Java WebSocket 规范

穆茂强 张开涛 [译者]

穆茂强的博客：

http://blog.csdn.net/mhmyqn

张开涛的博客：

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

1.引言

本规范定义了一组用于开发 web socket 应用的 Java API。 假定读者已经熟悉了 WebSocket 协议。 WebSocket 协议， 开发为组成 HTML5 技术集的一部分，其有 望带来易于开发和现代的、交互式的应用的网络效率一个新的层次。有关 WebSocket 的更多信息参见

•WebSocket 协议规范 [1]

•JavaScript 的 Web Socket API [2]

1.1.本文档的目的

与 Java WebSocket API 的 API 文档结合的本文档[链接]是 Java WebSocket API 规 范。 本规范定义了一个实现为了是 Java WebSocket API 的一个实现必须满足的要 求。该规范是在 Java 社区进程的规定下制定的。还有测试一个给定的实现是否

满足规范要求的测试兼容性套件（TCK），和实现本规范且通过了 TCK 的参考实现（RI），该规范定义了 WebSocket 应用开发的 Java 标准。虽然本文档中有开发人员使用 Java WebSocketAPI 的很多有用的信息，但其目的不是一个开发人员指南。同样，尽管本文档中有开发人员创建一个 Java WebSocket API 实现的很多有用的信息，但其不是一个关于实现所有必需特性的“怎么做”指南。

1.2.规范的目标

该规范的目标是定义希望在 Java 平台上支持 websocket 编程 API 的容器的要求。虽然该文档可能是一个使用规范定义的 API 的开发人员的有用参考，但该文档不是一个开发人员指南。

1.3.整个规范中使用的术语

1.3.1. 端点

一个 websocket 端点（endpoint）
是一个表示在两个连接的节点（peer）之间的
系列的 websocket 交互的一端的 Java 组件。

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

1.3.2. 连接

一个 websocket 连接（connection）是在两个使用
websocket 协议交互的端点之间 的网络连接。

1.3.3. 节点

使用在一个 websocket 端点上下文中，

websocket 节点（peer）用于表示与该端点
交互的另一个 websocket 参与者。

1.3.4. 会话

术语 websocket 会话（session）
用于表示在一个端点和单个节点之间
的一系列的
websocket 交互。

1.4.规范约定

该文档中的关键字“必须（MUST）”、“不能（MUST NOT）”、“需要（REQUIRED）”、
“应当（SHALL）”、“不得（SHALL NOT）”、
“应该（SHOULD）”、“不应该
（SHOULD NOT）”、“推荐
（RECOMMENDED）”、“可能（MAY）”、和
“可选 的（OPTIONAL）”
由[RFC2119]中的描述解释。
此外，
规范的要求可以使用相符的测试套件进行测试，
该测试套件以紧跟着一个

用于标识要求的数字的数字 WSC（WebSocket 兼容性）标记，例如“WSC-12”。Java 代码和实例数据片段的格式如图 1.1 所示：图 1.1：

示例 Java 代码

```
1  package com.example.hello;
2
3  public class Hello {
4          public static void main(String args[]) {
5                  System.out.println("Hello World");
6          }
7 }
```

一般形式的 URI “http://example.org/...”和“http://example.com/...”表示应用或上下文相关的 URI。

此规范的所有部分是标准的、除了示例、
注释和明确标记为“非标准”的部分。
非标准的注释格式如下所示。

注释：这是一个注释。

1.5.专家组成员

本规范是 Java 社区进程开发的作为 JSR
356[链接]。它是 JSR 356 专家组成员协
同工作的结果。
完整的公共邮件归档可以在这里找到[链接] 。
以下是专家组成
员：

Jean-Francois Arcand (Individual Member) Scott Ferguson (Caucho Technology, Inc) Joe Walnes (DRW Holdings, LLC)

Minehiko IIDA (Fujitsu Limited)
Wenbo Zhu (Google Inc.)
Bill Wigger (IBM)
Justin Lee (Individual Member)
Danny Coward (Oracle)
Rémy Maucherat (RedHat)
Moon Namkoong (TmaxSoft, Inc.)
Mark Thomas (VMware)
Wei Chen (Voxeo Corporation)

1.6.致谢

在开发规范期间，收到了许多审查评论、反馈和建议。尤其感谢：Jitendra Kotamraju、Martin Matula、Stepan Kopriva、Pavel Bucek、Jondean Healey、Joakim Erdfelt。

2.应用

Java WebSocket 应用由 websocket 端点组成。
一个 websocket 端点是一个表示在
两个节点之间的一个 websocket 连接的一端的
Java 对象。

主要有两种方法可以创建一个端点。
第一种方法是实现 Java WebSocket API 特定 的
API 类与所需的行为， 来处理端点生命周期、
消费和发送消息、发布自己、
或连接到一个节点。 通常情况下，
该规范把这种类型的端点叫做编程式端点。 第
二种方法是用 Java WebSocket API
特定的注解来装饰一个普通的旧 Java 对象
（POJO）。
该实现接着获取这些注解的类并在运行时创建相应的对象把

POJO 部 署为一个 websocket 端点。通常情况下，该规范把这种类型的端点叫做注解式端 点。
当讨论这两种端点中任何一种时：
编程式或注解式端点，规范都称为端点。

端点参与打开阶段握手并建立 web socket 连接。端点通常将发送和接收各种 web socket 信息。当 web socket 连接关闭时端点的生命周期将结束。

2.1.API 概述

本节给出了 Java WebSocket API 的简要概述，为之后的详细要求做准备。

2.1.1. 端点生命周期

一个逻辑 websocket 端点在 Java WebSocketAPI 中 由 Endpoint 类的实例表示。 实现必须确保每个 web socket

节点一个实例[WSC 2.1.1-1] 。
开发人员可以子类化 Endpoint 类， 为了拦截端点的生命周期事件：
一个节点建立连接、 一个打开的连接结束和在端点生命周期期间引发的一个错误。

除非另有 由 Java EE 组件支持的不同的生命周期， 否则 websocket 实现必须使用 每个 VM 每个应用一个 Endpoint 类实例来表示每个连接的节点逻辑端点[WSC 2.1.1-2]。第 7 张描述了所有类型的可以用于创建 websocket 端点 的 Java EE 组件 ，
其实现可能会创建一个不同数量的实例来表示逻辑端点。Endpoint 类的每个实例 在通常情况下仅处理来自一个且只有一个节点端点的连接。

2.1.2. 会话

Java WebSocket API 使用一个 Session 类实例模型化在一个端点和它的每个节点

之间的一系列的交互。
在一个节点和一个端点之间的交互以打开通知开始，其次 是一些， 可能是零个在端点和节点之间的 web socket 消息， 其次是终止连接的一个关闭通知或错误。
对于与一个端点交互的每个节点， 有唯一一个 Session 实例 表示该交互[WSC 2.1.1-1] 。对应于那个节点连接的这个 Session 实例 ，在其生命 周期中
的关键事件时传递给表示逻辑端点的端点实例。

http://jinnianshilongnian.iteye.com/

开发人员可以使用通过调用 Session 对象上的 getUserProperties()获取到的与特定
会话的特定应用信息关联的用户属性映射。
Websocket 实现必须维护这个会话数 据 供 以 后 访 问 使用 ， 直到在端 点 实例上调用

onClose()方法完成为止[WSC 2.1.2-2]。在那之后，websocket 实现允许丢弃开发人员数据。一个选择池化 Session 实例的 websocket 实现可能在这一点上重用相同的 Session 实例来表示一个新的连接，并为它分配一个新的唯一的 Session id [WSC 2.1.1-3]。

是分布式容器一部分的 Websocket 实现，在故障转移的情况下，可能需要从一个 节点迁移 websocket 会话到另一个节点。实现必须维护插入到 websocket 会话的开发人员数据对象，如果数据标记为 java.io.Serializable [WSC 2.1.2-4]。

2.1.3. 接收消息

Java WebSocket API

提供了多种方法用于一个端点从它节点的接收消息。开发人 员实现最适合他们需要的消息投递风格的 MessageHandler 接口 的子类型， 且通过在对应于节点的 Session 实例上注册处理器来从一个特定的节点注册感兴趣的消息。

API 限制 了每个 Session 的 MessageHandlers 注册为每个本地的 websocket 消息类 型一个 MessageHandler [WSC 2.1.3-1]。 换句话说，开发人员至多仅能注册一个 传入的文本消息的 MessageHandler、 一个传入的二进制消息的 MessageHandler、 和一个传入的 pong 消 息的 MessageHandler。如果这个违反了这个限制，websocket 实现必须产生一个错误[WSC 2.1.3-2]。

规范的未来版本可能会取消这个限制。

2.1.4. 发送消息

Java WebSocket API 使用一个端点的 RemoteEndpoint 类的实例模型化会话的每个节点。该类包含了用于从端点到它的节点发送 web socket 消息的多种方法。

示例

下面是一个服务器端点等待传入的文本消息的一个示例，且当它获得到发送它的
客户端的一个消息时立即响应。
这个示例是重复的，第一种方法仅使用 API 类：

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

```
public class HelloServer extends Endpoint {
    @Override
    public void onOpen(Session session) {
        final RemoteEndpoint remote = session.getRemote();
        session.addMessageHandler(new MessageHandler.Basic<String>() {
            public void onMessage(String text) {
                try {
                    remote.sendString("Got your message (" + text + "). Thanks !");
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
            }
        });
    }
}
```

第二种方法仅使用 API 中的注解：

```
@WebSocketEndpoint("/hello")
public class MyHelloServer {
    @WebSocketMessage
    public String doListen(String message) {
        return "Got your message (" + text + "). Thanks !";
    }
}
```

2.1.5. 关闭连接

如果一个到 websocket
端点打开的连接因为一些原因要被关闭，
无论是 由于从节 点 收到一个 websocket
关 闭事件的结果 ，
还是因为底层实现的原因关闭连接，
websocket 实现必须调用 websocket 端点 的
onClose()方法[WSC 2.1.5-1]。
待定 （TBD） 已经有一些反馈：
onClose()应该仅被调用 当节点接收到一个来 自
另一个节点的关闭帧 （close frame） 时，
和不应该， 例如， 假如实现的宿主端点
由于某些原因（例如，当取消部署应用时）需要
自 己关闭连接。 当连接由容器关 闭时，
在这方面我们欢迎任何反馈。
建议 websocket 开发人员使用其他一些方法
（例如使用 EJB 支持 websocket） 来
拦截生命周期事件。

2.1.6. 客户端和服务器

Websocket 协议是双向协议。一旦建议连接，web socket 协议在会话中的两方之间是双向的。在 websocket 客户端和 websocket 服务器之间区别只在于双方通过何种方式连接。在本规范中，我们会说，一个 websocket 客户端是一个初始化一个到一个节点的连接的 websocket 端点。我们会说，一个 websocket 服务器是一个发布一个等待来自节点连接的 websocket 端点。在大多数部署中，一个 websocket 客户端将仅连接到一个 websocket 服务器，且一个 websocket 服务器将接受来自不同客户端的连接。

因此，WebSocket API 仅辨别端点之间的区别是：来自端点的是

websocket 客户端，在配置和安装阶段中的是 websocket 服务器。

2.2.使用 WebSocket 注解的端点

Java 注解 已经成为广泛使用 的为 Java 对象增加部署特性的手段，尤其在 Java EE 平台中。Web Socket 规范定义了少量的 web socket 注解，允许开发人员获取 Java 类并把它们变成 web socket 端点。本节

本节给出了简要概述，
为本规范之后的详细要求做准备。

2.2.1. 注解式端点

类级别的 @WebSocketEndpoint 注解表示一个 Java 类是在运行 时变成一个

websocket 端点。 开发人员可以使用 value 属性为端点指定一个 URI 映射。encoders 和 decoders 属性允许开发人员指定编码应用对象到 web socket 消息和解码 web socket 消息到应用对象的类。

2.2.2. Websocket 生命周期

方法级别的@WebSocketOpen 和@WebSocketClose 注解允许开发人员在他们的 @WebSocketEndpoint 注解的 Java 类中装饰方法， 来指定分别当产生的端点从一个节点接收到一个新的连接时或当从一个节点的一个连接关闭时，它们必须被实 现回调[WSC 2.2.2-1]。

2.2.3. 处理消息

为了使注解的端点能处理传入的消息，
方法级的@WebSocketMessage 注解允许
开发人员表示当收到一个消息时实现必须调用哪一个方法[WSC 2.2.3-1]。

http://jinnianshilongnian.iteye.com/

2.2.4. 处理错误

为　了使注解的端点能处理外部事件引起的错误，
例如在解码一个收到的消息，一 个注解的端点可
以使用　　@WebSocketError
注解标记它的一个方法必须被实现调
用并带有这样一个错误发生时的错误消息，
[WSC 2.2.4-1]。

2.2.5. Ping 和 Pong

在 websocket 协议中 的 ping/pong 机制作为检查连接是否仍处于活动状态。 以下是协议的要求， 如果一个 websocket 实现接收自 一个节点传入的 ping 消息， 它必 须尽 可能快地用一个包含相同应用数据的 pong 消息响应那个节点[WSC 2.2.5-1]。希望发送单向 pong 消息的开发人员可以使用 RemoteEndpoint API 做这 件 事情 。 希 望 监 听 返 回 的 pong 消 息 的 开 发 人 员 可 以 为 它 们 定 义 一 个 MessageHandler 或使用 @WebSocketMessage 注解一个方法， 该方法规定一个 PongMessage 作为它的消息实体参数。 在这两种情况下， 如果实现收到一个发送 到该端点的 pong 消息， 它必须调用 MessageHandler 或注解的消息方法[WSC 2.2.5-2]。

http://jinnianshilongnian.iteye.com/

3.配置

WebSocket 应用配置了一些关键参数：在容器
URI 空间中标识一个 web socket
端点的路径映射、端点支持的子协议、
应用程序需要的扩展。此外，在打开阶段
握手期间，应用可以选择执行其他的配置任务，
如检查发出请求的客户端主机名，或处理 cookie。
本节详细介绍了在容器上支持这些配置任务的要求。
客户端和服务器端点配置包括一系列应用提供的编码和解码类，
实现必须使用它 们在 websocket
消息和应用定义的消息对象之间转换[WSC 3-1]。
下面是服务器特定的和客户端特定的配置选项定义。

3.1.服务器配置

为了部署一个编程式端点到对客户
端连接可用的 URI 空间，容器需要一个
ServerEndpointConfiguration 实例。
WebSocket API 提供了该接口的一个默认实现
[WSC-3.1-1]。
这些配置对象使用默认的策略来协商打开阶段握手，
并将建立（或不建立）每一个 websocket 连接。

3.1.1. URI 映射

默认服务器配置的 URI
映射策略是打开阶段握手 URI
通过以下方式匹配一个端 点的相对路径：

如果端点的相对路径是一个相对 URI，
当且仅当有一个 URI 完全匹配[WSC
3.1.1-1]，
或

相对路径是一个请求 URI 匹配的 URI 模板（level-1），当且仅当请求 URI 是

一个 URI 模板的展开版本。[WSC 3.1.1-2]

实现必须不建立连接,除非有一个匹配[WSC 3 1 1 3]。

3.1.2. 子协议协商

默认服务器配置必须在创建时提供按优先顺序排列的一系列支持的协议。

在子协 议协商期间，

此配置检查客户端提供的子协议列表，

并选择它支持的包含在客户 端提供的列表中

的第一个子协议，

或者没有如果没有匹配的[WSC-3.1.2-1]。

3.1.3. 扩展变更

在打开阶段握手时，

客户端提供了一系列它想使用的扩展。

默认服务器配置从这些扩展中选择一个它支持的，并以客户端所请求的相同顺序放置它们[WSC-3.1.3-1]。

3.1.4. 来源检查

默认服务器配置检查 Origin 头提供的主机名，

如果主机名不能被验证则握手失败[WSC-3.1.4-1]。

3.1.5. 握手变更

默认服务器配置除了上边描述的并没有改变打开阶段握手处理[WSC-3.1.5-1]。

开发人员可能希望定制上述规定的配置和握手协商策略。

为了做到这一点，他们可能会提供他们自己的 ServerEndpointConfiguration 实现，

或者通过直接实现它，或者子类化默认实现。例如，他们可能希望更多地介入握手处理。他们可能希望使用 Http Cookie 来跟踪客户端，或在握手响应中插入应用特定的头。为了做到这一点，他们可能覆盖 ServerEndpointConfiguration 的 modifyHandshake()方法，在那里他们可以有握手的 HandshakeRequest 和 HandshakeResponse 的全部访问权限。他们可能希望提供一个更复杂的方法来映射 URI 到端点，例如通过 web socket 注解的基于 shceme 的 URI-模板（参见第 4 章 注解）。

3.1.6. 不同实例间的状态共享

开发人员可以实现 ServerEndpointConfiguration 以保存应用状态，这将是非常有用的对所有

Endpoint 实例同一个逻辑端点的情况。

3.2.客户端配置

为了连接一个 websocket 客户端端点到它对应的 websocket 服务器端点，实现需 要配置信息。除了编码器和解码器列表，Java WebSocket API 需要以下属性：

3.2.1. 子协议

默认客户端配置使用开发人员提，它想要使用的供的子协议列表，按照优先顺序 来发送，在打开阶段握手中想要使用的它制定的子协议的名字[WSC-3.2.1-1]。**3.2.2.**

扩展

默认客户端配置必须使用开发人员提供的扩展列表来发送，
按照优先顺序来发 送，
在打开阶段握手中它制定的它想要使用的扩展，
包括参数。**3.2.3. 客户端配置变更**

一些客户端可能希望以适应的方式在客户端应用
中提供子协议列表或扩展列表，
或打开阶段握手的客户端与服务器交互的其他方面。
开发人员可能提供他们自己 的覆盖默认 API
行为的 ClientEndpointConfiguration 实现，
为了对其进行定制以 满足特定应用的要求。

4.注解

本节包含了一个在 Java WebSocket API 中的注解语义的完整规范。

4.1.@WebSocketEndpoint

这个类级别的注解表示它装饰的 Java 类必须被实现部署为一个 websocket服务器端点并使其在 web socket 实现的 URI-空间中可用[WSC-4.1-1]。 该类必须有一个 public 无参构造器， 且另外可能符合第 7 章中列出的类型之一。

4.1.1 value

Value 属性必须是一个 Java 字符串 ，

其是一个部分 URI 或 URI-模板（level-1），以 “/” 开头。 URI-模板的定义， 请参见[6]。实现使用 value 属性来部署端点到 web socket 实现的 URI 空间。实现必须把 value 作为相对于 web socket 实现的根 URI，用来确定是否与一个传入 的打 开阶段握手 的请求 的请求 URI 匹配 [WSC-4.1.1-2]。如果 value 是一个部分 URI，请求 URI 匹配，当且仅当精确匹配。如果 value 是一个 URI 模板 （level-1 ），请求 URI 匹配， 当且仅当请求 URI 是一 个 URI 模板的扩展版本。 Value 属性是强制的，在部署时， 实现必须拒绝缺少或残缺的路径[WSC-4.1.1-3]。

例如，

```
@WebSocketEndpoint("/bookings/{guest-id}")
public class BookingServer {
   @WebSocketMessage
   public void processBookingRequest(
      @WebSocketPathParam("guest-id") String guestID,
      String message,
      Session session) {
    // process booking from the given guest here
   }
}
```

在这种情况下，客户端可以使用任意 URI 连接到这个端点：

/bookings/JohnSmith
/bookings/SallyBrown
/bookings/MadisonWatson

然而，
如果端点注解是@WebSocketEndpoint("/bookings/SallyBrown")，
那么只有
一个客户群请求到/bookings/SallyBrown
将能够连接到这个 web socket 端点。

如果在 value 属性中使用 URI-模板，
开发人员可以使用@WebSocketPathParameter
取出变量路径片段，如下所述。

包含多于一个注解端点的应用可能不经意地使用相同的相对 URI。 Websocket 实现在部署时必须拒绝这样一个应用 ，并带有一个他不能解决的重复路径的错误提示消息[WSC-4.1.1-4]。

应用可能包含一个映射到一个路径的端点，该路径是 URI 模板的扩展形式， 该路径也被相同应用中的另一个端点使用。
在这种情况下，应用是有效的。在运行 时，如果

websocket 实现接收到一个精确匹配该相对 URI 的打开阶段握手，它必须匹配那个，即使在别处它是一个 URI-模板的扩展形式[WSC-4.1.1-5]。

规范的未来版本可能会允许更高级的 URI 模板。

4.1.2. encoders

encoders 属性包含一个（可能是空）为该端点充当编码器组件的 Java 类列表。这些类必须实现某种形式的 Encoder 接口，并具有 public 无参构造器和在这个 websocket 端点所在的应用的 classpath 内是可见的。实现必须按照编码器尝试编码匹配参数化类型的应用对象当它们试图使用 RemoteEndpoint API 发送时。实现必须使用第一个匹配列表中类型的编码器[WSC-4.1.2-2]。

4.1.3. decoders

decoders 属性包含一个 （可能是空）为该端点充当解码器组件的 Java 类列表。这些类必须实现某种形式的 Decoder 接口，并具有 public 无参构造器和在这个 websocket 端点所在的应用的 classpath 内是可见的。实现必须尝试使用列表中适 合于本地 websocket 消息类型的解码器解码 web socket 消息，并以解码的对象形 式传递消息到 webscoket 端点[WSC-4.1.3-1]。 在拥有它的 Decoder 实现上， 实现 必须使用解码器的 willDecode()方法来决定是否 Decoder 将匹配传入的消息[WSC-4.1.4-2]。

4.1.4. subprotocols

subprotocols 参数包含一个 （可能为空） 该端口支持的子协议名字的字符串列表。

实现必须在打开阶段握手使用
该列表来协商用 于建立连接的所需 的子协议
[WSC-4.1.4-1]。

http://jinnianshilongnian.iteye.com/

4.1.5. 配置

该可选的配置属性使开发人员能表明他想要的
websocket 实现使用一个定制的
DefaultServerConfiguration 子类 。 如果提供 了
一个， websocket 实现必须使用这个
来配置端点[4.1.5-1]。
开发人员可能使用这个技术在端点的不同实例间状态共享，
除了定制的打开阶段握手。

待定（TBD）：

可能还添加在注解中指定扩展的能力？

4.2.@WebSocketClient

这个类级别的注解表示它装饰的 Java 类被部署为一个将连接到驻留在 web socket 服务器的 web socket 端点的 websocket 客户端端点。该类必须有 public 无参构造器，且另外可能符合第 7 章中列出的类型之一。

4.2.1. encoders

encoders 属性包含一个（可能是空）为该端点充当编码器组件的 Java 类列表。这些类必须实现某种形式的 Encoder 接口，并具有 public 无参构造器和在这个 websocket 端点所在的应用的 classpath 内是可见的。实现必须按照编码器尝试编码匹配参数化类型的应用对象 当它们试图使用

RemoteEndpoint API 发送时。 实现必须使用第一个匹配列表中类型的编码器[WSC-4.1.2-2]。

4.2.2. decoders

decoders 属性包含一个 （可能是空）为该端点充当解码器组件的 Java 类列表。 这些类必须实现某种形式的 Decoder 接口，并具有 public 无参构造器和在这个 websocket 端点所在的应用 的 classpath 内 是可见的 。实现必须尝试使用列表中适 合于本地 websocket 消息类型的解码器解码 web socket 消息，并以解码的对象形 式传递消息到 webscoket 端点[WSC-4.2.2-1] 。 如果 Decoder 实现有这个方法， 实 现必须使用解码器的 willDecode()方法来决定是否 Decoder 将匹配传入的消息[WSC-4.2.2-2]。

4.2.3. subprotocols

subprotocols 参数包含一个 （可能为空） 该端口将支持的子协议名字的字符串列 表。实现必须在打开阶段握手使用该列表来协商用于建立连接的所需的子协议[WSC-4.2.3-1]。

待定（TBD）：

我们可能添加在注解中指定扩展的能力。

4.3.@WebSocketPathParam

这个注解 用 于 在一个装饰着任何@WebSocketMessage、@WebSocketError、@WebSocketOpen、 @WebSocketClose注解的带注解的端点上注解一个或多个方

法参数。 这些参数允许的类型是 String、 Java 原子类型、 或它的包装版本。 任何
使用该注解的其他类型的注解是错误的，
实现必须在部署时报告[WSC 4.3-1]。
这个注解的 value 属性必须存在，
否则实现必须抛出一个错误[WSC 4.3-2]。 如果
这个注解的 value
属性匹配一个在@WebSocketEndpoint
注解的那个端点上使用 URI
模板的一个元素的变量名，
那么实现必须在方法被调用时把调用 web socket
帧连接的请求 URI
的路径片段值关联到它注解的 value 参数。[WSC 4.3-3]。否则，使用这个注解注解的 String 参数值，
实现必须设置为 null。关联必须遵循下列规 则：

如果参数是 String，
容器必须使用路径片段的值[WSC 4.3-4]

如果参数是一个 Java 原子类型或它的装箱版本，

容器必须使用路径片段字符
串来构造类型相同的结果，假如该类型有 public
的一个 String 参数的构造器 来获得装箱类型，
且如果必要降低为原子类型[WSC 4.3-5]。

如果容器不能适当地解码路径片段为注解
的路径参数 ， 那么 容器必须发 出
DecodeException 异 常 到 web socket
包含的路径片段的错误处理方法[WSC 4.3-6]。

例如，

```
@WebSocketEndpoint("/bookings/{guest-id}")
public class BookingServer {
    @WebSocketMessage
    public void processBookingRequest(
        @WebSocketPathParam("guest-id") String guestID,
        String message,
        Session session) {
        // process booking from the given guest here
    }
}
```

在这个例子中，如果客户端使用 URI /bookings/

JohnSmith 连接到这个端点，那么 guestID 参数的值将为“JohnSmith”。

下面是一个关于路径参数是一个 Integer 的例子：

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

```
@WebSocketEndpoint("/rewards/{vip-level}")
public class RewardServer {
    @WebSocketMessage
    public void processReward(
        @WebSocketPathParam("vip-level") Integer vipLevel,
        String message,
        Session session) {
        // process reward here
    }
}
```

4.4.@WebSocketOpen

该注解是一个方法级别注解。
当一个新的客户端连接到这个端点时该注解定义的包装方法将被调用。
在连接建立后容器通知该方法[WSC-4.4-1]。
包装的方法仅能有一个可选的 Session 参数、一个可选的 EndpointConfiguration 参数和零个到 N 个@WebSocketPathParam 注解的 String 参数。如果 Session 参数存在，实现必须传入与新的连接关联的最新创建的 Session[WSC-4.4-2]。有一个方法上使用了该注解的任何 Java 类没有遵循这些规则，可能不被实现部署并向部署人员报告错误[WSC-4.4-3]。

4.5.@WebSocketClose

该注解是一个方法级别注解。
当一个新的客户端从该端点断开时，该注解定义的保证方法将被调用。

在连接被关闭之前容器通知该方法[WSC-4.5-1]。
包装的方 法仅能有一个可选的 Session 参数和零个到 N 个@WebSocketPathParam 注解的 String 参数。 如果 Session 参数存在，实现必须传入与连接关联的即将结束的 Session[WSC-4.5-2]. 。
如果该方法本身抛出 了 一个错误，实现必须把这个错误连 同会话一起传递到端点的 onError()方法[WSC-4.5-3] 。
有一个方法上使用 了该注 解的任何 Java 类没有遵循这些规则，可能不被实现部署并向部署人员报告错误 [WSC-4.5-4]。

4.6.@WebSocketError

该注解是一个方法级别注解。
该注解定义了每当到这个端点的任何连接生成了 个错误时被调用 的包装方法。
该包装方法仅能有可选的 Session 参数，

必须的 Throwable 参数和零个到 N 个@WebSocketPathParam 注解的 String 参数。如果有 Session 参数，
实现必须传入出现错误的连接的 Session[WSC-4.6-1]。 容器必须 以 Throwable 参数传入错误到这个方法。 [WSC-4.6-2]
使用该注解注解在一个方法 上的任何 Java 类如果不遵守这些规则，
可能不能被实现部署且报告错误给部署 人员 。[WSC-4.5-3]

4.7.@WebSocketMessage

该注解是一个方法级别的注解。
当接收到一个传入的消息时该注解定义的包装方

法将被调用。它装饰的方法可以具有许多形式：它的方法参数可以包括

a）任何下列选择之一

- **String** 用于整个文本消息处理
- **String** 和 **boolean** 对 用于部分文本消息处理
- **byte[]**或 **ByteBuffer**

用于整个二进制消息处理

- **byte[]**和 **boolean** 对，或 **ByteBuffer** 和 **boolean** 对 用于部分二进制消息处理
- 任何可解码的对象参数(端点配置的 **Decoders** 确定的)
- **PongMessage** 用于处理 pong 消息

和

b) 零个到 N 个使用@WebSocketPathParam 注解的 String 或 Java 原子类型参数 c) 可选的 **Session** 参数

这些参数可以以任何顺序列出。
该方法可以有一个非-void 的返回值类型，
在这种情况下，web socket 运行时必须
解释这个为 web socket 消息并返回给节点。
此返回值类型允许的数据类型，除了 void，是
- String、ByteBuffer、byte[]、任何 Java
原子类型或等价的类和有一个解码器
的任何类型。
任何 @WebSocketMessage 注解的方法，
不符合上边描述的形式是无效的。websocket
实现必须不部署这样的端点，
并如果尝试部署这样注解的端点，必须
发出一个部署错误[WSC-4.7-1]。
每一个 websocket
端点可能仅有一个消息处理方法用于每个本地

webscoket 消息 格式：文本、二进制和 pong。
Websocket 实现必须不部署这样的端点，
且如果尝 试部署这样一个注解的端点，
必须发出一个部署时错误[WSC-4.7-2]。

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

maxMessageSize

maxMessageSize
属性允许开发人员指定它注解的方法将能处理的以字节为单位
的最大大小的消息， 或-1 表示不限制。
默认是-1。
如果传入的消息超过最大消息大小，
实现必须报告这是一个错误[WSC-4.7-3]。

http://jinnianshilongnian.iteye.com/

5.异常处理和线程

WebSocket API
实现可以采用多种线程策略以提供一个可伸缩的实现。该规范目的是允许一系列的策略。然而，该实现必须满足一定的线程要求，以便为开发人员提供一个用于它们应用的一致的线程环境。

除非 Java EE 组件支持不同的生命周期（参考第 7 章），否则容器必须每个节点使用一个唯一的端点实例[WSC-5.1-1]。
在所有情况下，实现必须不在每个节点的多个线程中

同时调用一个端点实例[WSC-5.1-2]。
实现可能不调用在一个端点 上的 close 方法直到 open 方法完成之后[WSC-5.1-3]。

这个可以保证一个 websocket
端点实例是从不被每个节点的多个容器线程同时调用[WSC-5.1-4]。

如果 以片段形式收到一个传入的消息，容器必须确保相关的 onMessage()调用是按顺序调用的，且既没有与相同消息的片段交错也没有与其他消息交错[WSC-5.1.5]。

待定（TBD）：可能需要为
onMessage()定义一些关于容器创建回调线程和他们如何与用于（异常）发送的回调线程交互。

5.2.错误处理

该规范定义有三类错误（受查和非受查 Java 异常）。

5.2.1. 部署错误

这是是在部署一个包含 websocket 端点的应用期间引起的错误。 这些错误有些是在部署应用期间容器的故障引起的结果。 例如，容器可能没有足够的计算资源来 部署指定的应用。 在这种情况下，容器在部署过程期间必须提供一个错误提示消息给开发人员[WSC-5.2.1-1]。 其余的错误，由残缺的 websocket 应用引起的结果。 第 4 章提供了几个残缺的 websocket 端点的例子。 在这种情况下，容器在部署过程期间必须提供一个错误提示消息给部署人员[WSC-5.2.1-2]。

在这两种情况下，

在部署过程期间引起的一个部署错误必须停止应用的部署，在引起错误之前部署的任何形式良好的端点必须从服务中移除且不再有来自应用的 websocket 端点可以被容器部署，即使他们是有效的[WSC-5.2.1-3]。

如果部署错误发生在开发人员编程控制下，例如，当时有 WebSocketContainer API 部署一个客户端端点，部署错误必须由容器通过使用一个 DeploymentException 实例报告给开发人员[WSC-5.2.1-4]。在这种情况下，容器可以选择用语精确的错

http://jinnianshilongnian.iteye.com/

误消息。

如果部署错误发生在实现管理的部署期间，例如，由于部署一个容器部署端点所在的 WAR 文件由于扫描 WAR 文件，作为容器特定的部署过程的一部分，部署错误必须由实现被报告给部署人员[WSC-5.2.1-5]。

5.2.2. websocket 应用代码产生的错误

在一个 websocket 端点运行期间引起的所有错误必须被 websocket 实现捕获[WSC-5.2.2-1]。这些错误的例子包括端点使用的 Decoders 生成的受查异常，端点使用的消息处理代码产生的运行时错误。如果 websocket 端点提供了一个错误处理方法，无论是在编程式端点的情况下通过实现 onError()方法，还是在注解式端点情况下使用

@WebSocketError 注解， 实现必须以错误调用错误处理方法[WSC-5.2.2-2]。

如果开发人员没有在产生错误的端点上提供一个错误处理方法，这表示实现， 开 发人员不希望处理这样的错误。在这些情况下， 容器必须使这个信息可供以后分析，例如通过记录它[WSC-5.2.2-3]。

如果一个端点本身
的错误处理方法产生运行时错误，
容器必须使这个信息可供以
后分析[WSC-5.2.2-4]。

待定（TBD）：
是否他们希望容器应该允许取消部署生成太多错误的应用？ 在这 方面我们欢迎反馈。

5.2.3.

容器和/或底层连接产生的错误

在端点运行期间可能出现各种运行时错误。
这些可能包括中断的底层连接、 偶尔
的通信错误处理传入和传出消息、
或与节点通信的致命错误。 实现或它们的管理
员判断这样的错误是否是致命的 ，
正确运行的端点可能关闭端点连接， 使用
onClose()方法尝试通知两个参与者。
容器判断这样的错误是否是非致命的， 正确
运行的端点可能允许端点继续运行，
但必须在消息处理期间报告错误， 或者作为
检查异常由一个发生操作返回，
或者记录错误用于以后分析[WSC-5.2.3-1]。

http://jinnianshilongnian.iteye.com/

6.打包和部署

Java WebSocket 应用使用 Java
平台通常的约定打包。

6.1.JDK 上的客户端部署

用于 web
socket 应用的类文件和任何如 Java WebSocket 客户端应用的应用资源被
打包为 JAR 文件，
除了如文本或图像文件的任何资源之外它需要的
。

客户端容器不需要自动扫描 JAR 文件寻找 web
socket 客户端端点并部署它们。

使用 ContainerProvider 类获得一个到
WebSocketContainer 的引用，开发人员使用

WebSocketContainer 上的 connectToServer() API 部署编程式端点和注解式端点。

6.2.Web 容器上的应用部署

类文件和任何如文本和图像文件的资源被打包进 Java EE 定义的 WAR 包，或者 直接位于 WEB-INF/classes 下或打包为一个 JAR 文件并位于 WEB-INF/lib 下。

Java WebSocket 实现必须使用定义在[Servlet 3.0] 中的 web 容器扫描机制在部署时来发现包含在 WAR 文件中的注解式和编程式端点[WSC-6.2-1]。此外，websocket 实现必须使用 web socket 扫描机制来发现打包在 WAR 文件 （或在任何它的子 JAR 文件）中的 ServerApplicationConfiguration 接口的实现[WSC-6.2-2]。

如果 WAR 文件不包含
ServerApplicationConfiguration 实现，
它必须部署它扫描 定位的所有端点[WSC-6.2-3]。

如果扫描的 WAR 文件定位到一个或多个
ServerApplicationConfiguration 实现，websocket
实现必须实例化每一个它发现的
ServerApplicationConfiguration 类，它
必须把扫描的包含它的（顶层 EAR 或包含的
JAR）归档的结果传递到它的方法[WSC-6.2-4]。

该集合是通过调用 web socket 实现必须部署的
ServerApplicationConfiguration 类 集合 的
getEndpointConfigurationClasses() 和
getAnnotatedEndpointClasses()获得
的所有结果的合并[WSC-6.2-5]。

注意： 这意味着开发人员容易地部署 WAR
文件中的所有端点， 通过简单地把用

于它们 的类文件捆绑 到 WAR 包 中 。这也意味着通过提供一个 iehuo 多个 ServerApplicationConfiguration 实现类，开发人员可以精确地控制哪些 WAR 文件中的端点是被部署。 通过从扫描中 （参考 Servlet3.0， 8.2.1 节） 排除某些 JAR 文 件，这也使得开发人员可以限制一个潜在地漫长地扫描过程。最后一种情况可能

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

是可取的，一个 WAR 文件包含很多 JAR 文件，开发人员知道其不包含任何 websocket 端点。

6.3.在独立的 websocket 服务器容器中的应用部署

该规范推荐独立的 websocket 服务器容器（即那些不包括一个 servlet 容器的）定位任何 websocket 服务器端点和应用包中的 ServerApplicationConfiguration 类并部署配置类返回的所有服务器端点集合。不过，如果他们希望，独立的 websocket 服务器容器可以使用其他的实现技术来部署端点。

6.4.Websocket 服务器路径

WebSocket 实现包括服务器功能必须为 websocket 定义一个根或 URI 空间。所谓 的 websocket 根，就是所有相对 websocket 路径的 URI 都是相对的。如果 websocket 服务器不包括 Servlet API，websocket 服务器可以选择 websocket 根本身。如果 websocket 服务器包含 Java Servlet API，websocket 根必须与 web 应用的 servlet 上下文根是相同的[WSC-6.4-1]。

6.5.平台版本

Java 平台最低版本是：

• Java SE 版本 7，用于 Java WebSocket 客户端 API [WSC-35]。• Java EE 版本 6，用于 Java WebSocket 服务器 API [WSC-36]。

http://jinnianshilongnian.iteye.com/

http://jinnianshilongnian.iteye.com/

7.Java EE 环境

7.1.Java EE 环境

当 Java EE 平台支持时，支持 Web Socket 应用还有一些额外的要求。**7.1.1. 使用 Java EE 组件创建端点**

除了能应用 WebSocket 注解到 Java 类，

websocket 实现还必须支持使用以下的 JavaEE 组件创建 websocket 端点[WSC-7.1.1-1]：

- 有状态会话 EJB
- 单例 EJB
- CDI 托管 bean

(待定：CDI 托管 Bean 允许的范围)

7.2.Http Session 与通过身份认证的状态的关系

这通常是有用的，对于开发人员嵌入 web socket 服务器端点到一个更大的 web 应用，能够在每个客户端基础上在 web 资源（如 JSP、JSF、Servlet）和服务客 户端的 web socket 端点之间共享信息。由于 web socket 连接是随着 http 请求初始 化的，所以在客户端正运行的 HttpSession 和在该 HttpSession 内建立的任何

web socket 之间存在一个关联。该 API
允许在打开阶段握手期间访问关联到相同客户
端的唯一 HttpSession [WSC-7.2-1]。
类似地，
如果打开阶段握手请求已经通过了服务器的身份验证，
则打开阶段握手
API 允许开发人员查询请求的用户
Principal。如果已经建立与请求客户端的连接，
web socket
实现把出现在打开阶段握手时的用户 Principal
看作为与 websocket Session 关联的用户
Principal[WSC-7.2-2]。
在一个 websocket 端点是 web
应用中的受保护的资源的情况下（参考第 8
章），就是说，需要一个授权用户访问它，因此，

websocket 实现必须确保在底层实现
已经决定验证身份不再有效之后 websocket
端点不再保持与它的节点连接[WSC-7.2-3]。
这可能发生，例如，如果用户退出了包含的
web 应用，或如果身份验证超时，或由
于其他一些原因无效的。在这种情况下，
websocket 实现必须立即使用

http://jinnianshilongnian.iteye.com/

websocket 关闭状态码 1008
关闭连接[WSC-7.2-3]。
另一方面，如果 websocket 端点不是 web 应用
的一个受保护资源，那么在打开阶 段握手建立

的连接过程中用户身份可能失效或在 websocket 运行期间没有 websocket 实现需要关闭连接情况下得到改变。

8.服务器安全

Web Socket 端点使用 web 容器安全模型来保护。这样做的目的是为了便于 web socket 开发人员声明访问一个 web socket 服务器端的是否需要经过身份认证，和谁可以访问它，和是否需要加密连接或不需要。映射到一个给定的 ws:// URI（如在第 3 和第 4 章中描述的）的 web socket

是受在部署描述符中列出的具有相同主 机名、端口和路径的 http:// URI 保护的，
因为这是它的打开阶段握手的 URI。

因此，web socket
开发人员可以指定一个身份认证方案，授予访问的用户角色和 到它们的 web socket
端点的传输保证。

8.1. web socket 身份认证

该规范没有定义 web
socket 本身也可以进行身份认证的机制。相反，
通过以 servlet 定义的安全机制为基础，
需要身份认证的 web socket
在进行身份认证以前必须依
赖于试图初始化一个连接的打开阶段握手请求。
通常 ， 这将 由包含 web socket 的 web 应用 中的一个 Http 身份验证
（可能是基本的或基于表单的） 在 web socket

的打开阶段握手之前执行。

如果一个客户端向一个受安全机制保护的 web socket 发送了一个未经身份认证的打开阶段握手请求，web socket 实现必须返回 401（未授权）响应到打开阶段我去请求并不会初始化 web socket 连接[WSC-8.1-1]。

待定 添加/重用 Servlet 安全注解定义 web socket 授权。

8.2.web socket 授权

web socket 授权可以通过往它打包在的 web 应用的 web.xml 添加一个<security-constraint>元素。安全约束中使用的 url-pattern 必须由容器使用来匹配 web socket 的打开阶段握手的请求 URI[WSC-82]。

实现必须解释除了 GET（或默认，缺少的）之外的任何 http-method 不应用到 web socket[WSC-8.2-1]。

8.3.传输保证

NONE
传输保证必须由容器解释为到允许未加密的 ws:// 到 web socket 的连接[WSC-8.3-1]。CONFIDENTIAL 传输保证必须由实现解释为仅允许在加密（wss://）的连接之上访问 web socket[WSC-8.3-1]。这可能需要一个预先身份认证请求。

ToDo 添加示例

9.参考书目

[1]互联网工程任务组（IETF）上的 WebSocket 协议 RFC 见 http://tools.ietf.org/html/rfc6455

[2] 万维网联盟（W3C）上的 WebSocket API 规范 见 http://dev.w3.org/html5/websockets/

[3] 专家组邮箱归档 http://java.net/projects/websocket-spec/lists/jsr356-experts/archive

[4] Servlet 规范 3.1 http://jcp.org/en/jsr/detail?id=340

[5]关键字用于使用在 RFCs 指出要求等级 http://www.ietf.org/rfc/rfc2119.txt

[6] URI-模板 http://tools.ietf.org/html/rfc6570