1. 概述

WebSocket是HTML5中强大的通信功能，它定义了一个全双工通信信道。扩展了浏览器与服务端的通信功能（**实时通信**），使服务端也能主动向客户端发送数据。

在WebSocket之前，传统的实现方法有：1）Ajax轮询（polling）；2）Comet技术：长轮询（long-polling）、流（streaming）。

轮询是最原始的实现实时Web应用的解决方案。轮询技术要求客户端以设定的时间间隔周期性地向服务端发送请求，频繁地查询是否有新的数据改动。明显地，这种方法会导致过多不必要的请求，浪费流量和服务器资源。

　　Comet技术又可以分为长轮询和流技术。长轮询改进了上述的轮询技术，减小了无用的请求。它会为某些数据设定过期时间，当数据过期后才会向服务端发送请求；这种机制适合数据的改动不是特别频繁的情况。流技术通常是指客户端使用一个隐藏的窗口与服务端建立一个HTTP长连接，服务端会不断更新连接状态以保持HTTP长连接存活；这样的话，服务端就可以通过这条长连接主动将数据发送给客户端；流技术在大并发环境下，可能会考验到服务端的性能。

　　这两种技术都是基于请求-应答模式，都不算是真正意义上的实时技术；它们的每一次请求、应答，都浪费了一定流量在相同的头部信息上，并且开发复杂度也较大。

JavaEE 7中出了JSR-356:Java API for WebSocket规范。不少Web容器，如Tomcat,Nginx,Jetty等都支持WebSocket。Tomcat从7.0.27开始支持 WebSocket，从7.0.47开始支持JSR-356。

1. WebSocket的执行过程
2. 客户端向服务器发送数据
3. 服务器获取客户端的数据
4. 服务器向客户端发送数据
5. 客户端向服务器端获取数据

三、操作步骤

第一：创建WebSocket服务器的端点程序(Endpoint) 🡪 @ServerEndpoint("/echo")

1．服务器获取客户端的数据

2．服务器向客户端发送数据

3．三个事件

//定义了一个Endpoint

@ServerEndpoint("/echo")

public class EchoSocket {

public EchoSocket() {

System.out.println("echosocket....");

}

@OnOpen

public void open(Session session) {

//每个管道都是独立的Endpoint对象

System.out.println("session:"+session.getId());

}

@OnClose

public void close(Session session) {

System.out.println("sessionID:"+session.getId()+"已经关闭了...");

}

@OnMessage

public void echoMessage(Session session, String msg, boolean last) {

//服务器获取客户端的数据：msg

System.out.println("client:"+msg);

try {

//服务器向客户端发送数据

session.getBasicRemote().sendText("server:"+msg);

} catch (IOException e) {

e.printStackTrace();

}

}

}

注意：

1）服务器的端点程序有三个事件，分别使用三个注解来定义

@OnOpen:打开连接时，触发的响应事件

@OnMessage:发送数据时，触发的响应事件

@onClose:关闭连接时，触发的响应事件

2）每个管道都是独立的Endpoint对象

3）session指的是当前客户端与服务器端连接通信管道的会话(有多少个连接，就有多少个session）。

第二：定义WebSocket的核心配置类

public class DemoConfig implements ServerApplicationConfig {

/\*\*

\* 1.注解方式的启动

\* getAnnotatedEndpointClasses:扫描src下面所有的类，并把具有@ServerEndPoint注解的类添加到set集合中

\* 具有@ServerEndPoint注解的类就是一个Endpoint,它是一个Websocke的一个服务器程序

\*/

@Override

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

System.out.println("ServerApplicationConfig启动了....");

System.out.println("Endpoint的数量："+set.size());

//在启动服务器时，会扫描所有的Endpoint，并添加到set集合中，并返回;

//同时，在返回之前可以对set集合进行处理（过滤）

return set;

}

//2.接口方式的启动

public Set<ServerEndpointConfig> getEndpointConfigs(Set<Class<? extends Endpoint>> arg0) {

return null;

}

}

注意：

1．实现ServerApplicationConfig接口的类，会在项目启动时自动执行（与ContextListener相似），在启动服务器时，会扫描所有的Endpoint，并添加到set集合中，并返回。

2．启动方式有两种方法，分别为：1.注解方式的启动；2.接口方式的启动，在定义时，使用其中一种方法即可。

第三：编写客户端程序（js）

1. 创建WebSocket对象，并指定服务器端程序
2. 客户端向服务器发送数据
3. 客户端向服务器端获取数据

var ws = null;

function connect() {

//定义服务器端程序连接地址

var target = "ws://localhost:8080/websocket/echo";

//浏览器兼容判断

if ('WebSocket' in window) {

ws = new WebSocket(target);

} else if ('MozWebSocket' in window) {

ws = new MozWebSocket(target);

} else {

alert('WebSocket is not supported by this browser.');

return;

}

//客户端接收服务器端的数据

ws.onmessage = function (event) {

//console.info(event) ;

document.getElementById("result").innerHTML += event.data +"<br/>";

//alert(event.data) ;

};

}

// 客户端向服务器发送数据🡪ws.send(消息) ;

function sendMessage() {

var msgTxt = document.getElementById("msg") ;

if(ws!=null) {

ws.send(msgTxt.value) ;

msgTxt.value = "" ;

msgTxt.focus() ;

} else {

alert("WebSocket未连接，请连接") ;

}

}