Scala.js networking made easy

Olivier Blanvillain
EPFL, Switzerland
olivier.blanvillain@gmail.com

January 16, 2015

Abstract

abstract

1 Intro

Js/NodeJs Many Apis Comet/Websocket/Webrtc

2 Transport

Scope: Unified interfaces. No magic.

The interface:

```
trait Transport {
  type Address
  def listen(): Future[Promise[ConnectionListener]]
  def connect(remote: Address): Future[ConnectionHandle]
  def shutdown(): Unit
}

trait ConnectionHandle {
  def handlerPromise: Promise[MessageListener]
  def closedFuture: Future[Unit]
  def write(outboundPayload: String): Unit
  def close(): Unit
}
```

All implementations

- js
  - WebSocket client

    - SockJS client
    - WebRtc client
- netty
    - WebSocket server
    - SockJS server (in next netty release)
- tyrus
    - WebSocket client
- play
    - WebSocket client
    - SockJS client (with a plugin)

Wrappers - Akka - Autowire (RPC)sss

Two browser tests

3 Survivor game

Goal: Cross platform JS/JVM realtime mutiplayer game Everything but UI shared Clock synked, same game simulated on both platforms Pure functional design (taking advantage of scala collections immutability) "Lag compensation" React UI (& hack for the JVM version) Results: 60FPS on both platforms, lag free gameplay

4 Conclusion

A much longer example was written by Gil [1]. Now go read section 1!

References

[1] J. Y. Gil. $\LaTeX 2_\varepsilon$ for graduate students. manuscript, Haifa, Israel, 2002.