Open-Source Report: TCP Connection

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/playframework/playframework
License Type	The Play Framework source code is released under the **Apache 2.0** license.
License Description	• Commercial use • Modification • Distribution • Patent use • Private use
License Restrictions	• Trademark use • Liability • Warranty

https://github.com/playframework/playframework/tree/main/core/play/src/main/java/play/mvc

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.
Play comes with two configurable server backends called Akka-HTTP and Netty server, which handle the low-level work of processing HTTP requests and responses from TCP/IP packets once the TCP socket is created. In this project, we will use Akka - HTTP server backend to meet the requirement. After the TCP socket is created, Play will parse all the headers and body of the request, so it will know if the request is in proper format or not. Then Play will try to grab the response from the client, for example, users can return ok, redirect, and error etc. to the server with their payload, so the server can build a successed connection.

Where is the specific code that does what you use the tech for? You *must* provide a link to the specific file in the repository for your tech with a line number or number range
We use the https://github.com/playframework/playframework/blob/main/core/play-guice/src/main/java/play/inject/guice/Guiceable.java(Guiceable.java, line1-28), to allow us to import external libraries to the project. And from https://github.com/playframework/playframework/blob/main/core/play-guice/src/main/java/play/inject/guice/GuiceApplicationBuilder.java(GuiceApplicationBuilder.java, line 96-128) will check if the external libraries is available and build the new project depends on the libraries. Also, we import the HttpServer in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Http.java(Http.java, line 1-2404), this is http builder for play framework, and the whole class are used to get the request. As you can see from line 53 to line 399, these lines of code parsed the request header

and they can get the header by calling the getHeaders() function, and from line 612 to 963 are the functions for getting the body of the request. After got the body, Play tries to extract the data just received either by bytes or file. Then Play will send the response through Result in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Result.java, in line 41 to 46, we will see all the response variables Play sets, header, body, flash, session, cookies, and attributes. So Play will see the response the developer gave to it and send different responses depending on the different response status. If the developer sends ok with the response body, Play will first get the status, and in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Results.java(line 469), its a ok response with content as its argument, in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/StatusHeader.java(lines 93 to 106) will send the file or data we sent, so the TCP connection is successfully built.

```java
/*
 * Copyright (C) from 2022 The Play Framework Contributors <https://github.com/playframework>, 2011-2021 Lightbend Inc. <https://www.lightbend.com>
 */

package play.inject.guice;

import play.api.inject.guice.GuiceableModule;
import play.api.inject.guice.GuiceableModule$;
import play.libs.Scala;

public class Guiceable {

  public static GuiceableModule modules(com.google.inject.Module... modules) {
    return GuiceableModule$.MODULE$.fromGuiceModules(Scala.toSeq(modules));
  }

  public static GuiceableModule modules(play.api.inject.Module... modules) {
    return GuiceableModule$.MODULE$.fromPlayModules(Scala.toSeq(modules));
  }

  public static GuiceableModule bindings(play.api.inject.Binding... bindings) {
    return GuiceableModule$.MODULE$.fromPlayBindings(Scala.toSeq(bindings));
  }

  public static GuiceableModule module(Object module) {
    return GuiceableModule$.MODULE$.guiceable(module);
  }
}
```

Guiceable.java

```
     * Override the module loader with the given Play modules.
     *
     * @param modules the set of overriding modules
     * @return an application builder that incorporates the overrides
     */
    public GuiceApplicationBuilder load(play.api.inject.Module... modules) {
      return load(Guiceable.modules(modules));
    }

    /**
     * Override the module loader with the given Play bindings.
     *
     * @param bindings the set of binding override
     * @return an application builder that incorporates the overrides
     */
    public GuiceApplicationBuilder load(play.api.inject.Binding<?>... bindings) {
      return load(Guiceable.bindings(bindings));
    }

    /**
     * Create a new Play Application using this configured builder.
     *
     * @return the application
     */
    public Application build() {
      return injector().instanceOf(Application.class);
    }

    /**
     * Implementation of Self creation for GuiceBuilder.
     *
     * @return the application builder
     */
    protected GuiceApplicationBuilder newBuilder(
        play.api.inject.guice.GuiceApplicationBuilder builder) {
      return new GuiceApplicationBuilder(builder);
    }
}
```

GuiceApplicationBuilder.java

```
49
50  /** Defines HTTP standard objects. */
51  public class Http {
52
53    public static class Headers {
54
55      private final Map<String, List<String>> headers;
56
57      public Headers(Map<String, List<String>> headers) {
58        this.headers = new TreeMap<>(String.CASE_INSENSITIVE_ORDER);
59        this.headers.putAll(headers);
60      }
61
62      /**
63       * @return all the headers as a map.
64       * @deprecated Deprecated as of 2.8.0. Use {@link #asMap()} instead.
65       */
66      @Deprecated
67      public Map<String, List<String>> toMap() {
68        return headers;
69      }
70
71      /** @return all the headers as an unmodifiable map. */
72      public Map<String, List<String>> asMap() {
73        return Collections.unmodifiableMap(headers);
74      }
75
76      /**
77       * Checks if the given header is present.
78       *
79       * @param headerName The name of the header (case-insensitive)
80       * @return <code>true</code> if the request did contain the header.
81       */
82      public boolean contains(String headerName) {
83        return headers.containsKey(headerName);
84      }
85
86      /**
```

Part of the example in Http.java

```java
/** Any action result. */
public class Result {

  /** Statically compiled pattern for extracting the charset from a Result. */
  private static final Pattern SPLIT_CHARSET = Pattern.compile("(?i);\\s*charset=");

  private final ResponseHeader header;
  private final HttpEntity body;
  private final Flash flash;
  private final Session session;
  private final List<Cookie> cookies;
  private final TypedMap attrs;

  /**
   * Create a result from a Scala ResponseHeader and a body.
   *
   * @param header the response header
   * @param body the response body.
   * @param session the session set on the response.
   * @param flash the flash object on the response.
   * @param cookies the cookies set on the response.
   * @param attrs the typed attributes set on the response.
   */
  public Result(
```

Result.java

Open-Source Report: Parsing HTTP headers

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/playframework/playframework
License Type	The Play Framework source code is released under the **Apache 2.0** license.
License Description	• Commercial use • Modification • Distribution • Patent use • Private use
License Restrictions	• Trademark use • Liability • Warranty

https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.
For Parsing HTTP headers part, Play Framework provides HTTP routing, Result, Action, HTTP Parsers, in order to Parsing the HTTP headers. The router is responsible for converting each received HTTP request into an Action. A simple result contains a status code, a set of HTTP headers and a body to be sent to the client and we can add and update the headers. Action is a play function that handles a request and generate what result we want to send to the client. As we know, HTTP request includes Put and Post request and they contains a body, the body can use any format and contain HTTP headers, In Play, HTTP Parsers can transforms the request body into Scala value in order to let us processing of information better. This is how Parsing the HTTP headers works.

Where is the specific code that does what you use the tech for? You *must* provide a link to the specific file in the repository for your tech with a line number or number range.
For Parsing HTTP headers part, We use https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/routing/Router.java (Router.java, line 1-81) to build the router for our project and this allows us to translating each incoming HTTP request to an action, we can see that each route is contains an HTTP method and patterns which associated with the action generator. The action can handles a request and generate what result we want to send to the client. In our project, we using

controller, and the controller is a singleton object that generates Action values. It's in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Controller.java (Controller.java, line 1-16) the result in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Result.java (Result.java, line 116-118) we can see that it contains status, headers, and body, so we can set and send our HTTP request. For the parsing we use HTTP accept encoding scala to parse the HTTP header to scala value, it's in https://github.com/playframework/playframework/blob/main/core/play/src/main/scala/play/api/http/AcceptEncoding.scala (http/AcceptEncoding.scala, line 161-242), this in order to let us Parse the header and return a list of preferred encodings, and then we need to change the data into java that we use so we need to use the function in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/Http.java (Http.java, line 50-165, 751-770) in order to let us processing of data better, so the request went throught the router to controller and then parsing the headers, get the data we want and handle it, this is how the parsing HTTP parts works in our project.

```java
public interface Router {

  List<RouteDocumentation> documentation();

  Optional<Handler> route(RequestHeader request);

  Router withPrefix(String prefix);

  default Router orElse(Router router) {
    return this.asScala().orElse(router.asScala()).asJava();
  }

  default play.api.routing.Router asScala() {
    return SimpleRouter$.MODULE$.apply(
        new JavaPartialFunction<play.api.mvc.RequestHeader, Handler>() {
          @Override
          public Handler apply(play.api.mvc.RequestHeader req, boolean isCheck) throws Exception {
            Optional<Handler> handler = route(req.asJava());
            if (handler.isPresent()) {
              return handler.get();
            } else if (isCheck) {
              return null;
            } else {
              throw noMatch();
            }
          }
        });
  }

  static Router empty() {
    return play.api.routing.Router$.MODULE$.empty().asJava();
  }

  /** Request attributes used by the router. */
  class Attrs {
    /** Key for the {@link HandlerDef} used to handle the request. */
    public static final TypedKey<HandlerDef> HANDLER_DEF =
        new TypedKey<>(play.api.routing.Router.Attrs$.MODULE$.HandlerDef());
  }

```

Router.java

```java
/*
 * Copyright (C) from 2022 The Play Framework Contributors <https://github.com/playframework>, 2011-2021 Lightbend Inc. <https://www.lightbend.com>
 */

package play.mvc;

import static play.mvc.Http.*;

/** Superclass for a Java-based controller. */
public abstract class Controller extends Results implements Status, HeaderNames {

  /** Generates a 501 NOT_IMPLEMENTED simple result. */
  public static Result TODO(Request request) {
    return status(NOT_IMPLEMENTED, views.html.defaultpages.todo.render(request.asScala()));
  }
}
```

Controller.java

```java
/** Defines HTTP standard objects. */
public class Http {

  public static class Headers {

    private final Map<String, List<String>> headers;

    public Headers(Map<String, List<String>> headers) {
      this.headers = new TreeMap<>(String.CASE_INSENSITIVE_ORDER);
      this.headers.putAll(headers);
    }

    /**
     * @return all the headers as a map.
     * @deprecated Deprecated as of 2.8.0. Use {@link #asMap()} instead.
     */
    @Deprecated
    public Map<String, List<String>> toMap() {
      return headers;
    }

    /** @return all the headers as an unmodifiable map. */
    public Map<String, List<String>> asMap() {
      return Collections.unmodifiableMap(headers);
    }

    /**
     * Checks if the given header is present.
     *
     * @param headerName The name of the header (case-insensitive)
     * @return <code>true</code> if the request did contain the header.
     */
    public boolean contains(String headerName) {
      return headers.containsKey(headerName);
    }

    /**
     * Gets the header value. If more than one value is associated with this header, then returns
     * the first one.
     *
     * @param name the header name
     * @return the first header value or empty if no value available.
```

Http.java

```
90        * @param name the header name
91        * @return the first header value or empty if no value available.
92        */
93       public Optional<String> get(String name) {
94         return Optional.ofNullable(headers.get(name))
95             .flatMap(headerValues -> headerValues.stream().findFirst());
96       }
97
98       /**
99        * Get all the values associated with the header name.
100       *
101       * @param name the header name.
102       * @return the list of values associates with the header of empty.
103       */
104      public List<String> getAll(String name) {
105        return headers.getOrDefault(name, Collections.emptyList());
106      }
107
108      /** @return the scala version of this headers. */
109      public play.api.mvc.Headers asScala() {
110        return new play.api.mvc.Headers(
111            JavaHelpers$.MODULE$.javaMapOfListToScalaSeqOfPairs(this.headers));
112      }
113
114      /**
115       * Add a new header with the given value.
116       *
117       * @param name the header name
118       * @param value the header value
119       * @return this with the new header added
120       * @deprecated Deprecated as of 2.8.0. Use {@link #adding(String, String)} instead.
121       */
122      @Deprecated
123      public Headers addHeader(String name, String value) {
124        this.headers.put(name, Collections.singletonList(value));
125        return this;
126      }
127
```

Http.java

```
128       /**
129        * Add a new header with the given value.
130        *
131        * @param name the header name
132        * @param value the header value
133        * @return a new Header instance with the new header added
134        */
135       public Headers adding(String name, String value) {
136         return adding(name, Collections.singletonList(value));
137       }
138
139       /**
140        * Add a new header with the given values.
141        *
142        * @param name the header name
143        * @param values the header values
144        * @return this with the new header added
145        * @deprecated Deprecated as of 2.8.0. Use {@link #adding(String, List)} instead.
146        */
147       @Deprecated
148       public Headers addHeader(String name, List<String> values) {
149         this.headers.put(name, values);
150         return this;
151       }
152
153       /**
154        * Add a new header with the given values.
155        *
156        * @param name the header name
157        * @param values the header values
158        * @return a new Header instance with the new header added
159        */
160       public Headers adding(String name, List<String> values) {
161         Map<String, List<String>> newHeaders = new HashMap<>(this.headers.size() + 1);
162         newHeaders.putAll(this.headers);
163         newHeaders.put(name, values);
164         return new Headers(newHeaders);
165       }
166
```

Http.java

Open-Source Report: WebSockets

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/playframework/playframework
License Type	The Play Framework source code is released under the **Apache 2.0** license.
License Description	• Commercial use • Modification • Distribution • Patent use • Private use
License Restrictions	• Trademark use • Liability • Warranty

https://github.com/playframework/playframework/tree/main/core/play/src/main/java/play/mvc

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.
For Websockets part, Play Framework provides Akka and play.websocket for us to build a Websockets. Play framework's websockets handing is built around Akka Stream, it modeled as flow and the incoming websocket message follow by the flow and send it to the client. For sending and receiving message, we can use Play utility, ActorFlow in Play framework to allow us to convert the data to the flow. In our project, to build a websocket, first, we need to build a controller and contains Materializer in it, The Materializer is a factory for stream execution engines which allow us to create a flow and handle the websocket message. We also need a routes to map the incoming request to the controller. Then we using Actor function to receive message and handle the data. This is the tech that we use in our project for websocket.

Where is the specific code that does what you use the tech for? You *must* provide a link to the specific file in the repository for your tech with a line number or number range.
For Websockets part, the specific code that we use in the controller and route is provided in the previous report. The new tech that we use in this part is the AkkaStreams and the websocket. The websocket is in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/mvc/WebSocket.java (WebSocket.java, line 30-200) the function in this file allow us to call the websocket function receiving message and parse data. The AkkaStream is in https://github.com/playframework/playframework/blob/main/core/play/src/main/java/play/libs/stre

ams/AkkaStreams.java (AkkaStreams.java, line36-124) the AkkaStream allow us to Create websocket interface, which allow us to getting reques, message, and create a flow and sending the message. so when we getting a websocket message, it went into a flow by AkksStreams and handle by the play.websocket function. this is how the websockets works in our project.

```
67    /** Acceptor for JSON WebSockets. */
68    public static final MappedWebSocketAcceptor<JsonNode, JsonNode> Json =
69        new MappedWebSocketAcceptor<>(
70            Scala.partialFunction(
71                message -> {
72                  try {
73                    if (message instanceof Message.Binary) {
74                      return F.Either.Left(
75                          play.libs.Json.parse(
76                              ((Message.Binary) message).data().iterator().asInputStream()));
77                    } else if (message instanceof Message.Text) {
78                      return F.Either.Left(play.libs.Json.parse(((Message.Text) message).data()));
79                    }
80                  } catch (RuntimeException e) {
81                    return F.Either.Right(
82                        new Message.Close(CloseCodes.Unacceptable(), "Unable to parse JSON message"));
83                  }
84                  throw Scala.noMatch();
85                }),
86            json -> new Message.Text(play.libs.Json.stringify(json)));
87
88    /**
89     * Acceptor for JSON WebSockets.
90     *
91     * @param in The class of the incoming messages, used to decode them from the JSON.
92     * @param <In> The websocket's input type (what it receives from clients)
93     * @param <Out> The websocket's output type (what it writes to clients)
94     * @return The WebSocket acceptor.
95     */
96    public static <In, Out> MappedWebSocketAcceptor<In, Out> json(Class<In> in) {
97      return new MappedWebSocketAcceptor<>(
98          Scala.partialFunction(
99              message -> {
100               try {
101                 if (message instanceof Message.Binary) {
102                   return F.Either.Left(
103                       play.libs.Json.mapper()
104                           .readValue(
105                               ((Message.Binary) message).data().iterator().asInputStream(), in));
106                 } else if (message instanceof Message.Text) {
107                   return F.Either.Left(
```

WebSocket.java

```java
  public static <In, Out> MappedWebSocketAcceptor<In, Out> json(Class<In> in) {
    return new MappedWebSocketAcceptor<>(
        Scala.partialFunction(
            message -> {
              try {
                if (message instanceof Message.Binary) {
                  return F.Either.Left(
                      play.libs.Json.mapper()
                          .readValue(
                              ((Message.Binary) message).data().iterator().asInputStream(), in));
                } else if (message instanceof Message.Text) {
                  return F.Either.Left(
                      play.libs.Json.mapper().readValue(((Message.Text) message).data(), in));
                }
              } catch (Exception e) {
                return F.Either.Right(new Message.Close(CloseCodes.Unacceptable(), e.getMessage()));
              }
              throw Scala.noMatch();
            }),
        outMessage -> {
          try {
            return new Message.Text(play.libs.Json.mapper().writeValueAsString(outMessage));
          } catch (Exception e) {
            throw new RuntimeException(e);
          }
        });
  }

  /**
   * Utility class for creating WebSockets.
   *
   * @param <In> the type the websocket reads from clients (e.g. String, JsonNode)
   * @param <Out> the type the websocket outputs back to remote clients (e.g. String, JsonNode)
   */
  public static class MappedWebSocketAcceptor<In, Out> {
    private final PartialFunction<Message, F.Either<In, Message>> inMapper;
    private final Function<Out, Message> outMapper;
```

WebSocket.java

```
36      public static <In, FlowIn, Out> Flow<In, Out, ?> bypassWith(
37          Function<In, F.Either<FlowIn, Out>> splitter, Flow<FlowIn, Out, ?> flow) {
38        return bypassWith(
39            Flow.<In>create().map(splitter::apply),
40            play.api.libs.streams.AkkaStreams.onlyFirstCanFinishMerge(2),
41            flow);
42      }
43
44      /**
45       * Using the given splitter flow, allow messages to bypass a flow.
46       *
47       * <p>If the splitter flow produces Left, they will be fed into the flow. If it produces Right,
48       * they will bypass the flow.
49       *
50       * @param <In> the In type parameter for Flow
51       * @param <FlowIn> the FlowIn type parameter for the left branch in Either.
52       * @param <Out> the Out type parameter for Flow.
53       * @param flow the original flow.
54       * @param splitter the splitter function.
55       * @param mergeStrategy the merge strategy (onlyFirstCanFinishMerge, ignoreAfterFinish,
56       *     ignoreAfterCancellation)
57       * @return the flow with a bypass.
58       */
59      public static <In, FlowIn, Out> Flow<In, Out, ?> bypassWith(
60          Flow<In, F.Either<FlowIn, Out>, ?> splitter,
61          Graph<UniformFanInShape<Out, Out>, ?> mergeStrategy,
62          Flow<FlowIn, Out, ?> flow) {
63        return splitter.via(
64            Flow.fromGraph(
65                GraphDSL.<FlowShape<F.Either<FlowIn, Out>, Out>>create(
66                    builder -> {
67
68                      // Eager cancel must be true so that if the flow cancels, that will be propagated
69                      // upstream.
70                      // However, that means the bypasser must block cancel, since when this flow
71                      // finishes, the merge
72                      // will result in a cancel flowing up through the bypasser, which could lead to
73                      // dropped messages.
74                      // Using scaladsl here because of https://github.com/akka/akka/issues/18384
75                      UniformFanOutShape<F.Either<FlowIn, Out>, F.Either<FlowIn, Out>> broadcast =
76                          builder.add(Broadcast.create(2, true));
77                      UniformFanInShape<Out, Out> merge = builder.add(mergeStrategy);
78
79                      Flow<F.Either<FlowIn, Out>, FlowIn, ?> collectIn =
80                          Flow.<F.Either<FlowIn, Out>>create()
81                              .collect(
82                                  Scala.partialFunction(
```

AkkaStreams.java

```
                    Flow.<F.Either<FlowIn, Out>>create()
                        .collect(
                            Scala.partialFunction(
                                x -> {
                                  if (x.left.isPresent()) {
                                    return x.left.get();
                                  } else {
                                    throw Scala.noMatch();
                                  }
                                }));

                Flow<F.Either<FlowIn, Out>, Out, ?> collectOut =
                    Flow.<F.Either<FlowIn, Out>>create()
                        .collect(
                            Scala.partialFunction(
                                x -> {
                                  if (x.right.isPresent()) {
                                    return x.right.get();
                                  } else {
                                    throw Scala.noMatch();
                                  }
                                }));

                Flow<F.Either<FlowIn, Out>, F.Either<FlowIn, Out>, ?> blockCancel =
                    play.api.libs.streams.AkkaStreams
                        .<F.Either<FlowIn, Out>>ignoreAfterCancellation()
                        .asJava();

                // Normal flow
                builder
                    .from(broadcast.out(0))
                    .via(builder.add(collectIn))
                    .via(builder.add(flow))
                    .toInlet(merge.in(0));

                // Bypass flow, need to ignore downstream finish
                builder
                    .from(broadcast.out(1))
                    .via(builder.add(blockCancel))
                    .via(builder.add(collectOut))
                    .toInlet(merge.in(1));

                return new FlowShape<>(broadcast.in(), merge.out());
            }));
    }
}
```

AkkaStreams.java