是一种很好的做法。

OCPP 2.0.1 仅支持 JSON，但最好继续使用 -J 名称。 作为一种新的传输机制，可能会在 OCPP 的未来版本中引入。所以它将是 OCPP2.0.1J。

## 1.4. 约定

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 1.5. 定义和缩写

| Abbreviation | Description |
|---|---|
| **IANA** | Internet Assigned Numbers Authority (www.iana.org). |
| **OCPP-J** | OCPP communication over WebSocket using JSON. Specific OCPP versions should be indicated with the J extension. OCPP2.0.1J means we are talking about a JSON/WebSocket implementation of 2.0.1. |
| **OCPP-S** | OCPP communication over SOAP and HTTP(S). As of version 1.6 this should explicitly mentioned. Older versions are assumed to be S unless clearly specified otherwise, e.g. OCPP1.5 is the same as OCPP1.5S |
| **RPC** | Remote procedure call |
| **WAMP** | WAMP is an open WebSocket subprotocol that provides messaging patterns to handle asynchronous data. |

## 1.6. 引用

| Reference | Description |
|---|---|
| **[EMI3-BO]** | "eMI3 standard version V1.0" http://emi3group.com/documents-links/ |
| **[OCPP2.0-PART2]** | "OCPP 2.0.1: Part 2 - Specification". http://www.openchargealliance.org/downloads/ |
| **[RFC1951]** | "DEFLATE Compressed Data Format Specification version 1.3". https://www.ietf.org/rfc/rfc1951 |
| **[RFC2119]** | "Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. *http://www.ietf.org/rfc/rfc2119.txt* |
| **[RFC2616]** | "Hypertext Transfer Protocol — HTTP/1.1". http://tools.ietf.org/html/rfc2616 |
| **[RFC2617]** | "HTTP Authentication: Basic and Digest Access Authentication". http://tools.ietf.org/html/rfc2617 |
| **[RFC3629]** | "UTF-8, a transformation format of ISO 10646". http://tools.ietf.org/html/rfc3629 |
| **[RFC3986]** | "Uniform Resource Identifier (URI): Generic Syntax". http://tools.ietf.org/html/rfc3986 |
| **[RFC5246]** | "The Transport Layer Security (TLS) Protocol; Version 1.2". http://tools.ietf.org/html/rfc5246 |
| **[RFC6455]** | "The WebSocket Protocol". http://tools.ietf.org/html/rfc6455 |
| **[RFC7515]** | "JSON Web Signatures (JWS)". https://tools.ietf.org/html/rfc7515 |
| **[RFC7518]** | "JSON Web Algorithms (JWA)". https://tools.ietf.org/html/rfc7518 |
| **[RFC7692]** | "Compression Extensions for WebSocket". https://tools.ietf.org/html/rfc7692 |
| **[RFC8259]** | "The JavaScript Object Notation (JSON) Data Interchange Format". T. Bray. December 2017. https://tools.ietf.org/html/rfc8259 |
| **[WAMP]** | http://wamp.ws/ |

# 2. Benefits & Issues

The WebSocket protocol is defined in [RFC6455]. Working implementations of earlier draft WebSocket specifications exist, but OCPP-J implementations SHOULD use the protocol described in [RFC6455].

Be aware that WebSocket defines its own message structure on top of TCP. Data sent over a WebSocket, on a TCP level, is wrapped in a WebSocket frame with a header. When using a framework this is completely transparent. When working for an embedded system however, WebSocket libraries may not be available and then one has to frame messages correctly according to [RFC6455] him/herself.

# 3. WebSockets

对于使用 OCPP-J 的充电站和充电站管理系统（CSMS）之间的连接，CSMS 充当 WebSocket 服务器，充电站充当 WebSocket 客户端。

## 3.1. 客户端请求

要建立连接，充电站将启动 WebSocket 连接，如 [RFC6455] 第 4 节 "打开握手" 中所述。

OCPP-J 对 URL 和 WebSocket 子协议施加了额外的约束，以下两节 4.1.1 和 4.1.2 将对此进行详细介绍。

客户端（充电站）应始终保持此 WebSocket 连接处于打开状态。

### 3.1.1. 连接 URL

要启动 WebSocket 连接， 充电站需要一个 URL （[RFC3986]） 才能连接。 此 URL 从此以后称为 "连接 URL"。此连接 URL 特定于充电站。充电站的连接 URL 包含充电站标识，以便 CSMS 知道 WebSocket 连接属于哪个充电站。但是，建议 CSMS 不要仅依靠连接 URL 来标识充电站，而是根据其身份验证凭据仔细检查充电站的身份。

支持 OCPP-J 的 CSMS 必须提供至少一个 OCPP-J 端点 URL，充电站应从中派生其连接 URL。此 OCPP-J 端点 URL 可以是具有 "ws" 或 "wss" 方案的任何 URL。充电站如何获取 OCPP-J 终结点 URL 不在本文档的讨论范围之内。

为了派生其连接 URL，充电站修改 OCPP-J 端点 URL ，方法是先在路径上附加一个 "/"（U+002F SOLIDUS），然后附加一个唯一标识充电站。必须根据需要对此唯一标识字符串进行百分比编码，如 [RFC3986] 中所述。

示例 1：对于标识为 "CS001" 的充电站，该充电站连接到 OCPP-J 节点 URL 为 "ws://csms.example.com/ocpp" 的 CSMS，这将提供以下连接 URL：

*ws://csms.example.com/ocpp/CS001*

示例 2：对于标识为 "RDAM 123" 的充电站，该充电站连接到 OCPP-J 节点 URL "wss://csms.example.com/ocppj" 的 CSMS，这将提供以下 URL：

*wss://csms.example.com/ocppj/RDAM%20123*

充电站标识数据类型是标识符字符串（有关定义，请参阅 [OCPP2.0.1-PART2] 此外，可能不使用冒号 ":" 字符，因为唯一标识符还用于基本身份验证用户名。 冒号 ":" 字符用于分隔基本身份验证用户名和密码。充电站标识的最大长度为：48（注意：选择最大长度是为了确保与 [EMI3-BO] "第 2 部分：业务对象" 中的 EVSE ID 兼容。

### 3.1.2. OCPP 版本

OCPP 版本必须在 Sec-Websocket-Protocol 字段中指定。这应该是以下一个或多个值：

*表 1. OCPP Versions*

| OCPP Versions | WebSocket subprotocol name |
|---|---|
| **1.2** | ocpp1.2 |
| **1.5** | ocpp1.5 |
| **1.6** | ocpp1.6 |
| **2.0** | ocpp2.0 |
| **2.0.1** | ocpp2.0.1 |

OCPP 1.2、1.5、1.6、2.0 和 2.0.1 是官方的 WebSocket 子协议名称值。它们在 IANA 注册。

请注意，OCPP 1.2 和 1.5 都在列表中。 由于 JSON over WebSocket 解决方案独立于实际消息内容

该解决方案也可用于较旧的 OCPP 版本。请记住， 在这些情况下，实现最好也应保持对基于 SOAP 的解决方案的支持，以便进行互操作。

将 OCPP 版本作为 OCPP-J 节点 URL 字符串的一部分包括在内被认为是一种很好的做法。如果运行的 Web 服务可以在同一 OCPP-J 节点 URL 上处理多个协议版本，这当然不是必需的。

### 3.1.3. 打开 HTTP 请求的示例

以下是 OCPP-J 连接握手的打开 HTTP 请求的示例：

```
GET /webServices/ocpp/CS3211 HTTP/1.1
Host: some.server.com:33033
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: ocpp2.0.1, ocpp1.6
Sec-WebSocket-Version: 13
```

粗体部分在每个 WebSocket 握手请求中都可以找到 ，其他部分特定于此示例。

在此示例中，CSMS 的 OCPP-J 节点 URL 为 "ws://some.server.com:33033/webServices/ocpp"。 充电站的唯一标识符是 "CS3211"，因此请求的路径变为 "webServices/ocpp/CS3211"。

使用 Sec-WebSocket-Protocol 标头，充电站在此处指示它可以使用 OCPP2.0.1J 和 OCPP1.6J ，并优先选择前者。

此示例中的其他标头是 HTTP 和 WebSocket 协议的一部分，与在第三方 WebSocket 库之上实现 OCPP-J 的协议无关。 这些标头的角色在 [RFC2616] 和 [RFC6455] 中进行了说明。

## 3.2. 服务器响应

收到充电站的请求后， CSMS 必须按照 [RFC6455] 中所述完成握手和响应。 以下 OCPP-J 特定条件适用：

- 如果 CSMS 无法识别 URL 路径中的充电站标识符，则应发送状态为 404 的 HTTP 响应，并按所述中止 WebSocket 连接在 [RFC6455] 中。
- 如果 CSMS 不同意使用客户端提供的子协议之一 ，它必须在没有 Sec-WebSocket 协议标头的情况下完成 WebSocket 握手，然后立即关闭 WebSocket 连接。

因此，如果 CSMS 接受上述示例请求并同意将 OCPP 2.0.1J 与充电站配合使用， CSMS 的响应将如下所示：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: ocpp2.0.1
```

粗体部分在每个 WebSocket 握手响应中都可以找到，其他部分特定于此示例。

Sec-WebSocket-Accept 标头的作用在 [RFC6455] 中进行了说明。

Sec-WebSocket-Protocol 标头指示服务器将在此连接上使用 OCPP2.0.1J。 有关服务器应如何报告 "处理

消息失败" 的定义，请参阅：CALLERROR

## 3.3. WebSockets 压缩

OCPP 2.0.1 支持 RFC 7692：WebSocket 的压缩扩展，请参阅：[RFC6455]

*表 2. WebSocket compression support requirement for devices*

| Device | WebSocket Compression Support |
|---|---|
| Charging Station | Optional |
| CSMS | Required |
| Local Controller | Required |

OCPP 要求 CSMS（和本地控制器）支持 RFC 7692，WebSocket 压缩被视为减少移动数据使用的相对简单的方法。 对于充电站来说，这不是 一个硬性要求，因为在嵌入式平台上实现可能更复杂，但这被认为是有效的。为了减少移动数据使用量，建议在使用移动数据连接的充电站上实施。

RFC 7692 允许充电站和 CSMS 在连接设置期间进行协商。当双方都支持压缩扩展时，他们将在通过线路发送数据时使用 DEFLATE压缩（[RFC1951]）。 当其中一 方不支持它时，JSON 将以未压缩的方式发送（如 OCPP 1.6J）。

当充电站检测到未使用压缩时，建议不要关闭连接，因为在开发，测试期间转动压缩可能非常有用和调试。

有关更多详细信息 ，请阅读 RFC 7692。

# 4. RPC 框架

## 4.1. 介绍

Websocket是一种全双工连接，只需放置一个管道，数据进入，数据可以出来，并且没有明确的进出关系。WebSocket 协议本身不提供将消息作为请求和响应相关联的方法。 为了对这些请求/响应关系进行编码，我们需要在WebSocket之上安装一个小协议。此问题发生在 WebSocket 的更多用例中，因此有现有的方案可以解决它。最常用的是 WAMP（参见 [WAMP]），但该框架的当前版本对称地处理 RPC 不符合 WAMP 标准。 由于所需的框架非常简单， 我们决定定义自己的框架，受到 WAMP 的启发，省略了我们不需要的内容并添加了我们找到的内容。

基本上 ，我们需要的 非常简单：我们需要发送消息（CALL）并接收回复（CALLRESULT）或解释为什么无法正确处理消息（CALLERROR)。为了将来可能的兼容性，我们将保持这些消息的编号与 WAMP 同步。我们实际的 OCPP 消息将被放入一个包装器中，该包装至少包含消息类型、唯一消息 ID 和有效负载（OCPP 消息本身）。

### 4.1.1. 同步性

充电站或 CSMS 不得向另一方发送 CALL 消息，除非其之前发送的所有呼叫消息都已响应或超时。这并不意味着 CSMS 不能向另一个充电站发送消息，在等待第一个充电站的响应时，此规则是按 OCPP-J 连接的。 当收到带有 CALL 消息的消息 ID 的 CALLERROR 或 CALLRESULT 消息 时，已响应 CALL 消息。

在以下情况下，CALL 消息已超时：

- 它没有得到回应， 并且
- 自发送消息以来，依赖于实现的超时间隔已过去 。

实现可以自由选择此超时间隔。 建议他们考虑用于与另一方通信的网络类型。 移动网络的最坏情况往返时间通常比固定线路长得多。

| | |
|---|---|
| **注意** | The above requirements do not rule out that a Charging Station or CSMS will receive a CALL message from the other party while it is waiting for a CALLERROR or CALLRESULT. Such a situation is difficult to prevent because CALL messages from both sides can always cross each other. |

## 4.1.2. 消息有效性和字符编码

由包装器和有效负载组成的整个消息必须是使用 UTF-8（请参阅 [RFC3629]）字符编码的有效 JSON。此外，充电站和CSMS有权拒绝不符合 JSON 架构的消息。

请注意，所有有效的 US-ASCII 文本也是有效的 UTF-8，因此，如果系统仅发送 US-ASCII 文本，则它发送的所有消息都符合 UTF-8 要求。非 US-ASCII 字符应仅用于发送自然语言文本。这种自然语言文本的一个例子是 MessageType，它包含 OCPP 2.0.1 中 DisplayMessage 的文本。

## 4.1.3. 消息类型

要确定消息类型，必须使用以下消息类型之一编号。

*表 3. Message types*

| MessageType | Message Type Number | Description |
|---|---|---|
| CALL | 2 | Request message |
| CALLRESULT | 3 | Response message |
| CALLERROR | 4 | Error response to a request message |

当服务器收到消息类型编号不在此列表中的消息时，它将忽略消息有效负载。每种消息类型可能具有其他必填字段。

## 4.1.4. 消息标识

消息 ID 用于标识请求。任何 CALL 消息的消息 ID 必须与同一发件人以前用于同一 WebSocket 连接上的任何其他 CALL 消息的所有消息 ID 不同。 CALLRESULT 或 CALLERROR 消息的消息 ID 必须等于 CALLRESULT 或 CALLERROR 消息响应的 CALL 消息的 ID 。

*表 4. 唯一消息 ID*

| Name | Datatype | Restrictions |
|---|---|---|
| messageId | string[36] | 唯一消息 ID，最大长度为 36 个字符，以允许 UUID/GUID |

## 4.1.5. JSON Payload

消息的有效负载是一个 JSON 对象，其中包含与操作相关的参数。

如果没有有效负载，JSON 允许使用两种不同的表示法：*null* 或空对象 *{}*。 尽管这看起来微不足道，但我们认为最好只使用空对象语句。Null 通常表示未定义的内容，这与空不同，而且 *{}* 更短。

当某个字段在 OCPP 操作（0..1 或 0..*）中是可选的，并且对于特定请求/响应留空时，JSON 允许使用几种不同的方式来放置它在 JSON 字符串中。 但是由于 OCPP 是为无线链路设计的，因此 OCPP 只允许 1 个选项：不要将字段放在有效负载中（因此不允许空、{} 或 [] 空字段）。

当一个字段的基数为零/一对多（0..* 或 1..*）并且它被赋予了一个实体时，它仍将是一个列表，但大小为 1。

## 4.1.6. Action

CALL 消息中的"Action" 字段必须是不带 "Request" 后缀的 OCPP 消息名称。 例如：对于

"BootNotificationRequest"，操作字段将设置为 "BootNotification"。

顺便说一句：CALLRESULT 不包含操作字段。客户端可以通过 MessageId 字段将响应（CALLRESULT）与请求 （CALL）进行匹配。

### 4.1.7. 消息有效期

消息仅在以下情况下有效：

- Action 是已知的 Action
- JSON 有效负载是有效的 JSON
- Action 的所有必填字段都存在
- 所有数据都属于正确的数据类型

当消息无效时，服务器 SHALL（必须）使用 CALLERROR 响应。

## 4.2. 不同消息类型的消息结构

**注意**

You may find the Charging Station identity missing in the following paragraphs. The identity is exchanged during the WebSocket connection handshake and is a property of the connection. Every message is sent by or directed at this identity. There is therefore no need to repeat it in each message.

### 4.2.1. CALL

A CALL always consists of 4 elements: The standard elements MessageTypeId and MessageId, a specific Action that is required on the other side and a payload, the arguments to the Action. The syntax of a CALL looks like this:

*[<MessageTypeId>, "<MessageId>", "<Action>", {<Payload>}]*

*表 5. CALL Fields*

| Field | Datatype | Meaning |
|---|---|---|
| MessageTypeId | integer | This is a Message Type Number which is used to identify the type of the message. |
| MessageId | string[36] | This is a unique identifier that will be used to match request and result. |
| Action | string | The name of the remote procedure or action. This field SHALL contain a case-sensitive string. The field SHALL contain the OCPP Message name without the "Request" suffix. For example: For a "BootNotificationRequest", this field shall be set to "BootNotification". |
| Payload | JSON | JSON Payload of the action, see: JSON Payload for more information. |

例如，BootNotificationRequest 可能如下所示 ：

```
[2,
 "19223201",
 "BootNotification",
 {
  "reason": "PowerUp",
  "chargingStation": {
      "model": "SingleSocketCharger",
      "vendorName": "VendorX"
  }
 }
]
```

### 4.2.2. CALLRESULT

if the call can be handled correctly the result will be a regular CALLRESULT. Error situations that are covered by the definition of the OCPP response definition are not considered errors in this context. They are regular results and as such will be treated as a normal CALLRESULT, even if the result is undesirable for the recipient.

A CALLRESULT always consists of 3 elements: The standard elements MessageTypeId and MessageId and a payload, containing the response to the *Action* in the original Call.

The syntax of a CALLRESULT looks like this:

*[<MessageTypeId>, "<MessageId>", {<Payload>}]*

*表 6. CALLRESULT Fields*

| Field | Datatype | Meaning |
|---|---|---|
| MessageTypeId | integer | This is a Message Type Number which is used to identify the type of the message. |
| MessageId | string[36] | This must be the exact same ID that is in the call request so that the recipient can match request and result. |
| Payload | JSON | JSON Payload of the action, see: JSON Payload for more information. |

例如，BootNotification 响应可能如下所示 ：

```
[3,
 "19223201",
 {
   "currentTime":  "2013-02-01T20:53:32.486Z",
   "interval": 300,
   "status":  "Accepted"
 }
]
```

## 4.2.3. CALLERROR

我们只在两种情况下使用 CALLERROR：

1. An error occurred during the transport of the message. This can be a network issue, an availability of service issue, etc.
2. The call is received but the content of the call does not meet the requirements for a proper message. This could be missing mandatory fields, an existing call with the same unique identifier is being handled already, unique identifier too long, invalid JSON or OCPP syntax etc.

When a server needs to report a 'failure to process the message', the server SHALL use a Message Type: CallError (MessageTypeNumber = 4).

When a server receives a corrupt message, the CALLERROR SHALL NOT directly include syntactically invalid JSON (For example, without encoding it first). When also the MessageId cannot be read, the CALLERROR SHALL contain "-1" as MessageId.

When a message contains any invalid OCPP and/or it is not conform the JSON schema, the system is allowed to drop the message.

A CALLERROR always consists of 5 elements: The standard elements MessageTypeId and MessageId, an errorCode string, an errorDescription string and an errorDetails object.

The syntax of a CALLERROR looks like this:

*[<MessageTypeId>, "<MessageId>", "<errorCode>", "<errorDescription>", {<errorDetails>}]*

*表 7. CALLERROR Fields*

| Field | Datatype | Meaning |
|---|---|---|
| MessageTypeId | integer | This is a Message Type Number which is used to identify the type of the message. |
| MessageId | string[36] | This must be the exact same id that is in the call request so that the recipient can match request and result. |
| ErrorCode | string | This field must contain a string from the RPC Framework Error Codes table. |
| ErrorDescription | string[255] | Should be filled in if possible, otherwise a clear empty string "". |
| ErrorDetails | JSON | This JSON object describes error details in an undefined way. If there are no error details you MUST fill in an empty object *{}*. |

例如，CALLERROR 可能如下所示 ：

```
[4,
  "162376037",
  "NotSupported",
  "SetDisplayMessageRequest not implemented",
  {}
]
```

## 4.3. RPC Framework Error Codes

下表包含 OCPP RPC 框架允许的所有错误代码。

*表 8. Valid Error Codes*

| ErrorCode | Description |
|---|---|
| FormatViolation | Payload for Action is syntactically incorrect |
| GenericError | Any other error not covered by the more specific error codes in this table |
| InternalError | An internal error occurred and the receiver was not able to process the requested Action successfully |
| MessageTypeNotSupported | A message with an Message Type Number received that is not supported by this implementation. |
| NotImplemented | Requested Action is not known by receiver |
| NotSupported | Requested Action is recognized but not supported by the receiver |
| OccurrenceConstraintViolation | Payload for Action is syntactically correct but at least one of the fields violates occurrence constraints |
| PropertyConstraintViolation | Payload is syntactically correct but at least one field contains an invalid value |
| ProtocolError | Payload for Action is not conform the PDU structure |
| RpcFrameworkError | Content of the call is not a valid RPC Request, for example: MessageId could not be read. |
| SecurityError | During the processing of Action a security issue occurred preventing receiver from completing the Action successfully |
| TypeConstraintViolation | Payload for Action is syntactically correct but at least one of the fields violates data type constraints (e.g. *"somestring": 12*) |

## 4.4. 扩展回退机制

OCPP 在未来版本可能会添加额外的消息类型（除 CALL、CALLRESULT 和 CALLERROR）

当 OCPP 2.0.1 实现收到消息类型未知的消息时，它将使用带有错误代码的 CALLERROR 进行响应：MessageTypeNotSupported。这应该通知发送方有关不支持的消息类型。 然后，发送方应终止连接，或回退到已知的：CALL, CALLRESULT 和 CALLERROR。

# 5. 连接

## 5.1. 数据完整性

对于数据完整性，我们依赖于底层的 TCP/IP 传输层机制。

## 5.2. WebSocket Ping 与 OCPP Heartbeat 的关系

WebSocket 规范定义了用于检查远程终结点是否仍处于响应状态的 Ping 和 Pong 帧。 在实践中，这种机制还用于防止网络运营商在一段时间不活动后悄悄关闭底层网络连接。此 websocket 功能可用作大多数 OCPP 检测信号消息的替代，但不能替换其所有功能。

检测信号响应的一 个重要方面是时间同步。 Ping 和 Pong 帧不能用于此目的，因此建议每天至少发送一条原始检测信号消息，以确保在充电站。

## 5.3. 重新连接

When the connection is lost, the Charging Station SHALL try to reconnect. When reconnecting, the Charging Station SHALL use an increasing back-off time with some randomization until it has successfully reconnected. This prevents an overload of the CSMS when all Charging Stations reconnect after a restart of the CSMS.

The first reconnection attempts SHALL be after a back-off time of: `RetryBackOffWaitMinimum` seconds, plus a random value with a maximum of `RetryBackOffRandomRange` seconds. After every failed reconnection attempt the Charging Station SHALL double the previous back-off time, with a maximum of `RetryBackOffRepeatTimes`, adding a new random value with a maximum of `RetryBackOffRandomRange` seconds to every reconnection attempt. After `RetryBackOffRepeatTimes` reconnection attempts, the Charging Station SHALL keep reconnecting with the last back-off time, not increasing it any further.

When reconnecting, a Charging Station should not send a BootNotification unless one or more of the elements in the BootNotification have changed since the last connection. For the previous SOAP based solutions this was considered good practice but when using WebSocket the server can already make the match between the identity and a communication channel at the moment the connection is established. There is no need for an additional message.

## 5.4. Network node hierarchy

The physical network topology is not influenced by a choice for JSON or SOAP. In case of JSON however the issues with Network Address Translation (NAT) have been resolved by letting the Charging Station open a TCP connection to the CSMS and keeping this connection open for communication initiated by the CSMS. It is therefore no longer necessary to have a smart device capable of interpreting and redirecting SOAP calls in between the CSMS and the Charging Station.

# 6. OCPP Routing

For some topologies it is required to route OCPP-J messages. For example when implementing a Local Controller. This

section contains a solution for OCPP message routing that will work with any Charging Station and CSMS.

## 6.1. 本地控制器

A Local controller is a device that sits between the CSMS and any number of Charging Stations, creating a local group. It is located near to the Charging Station (maybe even connected wired to the Charging Stations), so it does not have problem of losing the connection to the Charging Stations. This is practically useful for doing Local Smart Charging: load balancing between the Charging Stations on the same location. The Local Controller can see all the messages, ongoing transactions etc. It can send charging profiles to the Charging Station to influence the energy used by the Charging Stations, this way preventing the group to use more energy than available at the location at that time.

The Local Controller SHALL work so the Charging Station doesn't have to behave different when connected to the Local Controller, compared to a direct connection to a CSMS. A Local Controller SHALL work so that a Charging Station can work out of the box with the Local Controller, requiring only the parameters that are needed to connect to the Local Controller to be set. The Local ControllerSHALL work so that the CSMS can not notice if the Charging Station is connecting to it directly, or via the Local Controller.

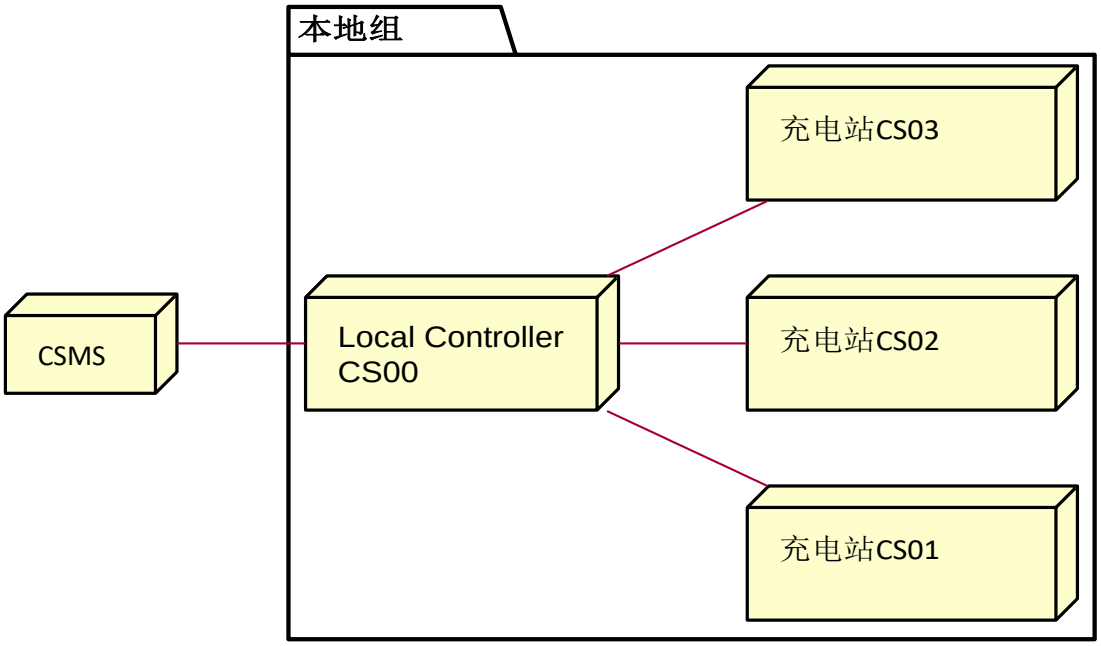


*图 1. Local Controller Topology*

## 6.2. 连接

For each Charging Station that connects to the Local Controller, two connections will be established:

1. A WebSocket connection from the Charging Station to the Local Controller (configured in the Charging Station)
2. A WebSocket connection from the Local Controller to the CSMS (configured in the Local Controller)

Both connections should use a similar connection URI with the same Charging Station identifier. To the CSMS, the connection from the Local Controller appears to be a regular Charging Station connection.

The Local Controller may open a separate WebSocket connection to the CSMS that allows the CSMS to address the Local Controller directly, which may be useful for changing settings or setting overall Charging Profiles.

When a Charging Station connects to the Local Controller, it SHALL connect to it like it would to a CSMS, using the same URI Path in the connection URL as it would use to connect to the CSMS. When the connection between Charging Station and the CSMS is successfully set up, the Local Controller SHALL set up a WebSocket connection to the CSMS with the same URI Path in the connection URL that was used by the Charging Station to setup the connection. The Local Controller SHALL open a WebSocket connection for every Charging Station that connects to it.

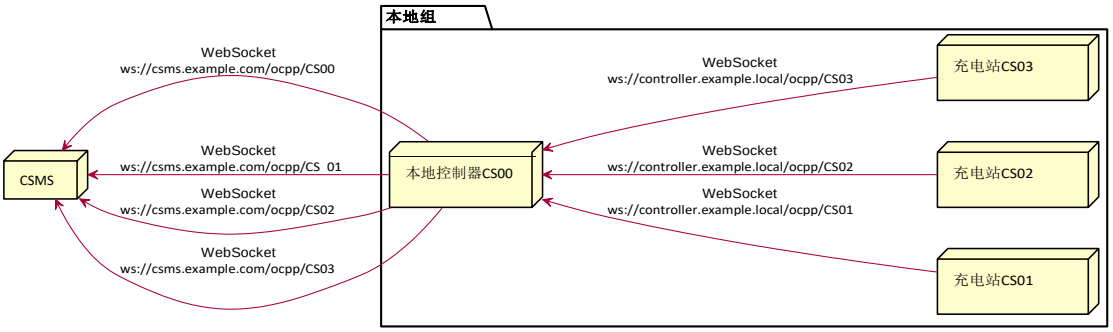


*图 2. Local Controller WebSocket Connections*

# 6.3. 连接丢失

每当 CSMS 和本地控制器之间的一个或多个 WebSocket 连接丢失时，本地控制器应关闭连接到它的充电站的所有相应 WebSocket。这是强制充电站排队消息所必需的，就像它直接连接到 CSMS 并且会失去与该CSMS的连接一样。

每当充电站和本地控制器之间的连接丢失时，本地控制器应关闭其为充电站与 CSMS 建立的 WebSocket 连接。这是让 CSMS 知道充电站处于脱机状态并且无法向其发送任何 CSMS 启动的消息所必需的。

# 6.4. 本地控制器启动的消息

本地控制器应将任何充电站发起的消息中继到 CSMS（反之亦然）。

由于本地控制器还可以向充电站发起自己的消息，因此本地控制器应注意以下事项：

1. 如果本地控制器将自己的消息发送到充电站，则应保证其消息的 ID 不会与 CSMS 使用的 ID 发生冲突，现在和未来。这可以通过将一系列数字分配给要使用的本地控制器（以及要跳过的 CSMS ）或使用 UUID/GUID 来完成。
2. 对来自充电站的消息对本地控制器发起的消息的回复不应发送到 CSMS。

# 6.5. 本地控制器安全性

For the local controller, the normal OCPP security mechanisms will be used, as described in [OCPP2.0.1-PART2], part A. Security. All security profiles described there MAY be used when a Local Controller is deployed. The security section (part A) only describes the roles of the CSMS, and Charging Station. When a local controller is used, the security specification SHALL be interpreted as follows:

- In the connection from the Charging Station to the Local Controller, the Charging Station SHALL act as the Charging Station, and the Local Controller SHALL act as the CSMS. When TLS is used, the Local Controller SHALL be the TLS server, and the Charging Station SHALL be the TLS client.
- In the connection from the Local Controller to the CSMS, the Local Controller SHALL act as the Charging Station, and the CSMS SHALL acts as the CSMS. When TLS is used, the CSMS SHALL be the TLS server, and the Local Controller SHALL be the TLS client.

When TLS with Client Side Certificates is used, the Local Controller SHALL have both a CSMS Certificate, and a Charging Station certificate (see [OCPP2.0.1-PART2] Part A - Keys used in OCPP), as it can function in both roles. These certificates SHALL be unique to the Local Controller. The Local Controller SHALL NOT store the Charging Station certificates of the attached Charging Stations. It SHALL also NOT store the CSMS Certificate of the CSMS. These certificates SHALL be kept private on their respective owners. The Local Controller SHALL only use its own certificates for setting up the TLS connections.

It SHALL be possible to distinguish the Local Controller from the CSMS based on the URL in the CSMS Certificate. Because the Local Controller is placed in the field, there is a risk that its certificates get stolen from it, e.g. by an attack on the hardware. In that case, it SHALL only be possible to use the CSMS Certificate on the Local Controller to communicate with the attached Charging Stations, not with any other Charging Stations in the infrastructure.

The TLS connections terminate on the Local Controller. So, the Local Controller can both read and manipulate data sent between the CSMSs and Charging Stations. If the security of the Local Controller is compromised, it will affect all attached Charging Stations. It is therefore RECOMMENDED to take sufficient security measures to protect the Local Controller. It is als

RECOMMENDED to sign critical commands or replies with the mechanism described in Signed Messages. In this way, it can be detected if the Local Controller tries to manipulate data.

# 7. Signed Messages

For certain architectures it can be useful to use signed OCPP messages. This gives the Charging Station and the CSMS the ability to guarantee that messages are sent by the other party. For example when a Local Controller is involved, the Charging Station can know that a message received from the Local Controller is created and signed by the CSMS.

Message signing can also be used when forwarding data from the Charging Station or the CSMS to 3th parties such as a DSO (Distribution System Operator).

Because message signing is not needed in all architectures and scenarios, it is not required for all OCPP implementations. It will depend on the security requirements if this is required.

This section defines a method to digitally sign any OCPP-J message. For each normal OCPP message an equivalent signed message is defined that encapsulates the normal message, and adds a digital signature.

## 7.1. Signed Message Format

For Signed OCPP Messages, JWS is used. For more information see: [RFC7515].Suppose

we have an OPC calls encoded in the OCPP-J message:

*[<MessageTypeId>, "<MessageId>", {<Extension>}, "<Action>", {<Payload>}]*

Then we define the equivalent signed message as follows.

*[<MessageTypeId>, "<MessageId>", {<Extension>}, "<SignedAction>", {<SignedPayload>}]*

The MessageTypeId and MessageId SHALL stay the same. The <SignedAction> field SHALL be the action name (<Action>) with the string "-Signed" appended. For instance, if "<Action>" is "BootNotification", then "<SignedAction>" is "BootNotification-Signed". The <SignedPayload> SHALL be the JWS encoding of the payload, computed according to the following settings:

- The JWS Payload SHALL be the <Payload> from the original message.
- The JWS Protected Header SHALL contain a field called "OCPPAction" containing the name (<Action>) of the OCPP action, and a field called "OCPPMessageTypedId" containing the message ID (<MessageTypeId>).
- The JWS Protected Header SHOULD contain the x5t#S256 field to identify the key used for signing, as specified in Section 7.4.
- The <SignedPayload> SHALL be encoded using the Flattened JWS JSON Serialization syntax.

## 7.2. 处理已签名的消息

When a Charging Station or CSMS receives a signed message, it SHALL extract the encapsulated normal message. It SHALL process it normally, following the OCPP 2.0.1 standard. It MAY perform additional actions that use the digital signature. This is optional, because a secure connection between the CSMS and the Charging Station is expected, hence a second process to validate a signature on message level is redundant. When a Charging Station receives a signed request, and it supports digital signing, it SHALL send a signed reply.

## 7.3. 允许的算法

The algorithms allowed for use with JSON Web Signatures are defined in the JSON Web Algorithms standard [RFC7518]. To limit the cryptographic algorithms that a Charging Station has to implement, for OCPP the same algorithms SHALL be used as for the TLS connection used to secure communications. This means that for generating the digital signatures, the Charging Station and CSMS SHALL use the following algorithms from the JSON Web Algorithms standard [RFC7518], section 3.1:

- ES256: ECDSA using P-256 and SHA-256
- RS256: RSASSA-PKCS1-v1_5 using SHA-256
- RS384: RSASSA-PKCS1-v1_5 using SHA-384

Note that RS256 and ES256 are the algorithms recommended by [RFC7518].

## 7.4. 密钥管理

This section does not prescribe specific keys to be used for digital signatures. The CSMS Certificate and Charging Station Certificate, used for setting up a secure TLS connection, MAY be used for signing. For many use cases, these will however not be the correct keys. For instance, if the use case is to provide non-repudiation of meter readings, the messages should be signed with a certificate stored in the calibrated measuring chip.

To be able to verify the digital signature, one needs to know which key was used to sign it. JSON Web Signatures supports several ways to store a key identifier with the signed message. As the certificates that can be used for signing are not specified, hash values will be used to identify them. Within the OCPP, the Charging Station and CSMS SHOULD include the field x5t#S256 in the JWS Protected Header to identify the certificate. Following the JSON Web Signatures standard [RFC7515] the value of this field SHOULD be set to the SHA-256 hash of the DER encoding of the signing certificate. How stakeholders can look up the certificate based on the hash value is out of scope for this document.

| | |
|---|---|
| 注意 | In the set up with a Local Controller, described in Local Controller, the TLS client key that would be signing messages to the CSMS will, in fact, not be the TLS client key that the TLS connection is using, since the key in the Local Controller is different from that in the Charging Station. Similarly, the TLS server key signing messages will be that of the CSMS, not that of the local controller. Therefore, implementation of the protocol MUST NOT regardthis mismatch as invalidating the signatures; in fact, it is an expected and desired property to provide end-to-end authenticity. |

# 8. 配置

添加了以下配置变量来控制 JSON/WebSockets 行为：

## 8.1. RetryBackOffRepeatTimes

| | | | |
|---|---|---|---|
| **Required** | yes | | |
| **Component** | **componentName** | OCPPCommCtrlr | |
| **变量** | **变量名称** | RetryBackOffRepeatTimes | |
| | **变量属性** | **易变性** | ReadWrite |
| | **变量特征** | **数据类型** | integer（整数） |
| **描述** | 当充电站重新连接时，在连接断开后，它将使用此变量的次数为之前的回退时间的两倍。当达到最大增量数时，充电站将保持连接，并具有相同的回退时间。 | | |

## 8.2. RetryBackOffRandomRange

| | | | |
|---|---|---|---|
| **Required** | yes | | |
| **Component** | **componentName** | OCPPCommCtrlr | |
| **变量** | **变量名称** | RetryBackOffRandomRange | |
| | **变量属性** | **易变性** | ReadWrite |
| | **变量特征** | **单位** | Seconds（秒） |
| | | **数据类型** | Integer（整数） |
| **描述** | 当充电站重新连接时，在连接断开后，它将使用此变量作为回退时间的随机部分的最大值。它将向每个递增的回退时间添加新的随机值，包括首次连接尝试（使用此最大值），其次数将比之前的回退时间加倍。当达到最大增量数时，充电站将以相同的回退时间保持连接。 | | |

## 8.3. RetryBackOffWaitMinimum

| | | | |
|---|---|---|---|
| **Required** | yes | | |
| **Component** | **componentName** | OCPPCommCtrlr | |
| **变量** | **变量名称** | RetryBackOffWaitMinimum | |
| | **变量属性** | **易变性** | ReadWrite |
| | **变量特征** | **单位** | Seconds（秒） |
| | | **数据类型** | Integer（整数） |
| **描述** | 当充电站重新连接时，在连接断开后，它将使用此变量作为最小回退时间，这是它第一次尝试重新连接时。 | | |

## 8.4. WebSocketPingInterval

| | | | |
|---|---|---|---|
| **Required** | yes | | |
| **Component** | **componentName** | OCPPCommCtrlr | |
| **变量** | **变量名称** | WebSocketPingInterval | |
| | **变量属性** | **易变性** | ReadWrite |
| | **变量特征** | **单位** | Seconds（秒） |
| | | **数据类型** | Integer（整数） |

| | |
|---|---|
| **描述** | 值为 0 将禁用客户端 websocket Ping / Pong。在这种情况下，要么没有 Ping / Pong，要么服务器启动 Ping，客户端用 Pong 响应。<br>正值被解释为 ping 之间的秒数。<br>不允许负值，设置配置将返回 "Rejected" 结果。<br><br>建议将 WebSocketPingInterval 配置得更小：MessageAttemptsTransactionEvent *MessageAttemptIntervalTransactionEvent。这将限制由连接问题触发事务相关消息的重新发送机制的机会。 |

# 9. 自定义数据扩展

在 JSON 架构文件中，所有类都将属性 *otherProperties* 设置为 *false*，以便 JSON 分析器将不接受消息中的任何其他属性。为了允许一些灵活性来创建用于实验目的的非标准扩展，每个 JSON 类都使用 "customData" 属性进行了扩展。 此属性的类型为 "CustomDataType"，它只有一个必需的属性："vendorId"，用于标识自定义类型。但是，由于它没有将*其他属性*设置为 *false*，因此可以使用 新属性自由扩展它。

与为 DataTransfer 消息定义的方式相同，"vendorId" 应是来自反向 DNS 命名空间的值，其中名称的顶层（反向时）应对应于供应商组织的公开注册的主 DNS 名称。

下面的示例显示了检测信号请求的架构定义中的 "CustomDataType" 定义和（可选）"customData" 属性：

```
{
   "$schema": "http://json-schema.org/draft-06/schema#",
   "$id": "HeartbeatRequest",
   "definitions":
       { "CustomDataType":
       {
           "description": "This class does not get 'AdditionalProperties =
false' in the schema
               generation, so it can be extended with arbitrary JSON
properties to allow adding
               custom data.",
           "javaType": "CustomData",
           "type": "object",
           "properties": {
               "vendorId": {
                   "type": "string",
                   "maxLength": 255
               }
           },
           "required": [ "vendorId" ]
       }
   },
   "type": "object",
   "additionalProperties": false,
   "properties": {
       "customData": {
           "$ref": "#/definitions/CustomDataType"
       }
   }
}
```

虽然标准的 HeartbeatRequest 有一个空的主体，一个自定义版本，它提供了主仪表的值和迄今为止所有会话的计数，可能看起来像这样：

```
{
    "customData": {
        "vendorId": "com.mycompany.customheartbeat",
        "mainMeterValue": 12345,
        "sessionsToDate": 342
    }
}
```

已实现此扩展（由其"vendorId"标识）的 CSMS 将能够处理数据。其他 CSMS

实现将简单地忽略这些自定义属性。

A CSMS can request a report of the *CustomizationCtrlr* component to get a list of all customizations that are supported by the Charging Station.