**Jocket – Polling 工作步骤**

1. 确定服务路径

路径由服务的提供方，通过Java注解指定，例如下图，路径为“/chat/simple”：



1. 创建Jocket Session

发起HTTP GET请求，生成Jocket的Session ID。Jocket的Session和HTTP的Session类似，但不是同一内容。URL支持两种格式：

http://<host>[:port]/<app-name><path>.jocket

http://<host>[:port]/<app-name>/create.jocket?jocket\_path=<path>

其中：

* app-name：WebApp的名称，在示例包中，app-name为“jocket”
* path：服务路径，对应上文中的“/chat/simple”。

因此，假定服务器IP为192.168.1.2，端口为8080，对于示例包，URL为可以使用：

http://192.168.1.2:8080/jocket/chat/simple.jocket

http://192.168.1.2:8080/jocket/create.jocket?jocket\_path=/chat/simple

如果调用成功，返回以下内容：

{

"sessionId":"zRU8XfDqNlSjPp2sDatL", //Jocket Session ID

"upgradable":true, //是否可将传输方式升级为WebSocket

"pingInterval":25000, //client端每隔多少毫秒发起一次ping（心跳）

"pingTimeout":20000 //ping超时时间（毫秒）。超过该时间，则关闭连接

}

1. 握手与心跳
   1. URL及数据格式说明

步骤2创建Session后，不能立即收发消息，需要先进行一次握手。

握手及后续所有其它操作的URL格式为：

http://<host>[:port]/<app-name>/jocket?s=<sessionId>

在上文的场景中，URL为：

http://192.168.1.2:8080/jocket/chat/jocket?s=zRU8XfDqNlSjPp2sDatL

上行采用POST方法，POST的内容为JSON格式，在本文中称为一个“包”（packet）。下行采用GET方法，GET返回的内容格式与POST上行的内容相同。

一个包可以包含以下属性：

* type：包的类型，目前支持open/close/ping/pong/upgrade/message
* name：如果类型为message，则可以通过name指定消息名（消息类型）。也可以不使用name，消息类型封装在data中
* data：包的具体数据，由type决定。对于open/ping/pong/upgrade，不需要提供本属性。data必须为字符串，如果需要传输JSON数据，需要将JSON转为字符串（例如通过JavaScript中的JSON.stringify函数）。
  1. Polling实现方式

客户端发起GET请求，准备接收数据。服务端如果有数据则可以立即完成该请求，否则将hold住该请求，使其不立即返回，直到有下行数据或超时。

客户端在接到GET返回的内容后，应立即再次发起一次新的GET请求，准备接收下一个包。同时解析GET返回的内容，如有必要，通知相关的listener。例如当收到type=message的消息时，应通知使用方。此步骤建议另开线程异步处理。

* 1. 握手

在获取到Jocket Session ID后，需要：

1. 发起GET请求，准备从服务端收数据（开始Polling）
2. 发起POST请求，内容为：

{"type":"ping"}

如果在指定时间（pingTimeout）内，通过Polling收到以下包

{"type":"pong"}

则视为握手成功，此时连接正式开启，应触发open事件。

* 1. 心跳

握手成功后，客户端应每隔一定时间（pingInterval），发起一次心跳请求。心跳的格式与握手完全相同，只是第一次称为“握手”，后续称为“心跳”。

1. 消息收发

如果包的type为message，则表示为普通消息。消息可以包含一个消息名（name）和消息数据（data）。用户也可以不使用消息名，所有内容均封装在消息数据中。程曦的APP属于该情况。

data的内容格式由用户决定，但必须为字符串格式。Jocket库不需要对data的内容进行解析，解析动作由用户处理。例如以下都是合法的消息：

{

"type":"message",

"data":"Nice to meet you!"

}

{

"type":"message",

"data":"{\"type\":\"Text\",\"content\":\"Nice to meet you!\"}"

}

1. 连接关闭

在以下情况下，连接会关闭：

* 服务端主动关闭连接：客户端会收到下行的type=close的包
* 客户端主动关闭连接：客户端需要向服务端发送type=close的包
* 因心跳超时，客户端关闭连接：客户端需要向服务端发送type=close的包
* 因其它因素（例如网络异常导致polling失败）关闭连接：客户端需要向服务端发送type=close的包

对于后两种情况，虽然type=close的包不一定能够发送到服务端，但仍然需要尝试发送。

在连接关闭时，客户端应释放相关资源，并触发close事件。

1. 重新连接

如果因网络异常等原因，导致连接关闭，客户端可以在library级别自动重连。