ALTRI PARADIGMI DI PROGRAMMAZIONE

A.A. 2020/2021

Laurea triennale in Informatica

17: Fallacies and Frameworks

8 FALLACIES OF DISTRIBUTED COMPUTING

Fallacia: (s,f) l'essere fallace, ovvero che può trarre in inganno, insidioso.

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn’t change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

[1-4]: Bill Joy, Dave Lyon, early 1990

[1-7]: L. Peter Deutsch, 1994

[8]: James Gosling, 1997

THE NETWORK IS RELIABLE

Non è necessario un guasto hardware: ci sono moltissimi modi in cui una rete può essere inaffidabile.

Un software distribuito deve tener conto della possibilità che le connessioni si interrompano o vengano negate, o i messaggi vadano persi.

Va bilanciato il rischio/costo di un tale evento con il costo della ridondanza e dell'affidabilità.

LATENCY IS ZERO

C'è un limite fisico alla velocità di un messaggio: la velocità della luce.

E' impossibile, per es. raggiungere una latenza inferiore ai 30ms fra le due coste dell'Atlantico.

Se il messaggio trasporta delle informazioni che dipendono dal tempo per una risoluzione maggiore di qualche secondo, la latenza dev'essere calcolata come parte della comunicazione .

BANDWIDTH IS INFINITE

La banda disponibile è sempre meno un problema, ma i dati da trasferire tendono ad aumentare altrettanto (o più) velocemente.

Inoltre, in molte situazioni i protocolli di rete danno priorità all'affidabilità della comunicazione invece che all'economia di trasmissione.

Se il messaggio da trasferire è particolarmente grande, va gestito come un insieme di parti trasferite singolarmente.

Le performance di un protocollo devono considerare anche la quantità di dati "amministrativi" che trasporta.

THE NETWORK IS SECURE

La diffusione delle reti ha portato, ovviamente, a maggiori opportunità per gli attaccanti e a maggiori possibilità di errore per i costruttori di dispositivi.

Se il contenuto del messaggio ha una qualche importanza o trasporta qualche informazione minimamente confidenziale, il protocollo deve essere reso sicuro:

- autenticato
- autorizzato
- integro
- confidenziale

TOPOLOGY DOESN'T CHANGE

L'ubiquità dell'accesso ha come conseguenza che un dispositivo è spesso in movimento da un punto di accesso ad un altro.

Il risultato è che la topologia del grafo che costituisce la rete è in continua evoluzione.

E' necessario che la comunicazione sia in grado di usare un metodo di indirizzamento dinamico e che non dipenda dalle caratteristiche fisiche della rete.

THERE IS ONE ADMINISTRATOR

La diffusione e frammentazione delle reti significa anche frammentazione delle responsabilità nella gestione delle reti stesse.

Sempre più di rado si può imporre requisiti al percorso delle connessioni (porte aperte, feature del trasporto) perché sempre più organizzazioni sono coinvolte nella loro gestione.

Bisogna puntare sempre alla massima semplicità nella struttura e nei requisiti del protocollo di comunicazione.

Requisiti particolari potrebbero non poter essere soddisfatti da uno qualsiasi dei nodi della rete, impedendo il transito dei dati.

TRANSPORT COST IS ZERO

Anche il costo del trasporto sta continuamente calando, ma non è mai zero. Sia in termini monetari, sia anche in termini energetici: la maggior parte dei dispositivi è a batteria.

Nel caso di dispositivi IoT, il bilancio energetico diventa il costo preponderante.

Come per altre voci, efficienza e sintesi devono essere criteri importanti nella costruzione di un protocollo di comunicazione.

THE NETWORK IS HOMOGENEOUS

Le reti non sono più costituite da una sola tecnologia, da un solo mezzo di comunicazione; un messaggio attraversa una varietà di mezzi trasmissivi, metodi di comunicazione, e apparati differenti.

Come già detto, il protocollo di comunicazione deve evitare di richiedere caratteristiche di trasporto particolari, ma affidarsi il più possibile alle funzioni più standard.

WEB FRAMEWORK

Abbiamo visto come costruire manualmente un server ed un client di rete.

Normalmente però non si lavora ad un livello così basso, ma si usano delle astrazioni:

- per semplificare la struttura del codice
- per sfruttare implementazioni già testate e corrette

Vediamo che forma prende uno degli esempi delle scorse lezioni implementato con una di queste astrazioni.

Vert.X è la proposta di Eclipse Foundation per la realizzazione di applicazioni web e microservizi scalabili e moderni.

Le sue principali caratteristiche sono:

- modularità e compattezza delle dipendenze
- event-driven, fortemente asincrono
- poliglotta
- altamente scalabile

- approccio non-standard
- "unopinionated"
- semplice e pragmatico
- open source (Apache 2)
- supportato dalla fondazione Eclipse

Usando i moduli vertx-core e vertx-web possiamo reimplementare il server TicTacToe come applicazione web e valutare le modifiche al protocollo e al codice che ci troviamo a fare.

ESEMPIO - API

Per prima cosa dobbiamo pianificare l'API che il servizio esporrà.

Per rendere l'esempio più concreto, esporremo sia una API Web usabile da browser, sia una JSON usabile da un client automatico.

API DESIGN

Creare una API è un'arte: fatta di stili, di scuole, di preferenze, di gusti personali e richiede esperienza.

L'interfaccia del nostro codice è il modo in cui comunichiamo agli altri sviluppatori cosa il nostro codice fa e come va usato.

HOME PAGE

`GET /`

HTML: Pagina HTML di benvenuto

INGRESSO AL GIOCO

`POST /game`

HTML: redirect al form di gioco

JSON: url dove ottenere lo stato del gioco

MOSSA IN UN GIOCO

`POST /game/{id}`

`move=x`

HTML: form con lo stato di gioco

JSON: stato del gioco

COME CHIAMARE L'API

```
$ curl -X POST -H "Accept: application/json" \
  http://localhost:8080/game
{"game":"/game/1134141953"}

$ curl -H "Accept: application/json" \
  http://localhost:8080/game/1134141953
```

STRUMENTI UTILI

cURL - https://curl.haxx.se/

jq - https://stedolan.github.io/jq/

ESEMPIO - IMPLEMENTAZIONE

L'astrazione che il modulo vertx-web ci mette a disposizione è un `router` che ci consente di associare ad una URL una lambda.
Questa può modificare la risposta impostando esito e contenuto.

Il modo in cui il nostro codice interagisce con il framework è definito da questa astrazione.

it.unipd.app2020.web.NetGame

```
class NetGame {
  String[] playerId = new String[2];
  private Optional< Game > game =
    Optional.empty();
```

it.unipd.app2020.web.GameServer

```
class GameServer {
  // active and completed games
  ConcurrentMap< String, NetGame > games =
    new ConcurrentHashMap<>();
  // awaiting players
  BlockingQueue< NetGame > openings =
    new LinkedBlockingQueue<>(16);
```

```
public Optional< GameIndex >
  status(String playerId) {
  var res = Optional.empty();
  if (games.containsKey(playerId) &&
    games.get(playerId).started()) {
    int idx = games.get(playerId).playerId[0]
      .equals(playerId) ? 0 : 1;
    res = Optional.of(new GameIndex(
      idx, games.get(playerId).status()));
  }
  return res;
}
```

it.unipd.app2020.web.Server

```
public static void main(String[] args) {

  // The Game Server
  GameServer gameServer = new GameServer();

  // The infrastructure parts
  Vertx vertx = Vertx.vertx();
  HttpServerOptions options = new HttpServerOptions()
    .setLogActivity(true);
  HttpServer server = vertx.createHttpServer(options);
  Router router = Router.router(vertx);
  // setup body handling for all routes
  router.route().handler(BodyHandler.create());
```

```
// Browser entry point
router.get("/").produces("text/html").handler(ctx -> {
  ctx.response().end(JOIN_FORM);
});
```

```
// POST /game - want to play
router.post("/game").produces("text/html").handler(ctx -> {
  GameLocation loc = new GameLocation(gameServer.create());
  ctx.response().setStatusCode(302)
    .putHeader("Location", loc.game).end();
});

router.post("/game").produces("application/json")
  .handler(ctx -> {
    GameLocation loc = new GameLocation(
      gameServer.create());
    ctx.response().putHeader("content-type",
      "application/json").end(location.toJson(mapper));
});
```

```
// POST /game/{id} move=x - play move x
router.post("/game/:playerId").produces("text/html")
  .handler(ctx -> {
    String playerId = ctx.request().getParam("playerId");
    boolean open = gameServer.open(playerId);
    if (!open)
      ctx.response().setStatusCode(404).end(GAME_NOT_FOUND);
```

```
String result = gameServer.status(playerId)
  .map((res) -> {
    int idx = res.idx; GameResult status = res.status;
    int move = Integer
      .valueOf(ctx.request().getParam("move"));
    if (idx == status.next && status.valid && !status.end)
      status = gameServer.player(playerId, move)
        .orElse(status);
      return render(playerId, idx, status);
    }).orElse(String.format(WAIT_FOR_ANOTHER, playerId));

    ctx.response().end(result);
});
```

```
router.post("/game/:playerId").produces("application/json")
  .handler(ctx -> {
    // ...
      return renderJson(playerId, idx, status);
    }).orElse(new GameStatus("wait for another")
      .toJson(mapper));

  ctx.response().end(result);
});
```

```
router.get("/game/:playerId").produces("text/html")
  .handler(ctx -> {
    String playerId = ctx.request().getParam("playerId");
    boolean open = gameServer.open(playerId);
    if (!open)
      ctx.response().setStatusCode(404).end(GAME_NOT_FOUND);

    String result = gameServer.status(playerId).map(
      (res) -> render(playerId, res.idx, res.status))
      .orElse(String.format(WAIT_FOR_ANOTHER, playerId));
  ctx.response().end(result);
});
```

```
public static void main(String[] args) {
  // ...
  server.requestHandler(router).listen(8080);
}
```

ESEMPIO - OSSERVAZIONI

L'approccio del framework Vert.X è tipico dei framework asincroni: ci viene fornito un ambiente all'interno del quale possiamo specificare gli handler che verranno richiamati in seguito a eventi definiti dal framework.

Le conseguenze sono:

- con un po' di disciplina, possiamo mantenere il codice che interagisce con il framework all'interno dell'handler
- gli handler compiono effetti collaterali, non sono quindi funzioni pure

- la struttura dichiarativa costringe ad esternalizzare l'organizzazione del codice
- è in carico a noi la gestione coerente dello stato condiviso

PERCHÉ USARE UN FRAMEWORK

Un framework fornisce un ambiente all'interno del quale un insieme di casi d'uso fondamentali è reso facile, efficiente e sicuro da implementare.

Un framework per applicazioni web, per esempio, rende facile:

- specificare le rotte a cui rispondere
- costruire le risposte

Lo stesso framework web, cercherà di rendere trasparente, ed eventualmente configurabile:

- la gestione dei dettagli del protocollo
- la sicurezza nel trattamento della comunicazione
- la suddivisione delle risorse fra le varie parti del sistema

Gli autori del framework scelgono le loro priorità fra semplicità d'uso, sicurezza ed efficienza.

Come utenti, noi otteniamo automaticamente ogni incremento su ciascuna di queste dimensioni.

PERCHÉ NON USARE UN FRAMEWORK

Un caso d'uso che non è fra quelli prescelti come importanti dagli autori del framework, può essere molto, molto complesso da implementare.

Un framework per applicazioni web, per esempio, può rendere difficile:

- rispondere a richieste esotiche
- controllare finemente l'erogazione della risposta

Lo stesso framework web, potrebbe rendere oscuro, oppure inaspettatamente:

- fallire l'esecuzione di una richiesta a causa di un baco nella sua implementazione
- aprire una falla di sicurezza a causa di un default errato
- permettere il sovraccarico del sistema a causa di una mancata limitazione delle risorse

Gli autori del framework scelgono le loro priorità fra semplicità d'uso, sicurezza ed efficienza.

Come utenti, dipendiamo completamente da loro per l'ordine con cui queste caratteristiche sono implementate, e dobbiamo sopportare i costi di aggiornamento se l'interfaccia fra il framework ed il nostro codice cambia.

PUBBLICITÀ

HUMBLE BUNDLE JAVA BOOKS

https://www.humblebundle.com/books/java-programming-more-oreilly-books

97 THINGS EVERY PROGRAMMER SHOULD KNOW

https://97-things-every-x-should-know.gitbooks.io/97-things-every-programmer-should-know/en/