Programación Declarativa Avanzada

Curso 2011/201

Juan Rodríguez Hortalá

Hoja 1 de Problemas: Erlang

Referencias básicas para programar

- Documentación del API: http://www.erlang.org/erldoc
- Erlang Reference Manual User's Guide v 5.9: http://www.erlang.org/doc/reference_manual/users_guide.html
- Documentación de la consola Erlang: http://www.erlang.org/doc/man/c.html

Utilizando la consola Erlang

El intérprete de Erlang se inicia ejecutando el comando `erl` desde la consola del sistema operativo, lo que producirá una salida similar a esta:

```
$ erl
Erlang R14B03 (erts-5.8.4) [source] [64-bit] [smp:8:8] [rq:8]
[async-threads:0] [hipe] [kernel-poll:false]

Eshell V5.8.4  (abort with ^G)
1>
```

En el prompt se indica en número de instrucción de consola Erlang que va ejecutar. Se puede introducir cualquier expresión Erlang para que se evalúe, seguida de un punto que funciona como terminador. En el caso de los matching se devolverá la evaluación de la expresión del lado derecho.

```
1> 1 + 2 .
3
2> X = 2 + 4 .
6
3> 6 = X .
6
4> 6 == X .
true
```

Se pueden introducir expresiones complejas que ocupen varias líneas en pantalla, ya el intérprete Erlang entiende que la instrucción no se termina hasta que se alcanza el punto.

```
5> case X of
5> 0 -> no ;
5> 6 -> ok
5> end .
ok
```

Para cargar un módulo `mod` iniciamos el intérprete Erlang desde el mismo directorio en el que se encuentra el archivo `mod.erl` y ejecutamos el comando `c(mod)`, que devolverá `{ok, mod}` en caso de compilación exitosa, o `error` en caso de error de compilación, junto con varios mensajes de error.

```
1> c(mod) .
{ok,mod}
2> c(mod) .
./mod.erl:4: syntax error before:
./mod.erl:2: function f/1 undefined
error
```

En caso de que la compilación tenga éxito se generará un archivo `mod.beam` que es similar a los archivos `.class` de Java. Desde el intérprete Erlang podemos hacer una llamada cualificada (precedida de `modulo:funcion(arg1, …, argn)`) de cualquier módulo cuyo `.beam` esté en la ruta en la que se esté ejecutando el intérprete.

```
3> echo:go() .
Main process received 'hello'
Echo server stopped
stop
```

Después de la primera llamada a una función de un módulo, la consola Erlang autocompletará las llamadas cualificadas a funciones de ese módulo usando el tabulador, al estilo de las consolas UNIX. El autocompletado estará siempre disponible para los módulos estándar precargados como `lists`

```
4> lists:
all/2          any/2          append/1       append/2       concat/1
delete/2       dropwhile/2    duplicate/2    filter/2       flatlength/1
…
```

Otros comandos básicos:

`q()` (quit) termina la ejecución de la consola Erlang (abreviatura de `init:stop()`)
`help()` escribe en pantalla la lista de comandos de consola disponibles
`h()` (history) escribe la lista de comandos introducidos en la sesión de consola
`v(N)` devuelve el valor calculado en la linea `N`
`memory()` escribe en pantalla información acerca del uso de memoria
`cd(Dir), ls(), ls(Dir), pwd()` hacen lo mismo que los comandos bash correspondientes

En cualquier momento se puede interrumpir la ejecución del intérprete Erlang (por ejemplo debido a un cómputo que no termina) pulsando Ctrl-C

```
2>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
c

2>
```

donde si elegimos `a` se cierra el intérprete y con `c` se continúa la ejecución (el resto de opciones muestran información de bajo nivel sobre el estado del intérprete Erlang).

1. Primeros programas

a) Define una función `spanish_date/0` que devuelva un string correspondiente a la fecha actual en formato “dia/mes/año”. Utiliza para ello las funciones `date/0` e `integer_to_list/1` del módulo `erlang`.

b) Define una función `temp_conv/1` que convierta entre temperaturas en grados Fahrenheit y grados Celsius, de forma que `temp_conv({c, GradosCelsius})` devuelva `{f, GradosFahrenheit}`, y `temp_conv({f, GradosFahrenheit})` devuelva `{c, GradosCelsius}` haciendo las conversiones correspondientes.

c) Define una función `fib/1` que calcule el número de Fibonacci correspondiente al entero de entrada implementando directamente la definición recursiva de la sucesión de Fibonacci.

d) Define una función `lin_fib/1` que implemente una versión de coste lineal de `fib/1` basándote en el siguiente algoritmo imperativo:

```
function f(a, b, n)
 { if(n <= 1) return b;
   else return f(b, a+b, n-1);
 }

function fib(n)
 { return f(0, 1, n); }
```

Compara la diferencia de eficiencia entre las dos versiones (se empieza a notar desde el número 35 más o menos).

2. Programación con listas

Escribe un módulo `my_list` donde se exporten funciones que realicen las siguientes operaciones (utiliza las funciones internas al módulo que consideres oportuno).

a) Define una función `min/1` devuelve el elemento más pequeño de la lista que toma de entrada. Defínela mediante recursión y también sin usar recursión y utilizando la función `lists:foldl/3`.

b) Define una función `sum/1` que devuelve la suma de todos los elementos de la lista de números que toma de entrada. Defínela mediante recursión y también sin usar recursión y utilizando la función `lists:foldr/3`.

c) Define una función `delete_all(Elem, List)` que devuelva el resultado de eliminar todas las apariciones de `Elem` en `List`. Defínela mediante recursión, también sin usar recursión y utilizando una combinación de `lists:concat/1` y `lists:map/2`, y por último sin recursión pero utilizando `lists:foldr/3`

d) Define una función `qsort/1` que ordene la lista de entrada usando el algoritmo Quicksort.

e) Define una función `from_to(N, M)` que acepte dos enteros y devuelva la lista `[N, N+1,`

`…, M]` si `N` es menor o igual que `M`, y la lista vacía en otro caso.

f) Define la función factorial sin utilizar recursión, empleando alguna función del módulo `lists` y alguna función de las definidas en apartados anteriores.

3. Entrada-Salida

a) Define una función `filter_even()` que solicite por la consola una lista de números y escriba en la consola una lista con los números pares de la lista leída, y que escriba "`wrong format`" en caso de no haber recibido una lista de números. Por ejemplo:

```
1> es:filter_even() .
Dame una lista de enteros> [2,5,6].
[2,6]
ok
2> es:filter_even() .
Dame una lista de enteros> {0}.
wrong format
ok
```

Sugerencia: estudia las funciones `io:read/1` y `io:fwrite/2` del módulo io, y las funciones `is_list/1` e `is_integer/1` del módulo `erlang`.

b) Define una función `mark_integers()` que solicite por la consola un término compuesto de números, átomos, tuplas y listas y escriba en la consola el resultado de reemplazar en dicho término los números enteros pares por el átomo `even` y los impares por el átomo `odd`, y que escriba "`wrong format`" en caso de no haber leído un término con el formato esperado.

Sugerencia: estudia las funciones de la familia `is_` y las funciones de conversión entre tuplas y listas del módulo `erlang`.

4. Base de datos usando listas

a) Define un módulo `db` que implemente una base de datos que asocia claves a valores utilizando listas de parejas `{Clave, Valor}`, y que ofrezca las funciones:

`new()` devuelve una nueva base de datos

`write(Clave, Valor, Db)` devuelve la base de datos resultado de modificar `Db` para eliminar cualquier asociación que pudiera existir a `Clave`, y añadir una asociación entre `Clave` y `Valor`

`read(Clave, Db)` devuelve el valor asociado a `Clave` en la base de Datos Db, o el átomo not_found si no existe ningún valor asociado

`delete(Clave, Db)` devuelve la base de datos resultado de modificar `Db` para eliminar cualquier asociación que pudiera existir a `Clave`

`store(FileName, Db)` intenta guardar la base de datos `Db` en el archivo de texto correspondiente en la ruta especificada por el string `FileName`, o muestra por pantalla el motivo de un error de escritura, en su caso

r`etrieve(FileName)` intenta leer una base de datos desde el el archivo de texto correspondiente en la ruta especificada por el string `FileName`, o muestra por pantalla el motivo de un error de lectura, en su caso

No se podrá utilizar ninguna función de librería excepto las de las librerías de entrada-salida.

Sugerencia: estudia las funcione `file:open/2`, `io:write/2`, `io:fwrite/3` y `io:read/2`. Cuidado al usar `io:read/2` porque esta función sólo es capaz de parsear a términos erlang las cadenas de caracteres terminadas en punto.

b) Reimplementa el módulo `db` en un nuevo módulo `db_tree` que se base en árboles binarios de búsqueda con parejas en los nodos, en vez de en listas. Los árboles binarios de búsqueda pueden representarse como tuplas `{node, {Key, Value}, Left, Right}` usando el átomo `leaf` para las hojas, o cualquier variante de esta representación.

c) Crea un módulo que implemente un servidor de base de datos de parejas clave-valor utilizando como estado o bien la lista de parejas del apartado a) o el árbol del apartado b).

- Define la función que realiza el bucle como una función local al módulo y crea funciones exportadas para iniciar el servidor, detenerlo, escribir, leer y borrar asociaciones clave-valor.
- Experimenta viendo como varios procesos Erlang acceden al mismo servidor creando varios procesos y llamando a las funciones exportadas por el módulo del servidor.