Web Technologies in Java EE

JAX-RS, JSON-P, WebSocket, AngularJS

Tomáš Remeš

Agenda

- **Client-Side** vs. **Server Side Web**
- **JAX-RS 2.0** – RESTFul Services

  - Origins + News in 2.0
- **JSON-P –** Java API for JSON Processing
- **Java API for WebSocket**
- **AngularJS –** client side JavaScript framework

Client-Side

vs

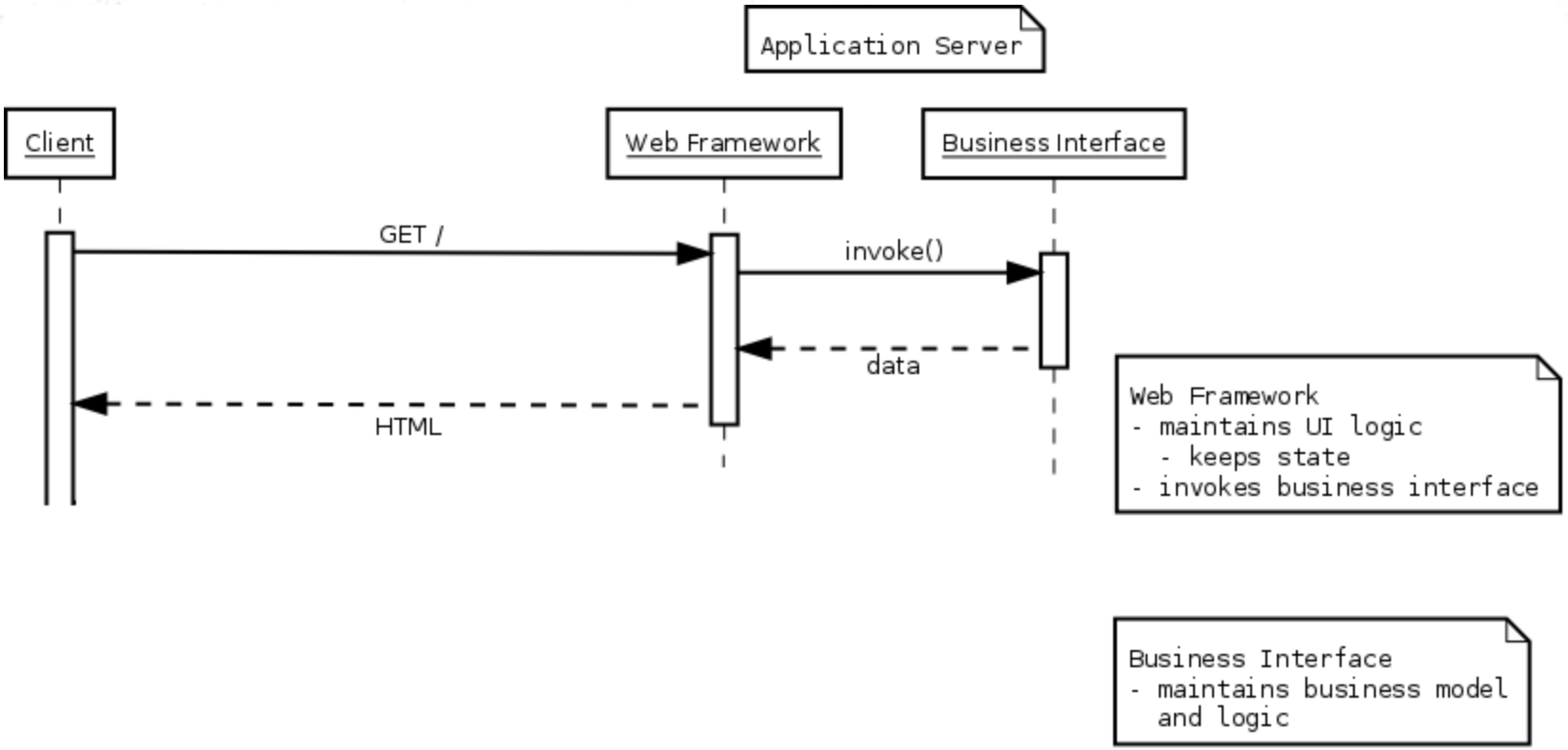
Server-Side

Web Architecture

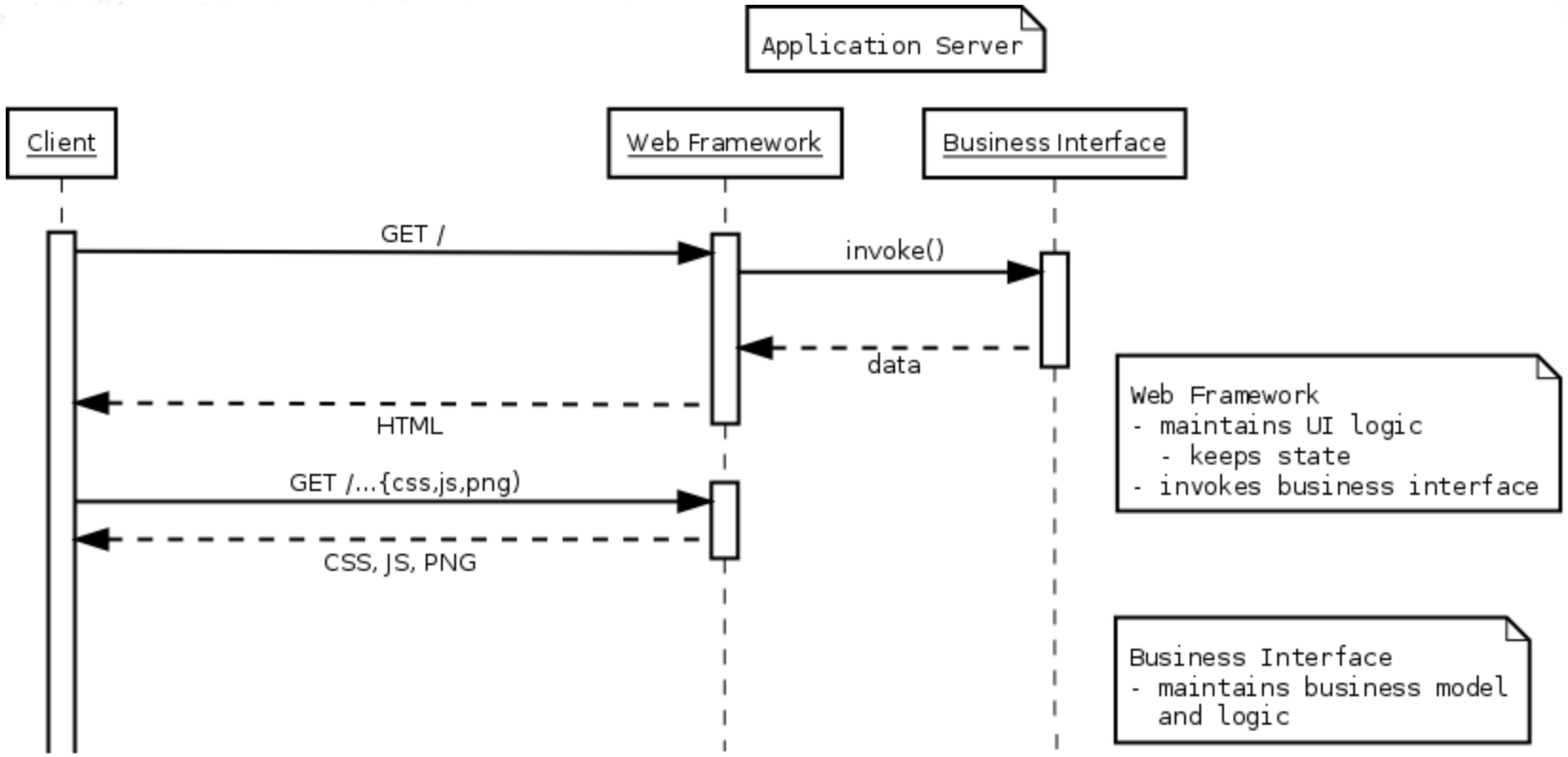
{Client vs. Server}-Side Web

- Server-Side Web (Thin Client)
  - well-established approach
  - 90's, 00's
- Client-Side Web (Thick Client)
  - modern approach
  - SPA (Single Page Applications)
  - Leverages enhancements in web standards & protocols
  - 10's

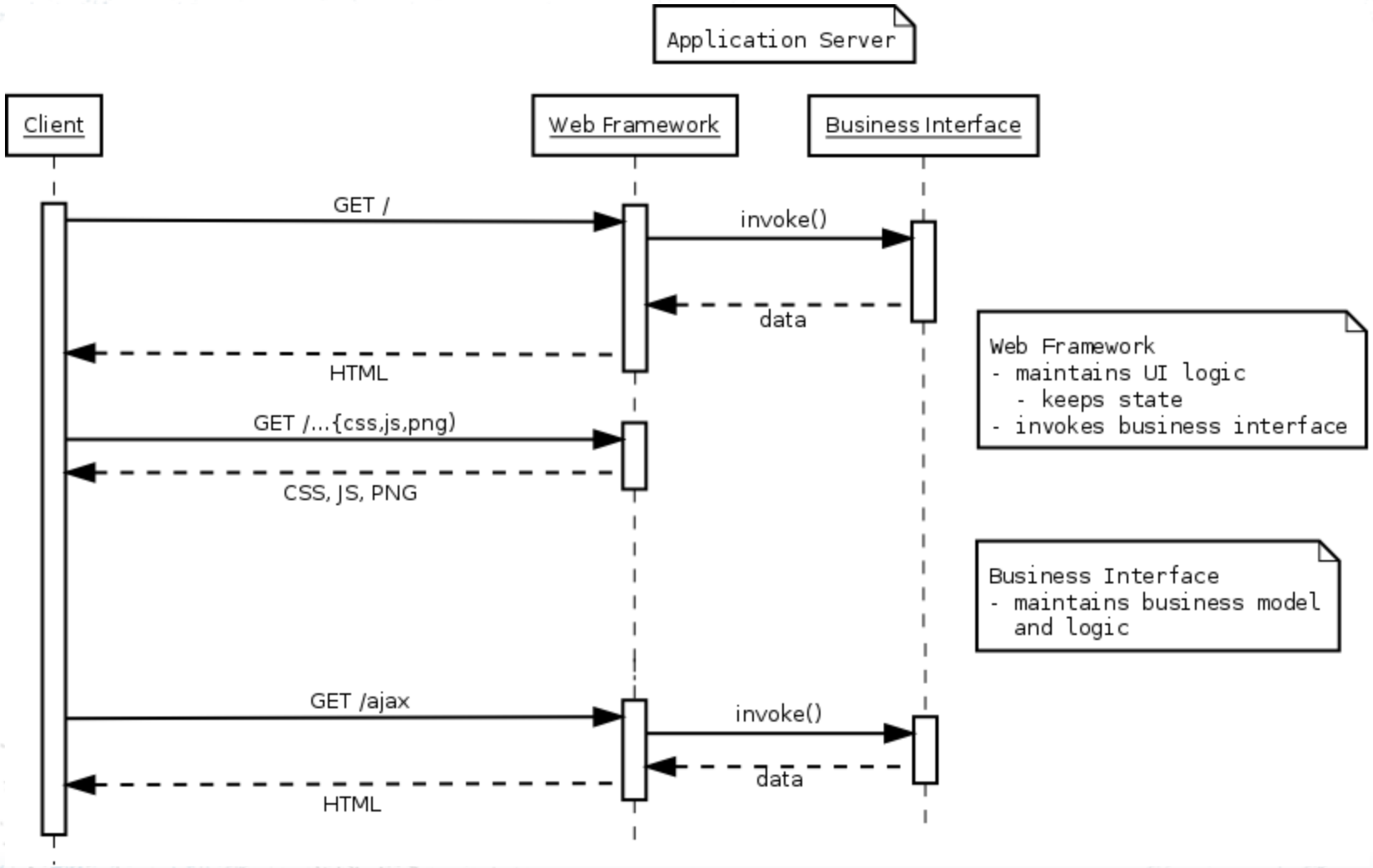
Server-Side


Application Server
Client
Web Framework
Business Interface
GET /
invoke()
data
HTML
Web Framework
- maintains UI logic
- keeps state
- invokes business interface
Business Interface
- maintains business model
and logic

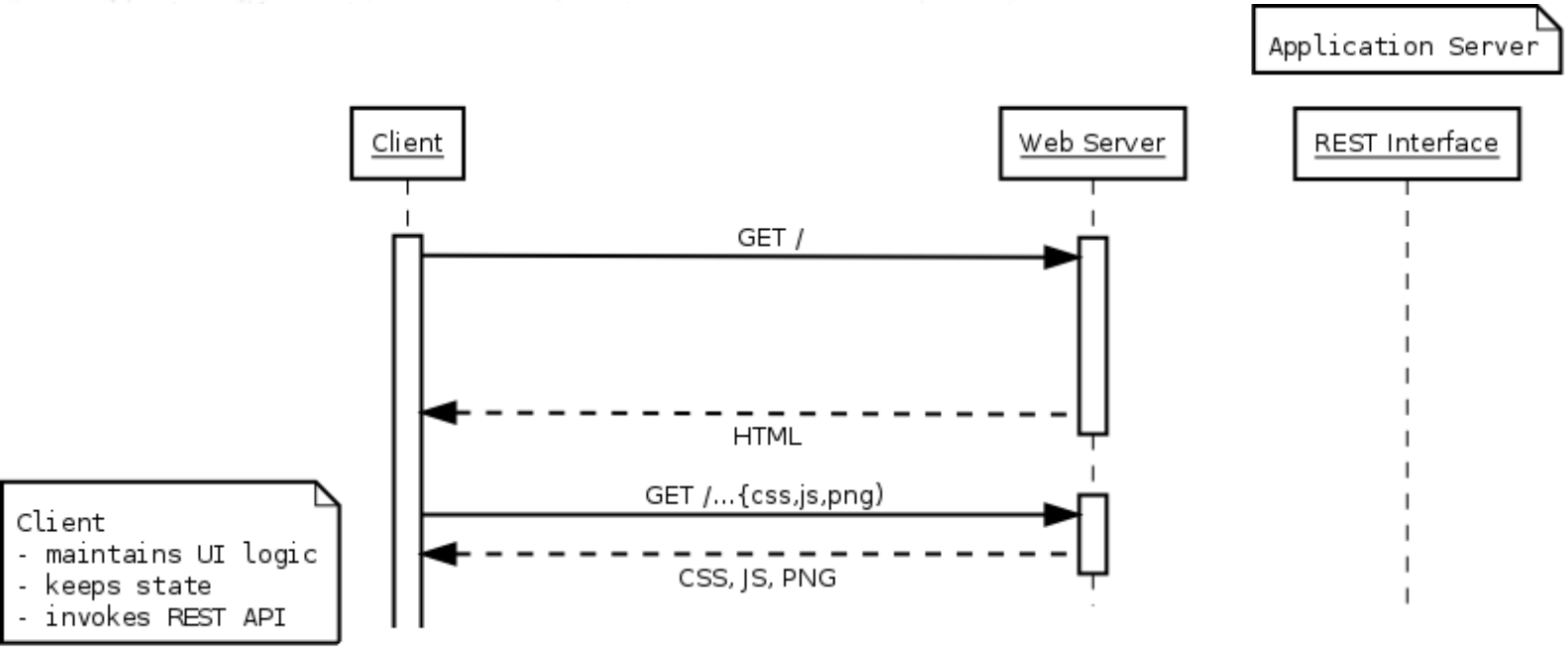
Server-Side


Application Server
Client
Web Framework
Business Interface
GET /
invoke()
data
HTML
GET /...{css,js,png}
CSS, JS, PNG
Web Framework
- maintains UI logic
- keeps state
- invokes business interface
Business Interface
- maintains business model
and logic

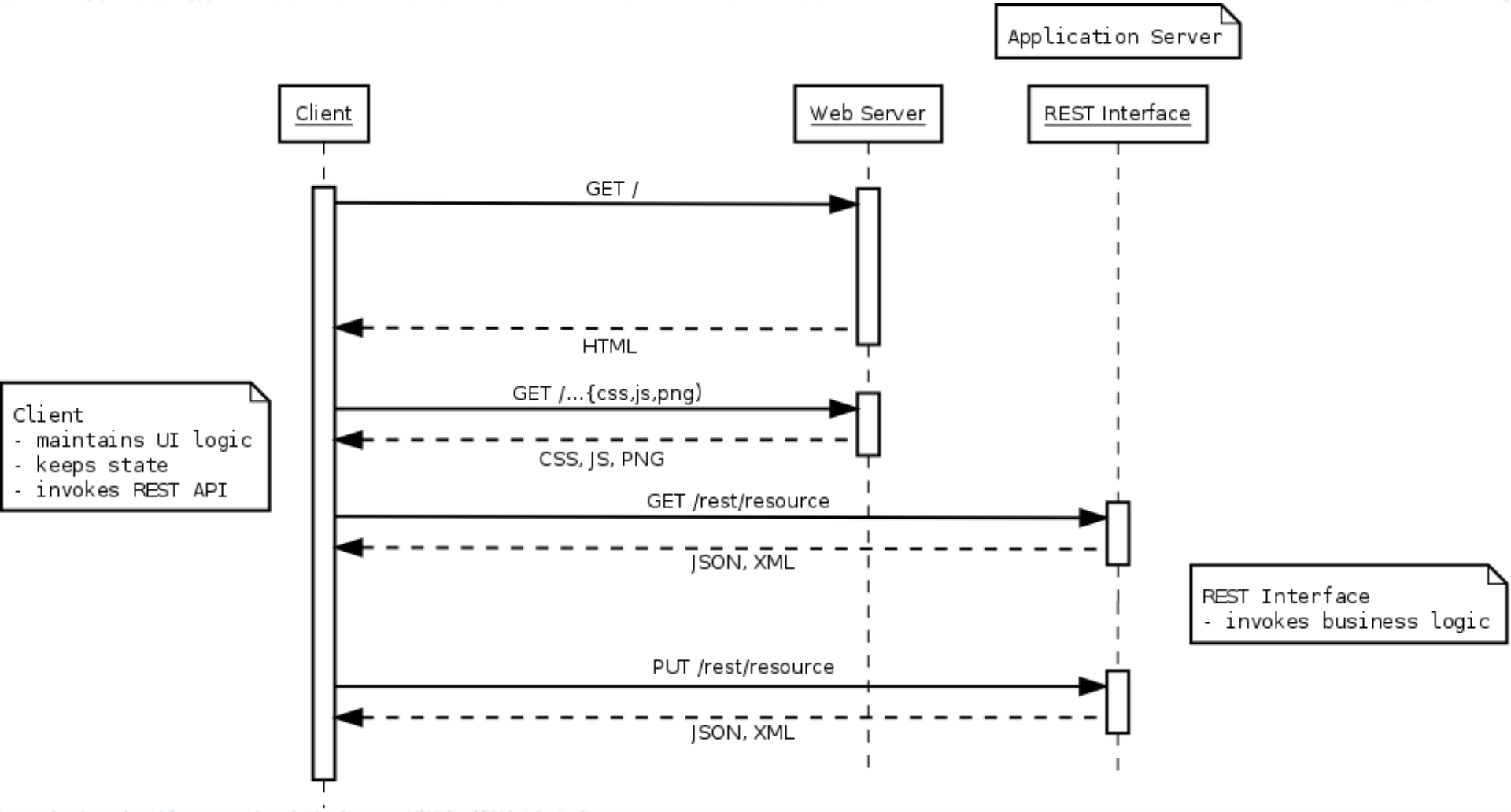
Server-Side


Application Server
Client
Web Framework
Business Interface
GET /
invoke()
data
HTML
GET /...{css,js,png)
CSS, JS, PNG
GET /ajax
invoke()
data
HTML
Web Framework
- maintains UI logic
- keeps state
- invokes business interface
Business Interface
- maintains business model
and logic

Client-Side


Application Server
Client
Web Server
REST Interface
GET /
HTML
GET /...{css,js,png}
CSS, JS, PNG
Client
- maintains UI logic
- keeps state
- invokes REST API

Client-Side


Application Server
Client
Web Server
REST Interface
GET /
HTML
GET /...{css,js,png)
CSS, JS, PNG
GET /rest/resource
JSON, XML
PUT /rest/resource
JSON, XML
Client
- maintains UI logic
- keeps state
- invokes REST API
REST Interface
- invokes business logic

Client-Side Web Approach

- Off-loading server
  - Stateless, Scalable, More battery demands
- Client-Side Frameworks
  - Angular, React, Ember, Backbone, ...
- Standards improvements
  - HTML5 + Protocols
- REST interfaces
  - Data-oriented, presentation independent

Server-Side Advantages

- Well-Known
- No need for another layer of abstraction
- Full granularity and control over the content

Java API for RESTful Services

JAX-RS 2.0

JAX-RS Origins

- RESTful Principles
  - Assign everything an ID
  - Link things together
  - Use common methods (GET, POST, …)
  - Stateless communication

JAX-RS 1.0 Goals

- POJO-Based API
- HTTP Centric
- Format Independence

JAX-RS API

- Application Resources
  - `@ApplicationPath (or web.xml)`
  - `@Path`
- HTTP methods
  - `@GET / @POST / @PUT / @DELETE /..`
- Parameters
  - `@PathParam / @QueryParam / ...`
- Media-Type
  - `@Consumes / @Produces`

Demo JAX-RS Endpoint

- http://javaee-samples.github.io/
- From root of samples (i.e javaee7-samples directory):

```
mvn –DskipTests=true clean install
```

- From javaee7-samples/ {path to sample} directory:

```
mvn test
```

- Start and deploy to server

```
<server> ./bin/standalone.sh
<sample> mvn wildfly:deploy
```

HTTP Method Purpose

Method	Meaning
@GET	Read, possibly cached
@POST	Modify or create **with** a known ID (modify/update)
@PUT	Modify or create **without** a known ID (create/modify)
@DELETE	Remove
@HEAD	GET with no response
(@PATCH)	json-patch

http://stackoverflow.com/questions/630453/put-vs-post-in-rest

http://stackoverflow.com/questions/25253899/how-to-implement-patch-requests-in-resteasy

Parameter Injection

Annotation	Example
`@PathParam("id")`	`@Path("/consumer/{id}")`
`@QueryParam("query")`	`GET /consumer/search?query=???`
`@CookieParam("username")`	`Cookie: ...`
`@HeaderParam("Authorization")`	`Header: Authorization: ...`
`@FormParam("inputName")`	`@Consumes("multipart/form-data")`
`@MatrixParam("query")`	`GET /consumer;name=???/orders`

http://stackoverflow.com/questions/630453/put-vs-post-in-rest

http://stackoverflow.com/questions/25253899/how-to-implement-patch-requests-in-resteasy

New in JAX-RS 2.0

- New Features
  - Client API
  - Filters and Interceptors
  - Asynchronous API
  - Hypermedia
- Improvements
  - Content-Type Negotiation
  - Validation Alignments

Demo

JAX-RS – Client API

Filters and Interceptors

- Customize JAX-RS
  - via well-defined extension points
- Use cases:
  - Caching
  - Logging
  - Compression
  - Security
- Shared between server & client

Filters

- Non-wrapping extension points
  - Pre: RequestFilter
  - Post: ResponseFilter
- Each filter decides to proceed or break chain
  - ContainerRequestContext#abortWith

Interceptors

- Wrapping extensions points
  - ReadFrom: `ReaderInterceptor`
  - WriteTo: `WriterInterceptor`
- Each handler decides to proceed or break chain
  - by calling `ctx.proceed();`

Asynchronous

- Let “borrowed” threads run free!
  - Suspend and resume connections
    - Suspend while waiting for an event (`@Suspended AsyncResponse`)
    - Resume when event arrives
- Client API support
  - Future<T>, InvocationCallback<T>

Demo

JAX-RS – Asynchronous

Demo

JAX-RS – Bean validation

Hypermedia

- extension to Hypertext concept
- enables you to send the client links (in http header or as response payload)
- `javax.ws.rs.core.Link`
- hypermedia content is represented by link – indicating other options
- HATEOAS (“hypermedia as the engine of application state”)
- `javax.ws.rs.core.UriBuilder`
- `javax.ws.rs.core.UriInfo`
  - provides access to application URI information

Java API for JSON Processing

JSON-P

Motivation: JSON

- JavaScript Object Notation
  - The format of the Web
    - Comes from JavaScript object syntax
    - Human readable
  - Language independent
    - Standard parsers in many languages
  - Key-value Pair Format

```
{ “firstName”: “John”, “lastName”: “Smith” }
```

Motivation: Java API for JSON

- Lot of vendor-dependent APIs
  - Need for standardization
- Standard API for JSON processing
  - generate, parse

JSON-P APIs

- Streaming API
  - Similar to StAX
- Object Model APIs
  - Similar to XML DOM

Demo

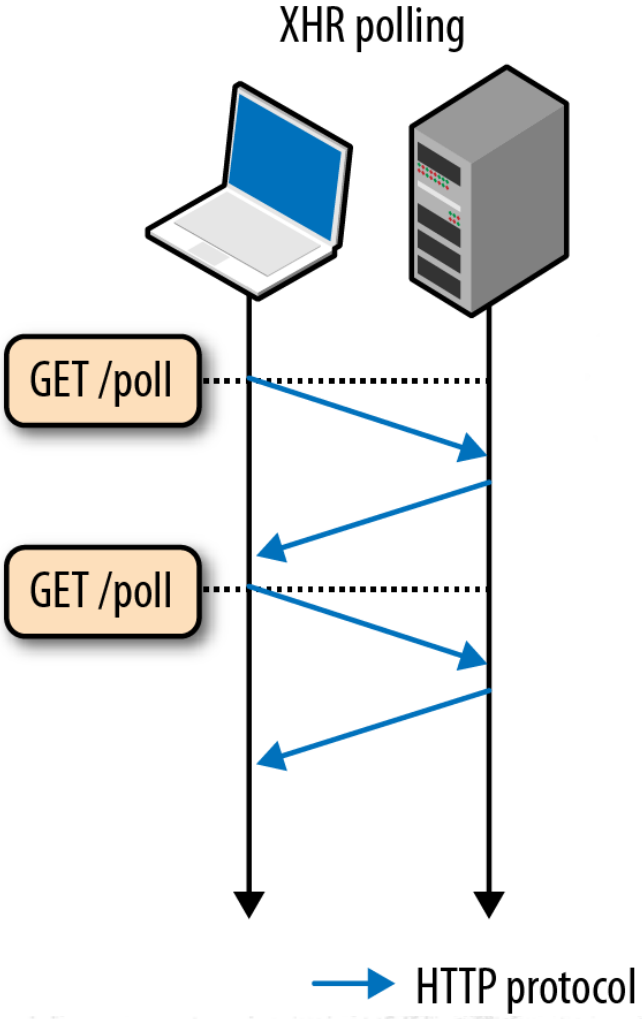
JSON Object Model API

Java API for WebSocket

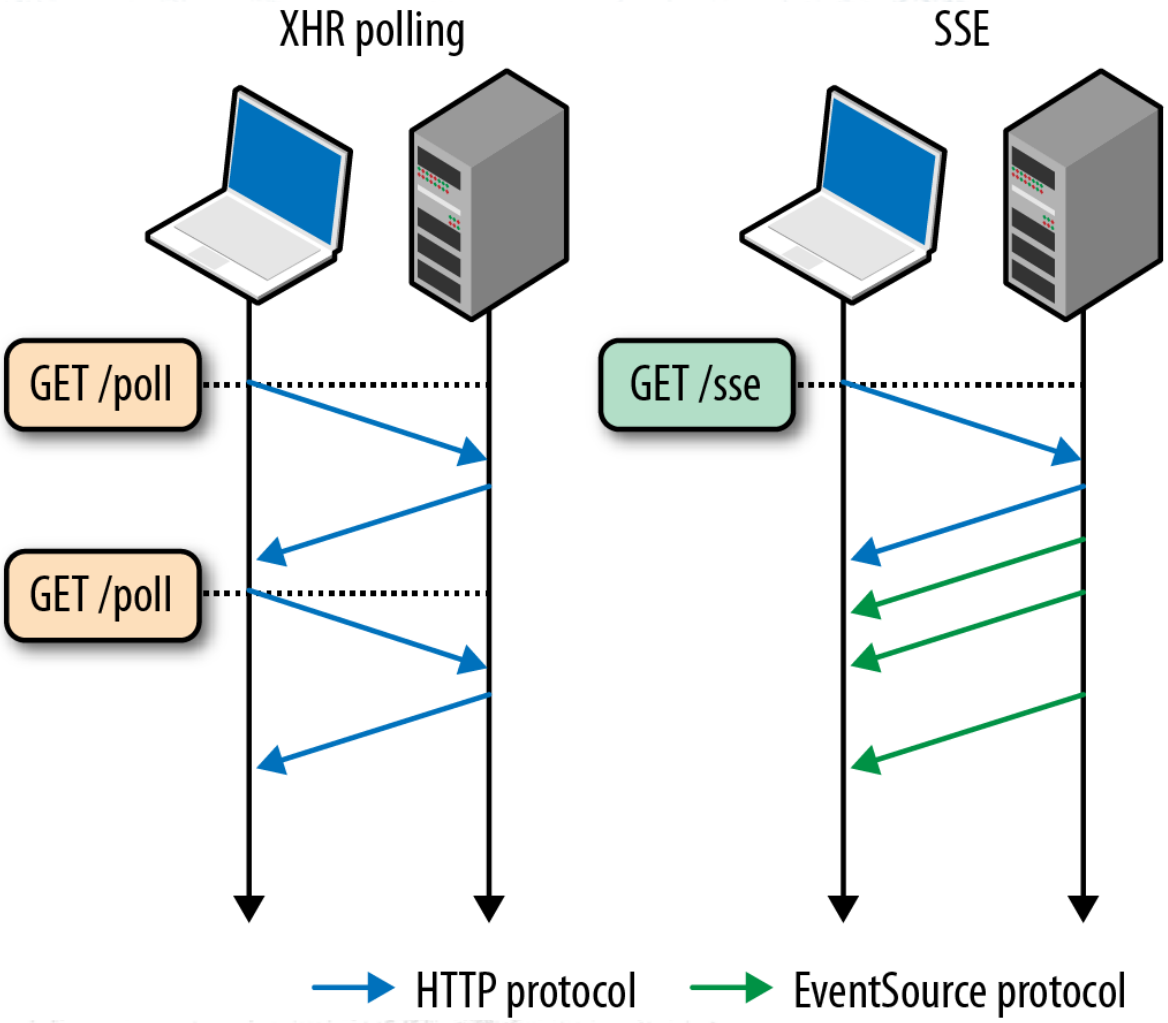
Motivation

- HTTP is half-duplex
- HTTP is inefficient
- HTTP hacked to achieve Push

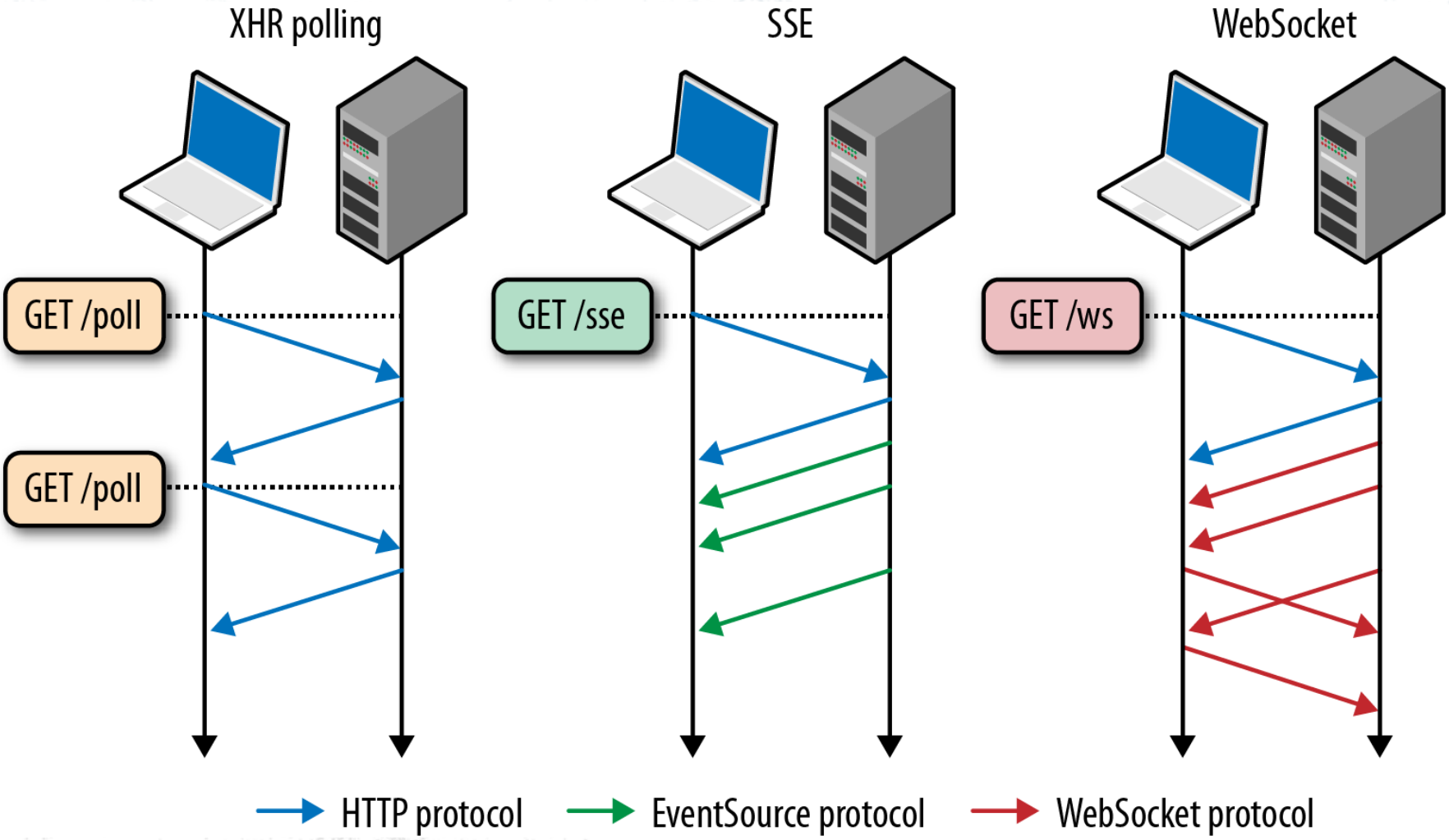
Server Push - Polling



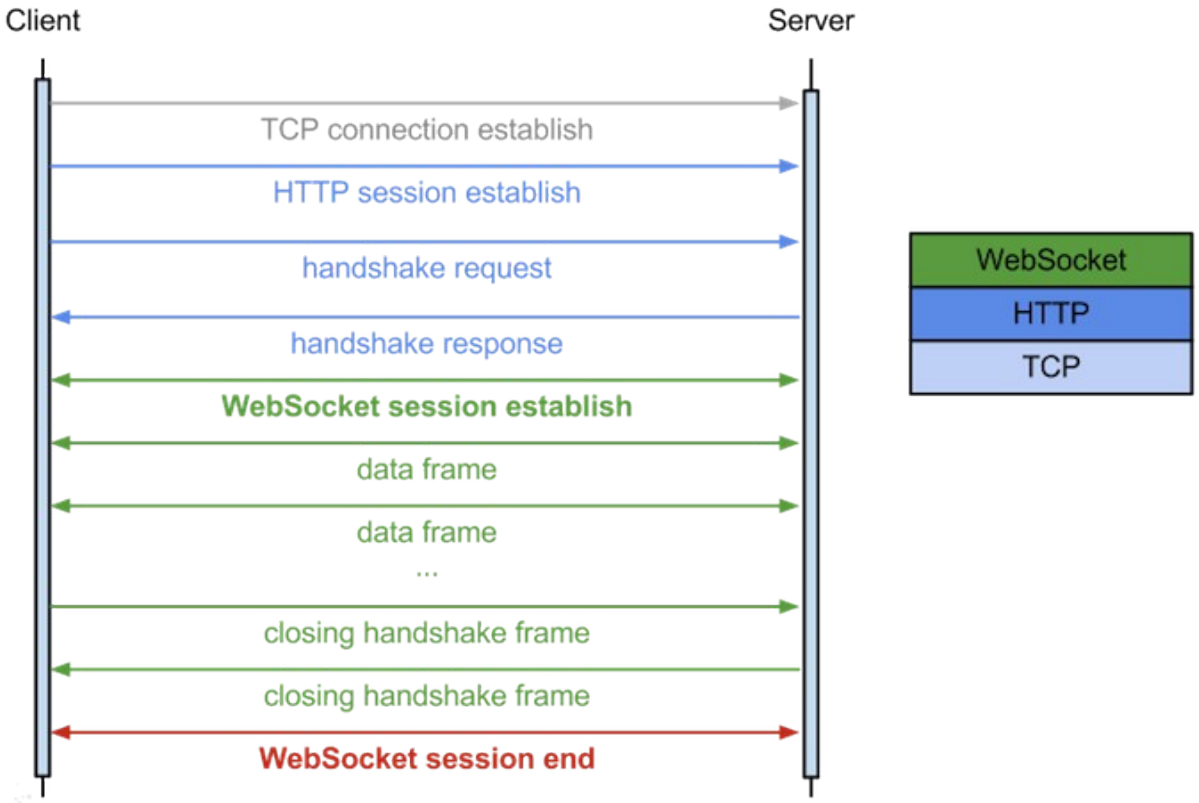
Server Push - Polling



Server Push - Polling



Handshake


Client
Server
TCP connection establish
HTTP session establish
handshake request
handshake response
WebSocket session establish
data frame
data frame
...
closing handshake frame
closing handshake frame
WebSocket session end
WebSocket
HTTP
TCP

WebSocket

- Full duplex & efficient communication
- A component of HTML5
  - JavaScript API under W3C
  - Protocol under IETF (Internet Engineering Task Force)
- Wide support for browsers
  - http://caniuse.com/#feat=websockets

WebSocket: Limitations

- Use of existing infrastructure
  - Proxies doesn't have to handle connection upgrade
- Fallback mechanisms
  - Atmosphere

WebSocket: Trade-offs

- WebSocket
  - Low efforts to maintain TCP connection
  - Limited by number of available ports
  - **Highly interactive applications**
- HTTP
  - Resource-consuming protocol
  - **Fairly interactive applications**

WebSocket: Use Cases

- Realtime, truly low latency
  - Chat applications
  - Live sports ticker
  - Realtime updating social streams
  - Multiplayer online games
- Requires architecture shift to
  - Non-blocking IO
  - Event queues

Java API for WebSocket

- Programmatic
- Annotation based
  - Our focus

WebSocket Annotations

- `@ServerEndpoint`
  - `@OnOpen`
  - `@OnMessage`
  - `@OnClose`

Demo

WebSocket Whiteboard

Method Parameters

- `Session`
- Implicitly supported types
  - `String, byte[]`
  - `JsonArray, JsonObject`
- More types supported by Encoders

Integration to Java EE 7

- Relation to Servlet 3.1
  - `HttpServletRequest.upgrade(ProtocolHandler)`
- Dependency Injection
  - CDI beans
  - EJB beans
- Security
  - `ws://... vs. wss://…`
  - `web.xml: <security-constraint>`

AngularJS

AngularJS

- Client side JavaScript Framework for adding interactivity to HTML
- MVC
- Adding behaviour to HTML through **directives**
- `<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>`

AngularJS bootstrap – app.js

- `<html ng-app="your_module_name">`
- app.js

```
var testApp = angular.module('testApp',
[]);
```

- Start booting process for application
- Defining controllers as dependencies of application using inline array annotations

AngularJS bootstrap – app.js

- Define route for main page (`ngRoute`)
- So how does it look like?
- ```
  var testApp = angular.module('testApp',
  ['ngRoute',
   'myFirstController'
  ]);
  ```
  ```

AngularJS - directives

- Directive is a marker on a HTML tag that tells Angular to run or reference some JS code
- Enhance functionality of the template/presentation logic
- Simplify model and view
- Option to build custom directives!!
- `ng-include, ng-repeat, ng-model, ...`

AngularJS – HTML templates

- In HTML template you can use:
  - Expressions – allows you to insert dynamic values e.g `{{ “Hello world” }}, {{ message.name }}`
    - Filters – format or filter values e.g

      ```
      {{message.date | date:'yyyy-MM-dd HH:mm:ss Z'}}
      ```
  - Directives – e.g `ng-repeat="message in messageList"`
  - Custom directives – could be attribute, element, class, … e.g `<message></message>`

AngularJS – custom directive

- much like controllers, directives are registered on modules
- to register use `module.directive` which returns an object
-
```
msgDirective.directive('message', function
() {
    return {
     restrict: 'E',
     templateUrl: 'templates/message.html'
    }; })
```

AngularJS – controllers.js

- Initializing the model (Set up the initial state of `$scope` object)
- Adding behavior to the `$scope` object (defining functions)
- `$scope` is an object that refers to the application model. Used to gain access to the model related to a particular controller
- Scopes are arranged in hierarchical structure (`$scope` has parent scope)

AngularJS – controllers.js

- No presentation logic and no formatting !!
- Related directives:
  - `ng-controller` – attaches a controller to the DOM
  - `ng-model` – binds elements to the `$scope`
  - `ng-submit` – submit a form

AngularJS – services.js

- a service is a function, or object, that is available for, and limited to, your AngularJS application.
- register/create with service **factory** function with an Angular module
- ```
  myModule.factory('serviceId', function() {
    var shinyNewServiceInstance;
    // factory function body that constructs shinyNewServiceInstance
    return shinyNewServiceInstance;
  });
  ```
  ```

AngularJS – services.js

- some predefined Angular services:
  - `$location`, `$log`, `$window`, ...

AngularJS – services.js
communication with REST

- two ways to communicate with REST
  - `$http` service – low level interaction using `XMLHttpRequest`
  - `$resource` object – high level approach (we are going to use it in our lab)
- `$resource(url, [paramDefaults], [actions], options);`

Summary

- JAX-RS
  - RESTful endpoints, SPA, stateless
- WebSocket
  - Realtime bi-directional communication
- JSON-P
  - Standardization of JSON processing
- AngularJS
  - ClientSide JavaScript framework

Thank you!

You can reach me at tremes@redhat.com