Erlang Behaviors

Programación Funcional Avanzada

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad Simón Bolívar

UNIVERSIDAD SIMÓN BOLÍVAR

Erlang Behaviors

Behaviours in case you're british or canadian

- Formalización de los patrones de diseño concurrente.
- Módulos que implementan los patrones concurrentes comunes.
  - Separación entre «concurrencia» y «operación».
  - Explotar paradigma funcional del lenguaje – funciones de primera clase, genericidad, y verificación dinámica.
- Bloques fundamentales para construir aplicaciones concurrentes.
- Disponibles en el OTP estándar Erlang.

UNIVERSIDAD SIMÓN BOLÍVAR

Behaviors disponibles

- **Worker behaviors** – hacen «el trabajo».
  - Servidores simples – hacen «una cosa».
  - Servidores transicionales – máquinas de estado.
  - Servidores reactivos – *event driven*.
  - Hojas del árbol de dependencia concurrente.

Behaviors disponibles

- **Worker behaviors** – hacen «el trabajo».
  - Servidores simples – hacen «una cosa».
  - Servidores transicionales – máquinas de estado.
  - Servidores reactivos – *event driven*.
  - Hojas del árbol de dependencia concurrente.
- **Supervisor behaviors** – cuidan de otros procesos.
  - Controlar el ciclo de vida de los *workers*
  - Nodos internos del árbol de dependencia concurrente.

UNIVERSIDAD SIMÓN BOLÍVAR

Behaviors disponibles

- **Worker behaviors** – hacen «el trabajo».
  - Servidores simples – hacen «una cosa».
  - Servidores transicionales – máquinas de estado.
  - Servidores reactivos – *event driven*.
  - Hojas del árbol de dependencia concurrente.
- **Supervisor behaviors** – cuidan de otros procesos.
  - Controlar el ciclo de vida de los *workers*
  - Nodos internos del árbol de dependencia concurrente.
- **Application** – el objetivo final
  - Componente reusable e instalable – el árbol completo.
  - Información sobre dependencias de software.
  - Mnesia, Cowboy/Yaws (HTTP), Agente SNMP, etc.

UNIVERSIDAD SIMÓN BOLÍVAR

Estructura de un Behavior

Todos se parecen igualito

- Módulo con código «genérico»
  - Se «declara» al comienzo del módulo propio.

```
-behavior(supervisor).
```

  - Se usa con llamadas calificadas – `supervisor:foo(...)`

Estructura de un Behavior

Todos se parecen igualito

- Módulo con código «genérico»
  - Se «declara» al comienzo del módulo propio.

```
-behavior(supervisor).
```

  - Se usa con llamadas calificadas – `supervisor:foo(...)`
- Funciones para el modelo de concurrencia particular.
  - Iniciar y posiblemente registrar el proceso.
  - Enviar y recibir mensajes – síncronos y asíncronos.
  - Ocasionalmente, definir el formato de los mensajes.
  - Almacenar y administrar *loop data* («estado» del behavior)
  - Detener el proceso.

UNIVERSIDAD SIMÓN BOLÍVAR

Estructura de un Behavior

Todos se parecen igualito

- Módulo con código «genérico»
  - Se «declara» al comienzo del módulo propio.

```
-behavior(supervisor).
```

  - Se usa con llamadas calificadas – `supervisor:foo(...)`
- Funciones para el modelo de concurrencia particular.
  - Iniciar y posiblemente registrar el proceso.
  - Enviar y recibir mensajes – síncronos y asíncronos.
  - Ocasionalmente, definir el formato de los mensajes.
  - Almacenar y administrar *loop data* («estado» del behavior)
  - Detener el proceso.
- Programador escribe funciones para «llenar los blancos»
  - Behaviors usan *callbacks* antes de que fuera «cool».
  - Compilador verifica que las funciones estén presentes.
  - Cada behavior requiere funciones diferentes.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Iniciar el servidor
  - `gen_server:start/4` – inicio simple.
  - `gen_server:start_link/4` – inicio con dependencia.
  - Ambos son *síncronos* – más determinismo al arrancar.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Iniciar el servidor
  - `gen_server:start/4` – inicio simple.
  - `gen_server:start_link/4` – inicio con dependencia.
  - Ambos son *síncronos* – más determinismo al arrancar.
- *Callback* – `init/1`
  - Debe retornar el *loop data* inicial.
- El programador escribe y exporta:
  - Funciones `start` y `start_link` aprovechando `gen_server:start/4` y `gen_server:start_link/4`.
  - Función `init/1` apropiada.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Enviar mensajes predefinidos («esperados») al servidor
  - `gen_server:cast/2` – mensajes asíncronos.
  - `gen_server:call/2` – mensajes síncronos.
  - `gen_server:call/3` – mensajes síncronos con *timeout*.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Enviar mensajes predefinidos («esperados») al servidor
  - `gen_server:cast/2` – mensajes asíncronos.
  - `gen_server:call/2` – mensajes síncronos.
  - `gen_server:call/3` – mensajes síncronos con *timeout*.
- *Callbacks*
  - `handle_cast/2` – procesar mensaje y generar nuevo *loop data*.
  - `handle_call/2` – procesar mensaje y generar respuesta junto con nuevo *loop data*.
- El programador escribe y exporta:
  - Funciones `handle_cast/2` y `handle_call/2`.
  - Generalmente usando una auxiliar común.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Aceptar mensajes espontáneos («inesperados»)
  - Otro proceso enlazado terminó y quiere avisarlo.
  - Alguien «bendecido y afortunado» descubrió el `Pid`.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Aceptar mensajes espontáneos («inesperados»)
  - Otro proceso enlazado terminó y quiere avisarlo.
  - Alguien «bendecido y afortunado» descubrió el `Pid`.
- *Callback*
  - `handle_info/2` – procesar mensaje y generar nuevo *loop data*.
- El programador escribe y exporta:
  - Función `handle_info/2`.
  - Generalmente haciendo *logging* y reenviando «a quien pueda interesar».
  - No está bien ignorar estos mensajes.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Detener el servidor – por las buenas o por las malas.
  - Recibió un mensaje esperado indicando que debe detenerse.
  - Recibió un mensaje inesperado que le obliga a detenerse.

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Detener el servidor – por las buenas o por las malas.
  - Recibió un mensaje esperado indicando que debe detenerse.
  - Recibió un mensaje inesperado que le obliga a detenerse.
- *Callback*
  - `terminate/2` – considerar la razón y *loop data* en su proceso de limpieza y finalización.
- El programador escribe y exporta:
  - Función `terminate/2`.
  - Es la «simétrica» de `init/1`.
  - Generalmente haciendo *logging* y reenviando «a quien pueda interesar».

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Actualizar el servidor – changing teh codez!
  - ¡Sin detenerlo!
  - Activado por el *Release Handler Subsystem*

UNIVERSIDAD SIMÓN BOLÍVAR

`gen_server` – servidor genérico

Worker Behavior fundamental – cliente/servidor simple

- Actualizar el servidor – changing teh codez!
  - ¡Sin detenerlo!
  - Activado por el *Release Handler Subsystem*
- *Callback*
  - `code_change/3` – realizar cualquier conversión de representación interna y generar el nuevo *loop data*.
- El programador escribe y exporta:
  - Función `code_change/3`.
  - «No hacer nada» es una opción – ¡pero reportarlo!

UNIVERSIDAD SIMÓN BOLÍVAR

Más Worker Behaviors

Servidores con comportamiento estructurado

- `gen_fsm`
  - Servidores modelados como *Finite State Machines*
  - Cada *callback* debe producir la respuesta e indicar el nuevo estado – Máquina de Moore o Mealy.
  - SMTP, IMAP, POP, ...
  - Pronto será reemplazado por `gen_statem`.

UNIVERSIDAD SIMÓN BOLÍVAR

Más Worker Behaviors

Servidores con comportamiento estructurado

- `gen_fsm`
  - Servidores modelados como *Finite State Machines*
  - Cada *callback* debe producir la respuesta e indicar el nuevo estado – Máquina de Moore o Mealy.
  - SMTP, IMAP, POP, ...
  - Pronto será reemplazado por `gen_statem`.
- `gen_event`
  - Servidores modelados como reactivos a eventos.
  - Ciclo cerrado atento a eventos que los despacha a los *callbacks*.
  - Alarmas, *loggers*, ...

UNIVERSIDAD SIMÓN BOLÍVAR

Más Worker Behaviors

Servidores con comportamiento estructurado

- `gen_fsm`
  - Servidores modelados como *Finite State Machines*
  - Cada *callback* debe producir la respuesta e indicar el nuevo estado – Máquina de Moore o Mealy.
  - SMTP, IMAP, POP, ...
  - Pronto será reemplazado por `gen_statem`.
- `gen_event`
  - Servidores modelados como reactivos a eventos.
  - Ciclo cerrado atento a eventos que los despacha a los *callbacks*.
  - Alarmas, *loggers*, ...

Lean el *«OTP Design Principles»*

UNIVERSIDAD SIMÓN BOLÍVAR

`supervisor` – gerente de procesos (pun intended)

Procesos explotados por procesos

- Iniciar hijos, supervisarlos y actuar cuando terminen.
- Descrito con una tupla con nombre, acciones e hijos.
- Acciones especificadas al iniciar el supervisor
  - Estrategia de reinicio – `one_for_one`, `one_for_all`, `rest_for_one`
  - Cantidad de reinicios permitidos en un intervalo de tiempo.
- Hijos a supervisar – lista de especificaciones
  - Módulo, función y argumentos iniciales.
  - Condiciones de reinicio – `transient`, `temporary` o `permanent`.
  - Tiempo máximo en `terminate` – to kill it with fire from the sky!
  - Tipo de hijo – `worker` o `supervisor`

UNIVERSIDAD SIMÓN BOLÍVAR

`supervisor` – gerente de procesos (pun intended)

Procesos explotados por procesos

- Iniciar el supervisor
  - `supervisor:start/2`
  - `supervisor:start_link/2`

`supervisor` – gerente de procesos (pun intended)

Procesos explotados por procesos

- Iniciar el supervisor
  - `supervisor:start/2`
  - `supervisor:start_link/2`
- *Callback*
  - `init/1` – debe preparar la tupla descriptiva
- El programador escribe y exporta:
  - La función `init/1` adecuada.
  - Una función `start` o `start_link` en función de `supervisor:start/2` o `supervisor:start_link/2`

UNIVERSIDAD SIMÓN BOLÍVAR

`supervisor` – gerente de procesos (pun intended)

Procesos explotados por procesos

- Gestión dinámica de procesos supervisados
  - `supervisor:start_child/2`
  - `supervisor:terminate_child/2`

UNIVERSIDAD SIMÓN BOLÍVAR

`supervisor` – gerente de procesos (pun intended)

Procesos explotados por procesos

- Gestión dinámica de procesos supervisados
  - `supervisor:start_child/2`
  - `supervisor:terminate_child/2`
- *Callback*
  - `start_pool/3` – especificación de hijos.
  - `stop_pool/1` – terminarlos todos.
- El programador escribe y exporta:
  - La función `start_pool/3` en función de `supervisor:start_child/2`.
  - La función `stop_pool/1` en función de `supervisor:stop_child/2`.

UNIVERSIDAD SIMÓN BOLÍVAR

`application` – componente concurrente reusable

No necesariamente una aplicación terminada

- «Paquete» instalable de supervisores y workers.
- «Normales» – inician el árbol de supervisión automáticamente.
- «Librerías» – sólo para ser usadas por otras aplicaciones.
- Visibles en el entorno Erlang

```
> application:which_applications()
```

UNIVERSIDAD SIMÓN BOLÍVAR

Organización por convención

- Jerarquía de directorios predeterminada
  - `src` – código fuente de todos los módulos.
  - `ebin` – ejecutables BEAM y *resource file*.
  - `include` – encabezados Erlang (`.hrl`) si hubiera.
  - `priv` – material colateral estático (imágenes, texto, etc.)

UNIVERSIDAD SIMÓN BOLÍVAR

Organización por convención

- Jerarquía de directorios predeterminada
  - `src` – código fuente de todos los módulos.
  - `ebin` – ejecutables BEAM y *resource file*.
  - `include` – encabezados Erlang (`.hrl`) si hubiera.
  - `priv` – material colateral estático (imágenes, texto, etc.)
- Resource File («app file») – `foo.app` con una tupla Erlang.
  - Descripción – lo que muestra `which_applications/0`.
  - Versión – necesaria para *«code change»*.
  - Lista de módulos necesarios – propios y adicionales.
  - Nombres que registrará.
  - Aplicaciones de las cuales depende.
  - Módulo y *callback* para iniciar la aplicación.

UNIVERSIDAD SIMÓN BOLÍVAR

Organización por convención

- Jerarquía de directorios predeterminada
  - `src` – código fuente de todos los módulos.
  - `ebin` – ejecutables BEAM y *resource file*.
  - `include` – encabezados Erlang (`.hrl`) si hubiera.
  - `priv` – material colateral estático (imágenes, texto, etc.)
- Resource File («app file») – `foo.app` con una tupla Erlang.
  - Descripción – lo que muestra `which_applications/0`.
  - Versión – necesaria para *«code change»*.
  - Lista de módulos necesarios – propios y adicionales.
  - Nombres que registrará.
  - Aplicaciones de las cuales depende.
  - Módulo y *callback* para iniciar la aplicación.
- Gestión de una aplicación
  - `application:start/1`
  - `application:stop/1`
  - *Callbacks* se llaman igual.

UNIVERSIDAD SIMÓN BOLÍVAR

Quiero saber más...

- Learn you some Erlang for Great Good!
- OTP Design Principes User's Guide – en línea y PDF.
- El libro del creador de Erlang – mejor que el de O'Reilly

Joe Armstrong
Programming Erlang: Software for a Concurrent World
The Pragmatic Bookshelf
Raleigh, North Carolina
ISBN-10: 1-9343560-0-X

UNIVERSIDAD SIMÓN BOLÍVAR