Java WebSocket 基础 消息通信

鹿克阿特米 2019-08-07 20:36:41 526 收藏 1

展开

为了发送WebSocket消息，无论注解式端点还是编程式端点，其API都是相同的：RemoteEndpoint和它的子类（RemoteEndpoint.Basic和RemoteEndpoint.Async）提供了发送消息的所有方法。

Ping和Pong消息通常被开发人员用来检查WebSocket底层连接的健康性。

从Session对象获得的RemoteEndpoint实例提供方法：

public void sendPing(ByteBuffer applicationData) throws IOException, IllegalArgumentException;

public void sendPong(ByteBuffer applicationData) throws IOException, IllegalArgumentException;

1

2

3

WebSocket规范定义了Ping和Pong消息可以传输125个字节大小的二进制数据

调用这些方法时，如果传入的ByteBuffer参数超过了125个字节，则会抛出IllegalArgumentException异常

RemoteEndpoint接口和它的派生类中的所有发送消息的方法都会通过抛出IOException异常来表明在消息的传输过程中出现了错误

1. 发送消息

同步发送 RemoteEndpoint.Basic接口

1.1 发送字符串消息

//发送文本消息

public void sendText(String text) throws IOException;

//使用Writter API发送String消息

public Writter getSendStream() throws IOException;

//以分片的方式发送文本消息

public void sendText(String partialMessage, boolean isLast) throws IOException;

1

2

3

4

5

6

7

8

1.2 发送二进制消息

//发送二进制消息

public void sendBinary(ByteBuffer data) throws IOException;

//分片发送二进制消息

public void sendBinary(ByteBuffer partialByte, boolean isLast) throws IOException;

//使用流发送二进制消息

public Outputstream getSendStream() throws IOException;

1

2

3

4

5

6

7

8

1.3 发送对象消息

//发送Java对象消息

public void sendObject(Object data) throws IOException, EncodeException;

1

2

Websocket是如何实现把对象编程WebSocket消息的，答案取决于你传什么类型的对象

如果你传一个Java基本类型对象（或者其等值装箱类），则WebSocket实现会把数据转换成一个标准的Java字符串对象（使用toString()方法）

如果传入的是其他对象，那么要为WebSocket实现提供一个javax.websocket.Encoder接口的实现

public String encode(T object) throws EncodeException;

1

public class DrawingEncoder implements Encoder.Text<DrawingObject> {

@Override

public String encode(DrawingObject drawingObject) throws EncodeException {

...

}

@Override

public void init(EndpointConfig endpointConfig) {}

@Override

public void destroy() {}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

javax.websocket.Encoder.Text<T>是最通用的接口，T就是想发送的对象类型（DrawingObject）

每次使用sendObject方法发送T类型的对象时，WebSocket实现都会调用相应的编码器，发给远程端点的实际是encode()方法返回的字符串

如果你的编码器无法把指定对象转换成字符串，很可能会抛出EncodeException异常，在这种情形下，EncodeException将会传播给sendObject方法

还可以灵活选择其他的Encoder接口

Encoder.TextStream<T> T转换成Writer

Encoder.Binary<T> T转换成ByteBuffer 对象编码成二进制消息

Encoder.CharacterStream<T> 或 Encoder.BinaryStream<T> T转换成OutputStream 把对象编码成Java I/O流

创建好编码器后，要选择如何在端点上配置它

注解式：

@ServerPoint(value = "/fruit", encoders = {MyAppleEncoder.class})

public class ... {}

1

2

编程式：

//为编程式客户端端点配置编码器

List<Class<? extends Encoder>> encoders = new ArrayList<>();

encoders.add(MyAppleEncoder.class);

ClientEndpointConfig config = ClientEndpointConfig.Builder.create().encoders(encoders).build();

1

2

3

4

1

2. 接收消息

2.1 注解式端点接收消息

//接收文本消息

@OnMessage

public void handleTextMessage(String textMessage) {

...

}

//接收二进制消息

@OnMessage

public void handleBinaryMessage(byte[] messageData) {

...

}

@OnMessage

public void handleBinaryMessage(ByteBuffer messageData) {

...

}

//接收Pong消息，需要声明一个类型为 javax.websocket.PongMessage 的方法参数

@OnMessage

public String handlePongMessage(PongMessage pongMessage) {

return "I got " + pongMessage.getApplicationData().length + " bytes of data.";

}

//处理文本消息流

@OnMessage

public void handle(Reader message) {}

//

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

//处理到达的文本消息片段

@OnMessage

public void handlePartial(String textMessagePart, boolean isLast) {

}

//处理到达的二进制消息片段

@OnMessage

public void handlePartial(byte[] data, boolean isLast) {}

@OnMessage

public void handlePartial(ByteBuffer data, boolean isLast) {}

1

2

3

4

5

6

7

8

9

10

把接收到的消息转换成Java对象，需要解码类

提供Decoder接口的实现

public class DrawingDecoder implements Decoder.Text<DrawingObject> {

@Override

public DrawingObject decode(String s) throws DecodeException {

...

}

@Override

public boolean willDecode(String s) {

...

}

@Override

public void init(EndpointConfig endpointConfig) {}

@Override

public void destroy() {}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

Decoder.Text<T> 把入站的文本消息转换成T类型的对象

willDecode 判断消息格式是否要用此解码

还有其他类型的解码器

Decoder.TextStream<T> Reader转换成T

Decoder.Binary ByteBuffer转换成T

Decoder.BinaryStream InputStream转换成T

Java基本类型和它的等价类可以自动转换（Integer、Duble …）

如果想解码的原始数据无法转换成期望的Java对象，将会抛出DecodeException异常。在所有这些解码失败中，引起失败的入站消息不会被传递，但解码器中产生的DecodeException异常会被传递给端点的错误处理方法。

@OnMessage注解的方法的返回值类型决定了寄回消息给发送者的消息类型

文本消息：String

二进制消息：byte[] 或 ByteBuffer

通过之前的转换规则，还可以响应标准Java基本类型及其等价类的文本信息（Integer、boolean …）

Java WebSocket在接收消息上有一个严格的限制： 每个注解式端点最多只有一个消息处理方法处理每种本地WebSocket消息类型（即文本消息、二进制消息 和 Pong消息）

例如：

@ServerEndpoint(value = "/orchard", decoders = "{OrangeDecoder.class}")

public class FruitTree {

@OnMessage

public void handleString(String message) {

...

}

@OnMessage

public void handleOrange(Orange orange) {

...

}

}

1

2

3

4

5

6

7

8

9

10

11

12

假设OrangeDecoder 解码器实现了Decoder.Text<Orange> 或 Decoder.TextStream<T>中的任意一个，那么这个端点中就会有两个方法能够处理入站的文本消息，会部署失败，需要分到两个不同端点中。

2.2 编程式端点接收消息

编程式端点使用什么方式接收消息取决于MessageHandler接口和它的子类性

public class ProgrammaticEchoServer extends Endpoint {

@Override

public void onOpen(Session session, EndpointConfig endpointConfig) {

final Session mySession = session;

mySession.addMessageHandler(new MessageHandler.Whole<String>() {

@Override

public void onMessage(String incomingMessage) {

try {

mySession.getBasicRemote().sendText("I got this (" + incomingMessage + ") so I am sending it back.");

} catch (IOException e) {

System.out.println("something went wrong:" + e.getMessage());

}

}

});

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

MessageHandler.Whole<String> 文本消息 String

MessageHandler.Whole<Reader> 文本消息 Java I/O流

MessageHandler.Whole<ByteBuffer> 二进制 Java NIO ByteBuffer

MessageHandler.Whole<byte[]> 二进制 Byte数组

MessageHandler.Whole<InputStream> 二进制 Java I/O流

MessageHandler.Partial<String> 文本片段 Java字符串序列

MessageHandler.Partial<ByteBuffer> 二进制片段 ByteBuffer序列

MessageHandler.Partial<byte[]> 二进制片段 字节数组序列

MessageHandler.Whole<T> 和 Decoder.Text<T> 或者 Decoder.TextStream<T> 文本消息 对象类型T

MessageHandler.Whole<T> 和 Decoder.Binary<T> 或者 Decoder.BinaryStream<T> 二进制消息 对象类型T

//在编程式服务器端点中配置Decoder

public class MyServerApplicationConfig implements ServerApplicationConfig {

@Override

public Set<ServerEndpointConfig> getEndpointConfigs(Set<Class<? extends Endpoint>> set) {

Set<ServerEndpointConfig> configSet = new HashSet<>();

List<Class<? extends Endpoint>> decoders = new ArrayList<>();

decoders.add(MyOrangeDecoder.class);

ServerEndpointConfig serverEndpointConfig = ServerEndpointConfig.Builder.create(MyEndpoint

.class, "/fruit").decoders(decoders).build();

configSet.add(serverEndpointConfig);

return configSet;

}

@Override

public Set<Class<?>> getAnnotatedEndpointClasses(Set<Class<?>> set) {

return set;

}

}