Fan programming language: Introduction

Chris Grindstaff

Barcamp 2008

What is Fan

- A human-centric neighborhood in Richmond, VA

- "Fan is designed as a **practical programming** language to make it easy and fun to get real work done. It is **not an academic language** to explore bleeding edge theories, but based on solid real world experience."

Why should I care?

- You like learning languages
- You're looking for Java 3000, ng, etc
  - You think Scala is too complicated
- You have an insatiable thirst for knowledge
- You have nothing better to do on a Saturday :-)

Features

- Statically and dynamically typed (with some type inferencing)
- Closures
- Interesting concurrency model
  - Immutability
  - Message passing
  - URI namespace (whiteboard)

Features

- Objects all the way down (no primitives)
- Mixins
  - Similar to Ruby's mixins / Java interfaces with implementation
- Facet support
- Nice set of APIs
- Code organization with Pods
- Tools
  - Interpreter
  - Build tools
- FWT (SWT UI toolkit for Fan)
- Reflection
- Serialization
  - Subset of language
- Benevolent dictator model of development

Cons

- Java/.Net interop is weak at the moment
- No IDE support
- Virtual/override methods

Key concepts

- Pod (namespace and unit of deployment)
  - Classes (single inheritance)
  - Mixins (multiple inheritance / interfaces with implementations)
    - Slots
      - Fields (accessed via methods, no need for getter/setter)
      - Methods
    - Slot names are unique

Simple data types

- “color = $from.color”
- “foo ${tree.get(4)}”
- r”c:\stuff\fan.txt”

Strings with interpolation and “raw” support

- 0xcafe_babe
- 299_792_458
- 6.32d

64 bit ints, floats, decimal

- 500ms
- 12s
- 42min
- 7days

Durations

- `the best report.doc`
- `http://gstaff.org`
- `/some/path/to/me.txt`

URI

Simple data types (cont)

- Str#
- foo::MyType#
- Str#capitalize

Types and slots

- 0..5 // [0,5]
- x…y // [x,y)
- 0..myList.size

Ranges

- [1,2,3] // Int[]
- [6, 7f, 8] // Num[]
- [3, “3”, [3,4]] //Obj[]
- [,] // empty list

List

- [2:”two”, 4:”four”]
- [“a”:[1,2], “b”:4sec]
- [:] //empty map

Map
(keys must be immutable)

Sugar (yumm)

- a + b //a.plus(b)
- a[b] //a.get(b)
- a[b] = foo //a.set(b, foo)
- a[b] //a.slice(b)

(at last count) 24 shortcut methods

- Safe invoke
  - weight = aCar?.door("left")?.handle?.weight
  - short-circuit message sends if any part is null

Typing "Keep it simple"

Static

- Method and field signatures require types
- Local vars, lists, and maps are inferenced

```
Void add(Str first, Str last) {
  person := Person.make(first, last)
  person.age = 8
  people.add(person)
}
```

Dynamic

- The arrow "->" is not compile time checked.
- The dot "." is compile time checked
- At runtime if the message is not understood the trap() method is called. Like missing_method in Ruby

```
This fight(Obj enemy, Int weapon) {
  // Attack the opponent
  damage := rand(strength + weapon)
  echo("You hit for $ damage!")
  enemy -> hit(damage)
  ...
}
```

Closures

- Real closures that capture local variables
- Basic syntax: |A a, B b...->R| { statements }
- Heavy use by List, Map, Thread

```
4.times |Int i| {echo(i)}   //print 0 to 3

add := |Int a, Int b->Int| {return a + b}
add(3,4)  //7

q := [1,2,3,4]
q.findAll |Int m->Bool| {return m % 2 == 0} //[2,4]
```

```
i := 0
f := |->Int| {return ++i}
echo(f())  //prints 1
echo(f())  //prints 2
echo(i)    //prints 2
```

Concurrency

- No shared mutable state between threads
- Messaging passing of immutable state between threads
  - Immutable is a first class concept
  - Each thread has a message queue
- Whiteboard creates a namespace of URIs for threads to share state
  - Thread.sendAsync
  - Thread.sendSync
  - Namespace.create
  - Namespace.put

Concurrency (cont)

- Passing an object between threads
- The object must be either:
  - Immutable – will be passed by reference
  - Serializable – deep copy made and passed

```
svr := Thread("server") |Thread t| {
  t.loop |Obj msg->Obj| {
    echo("reflector received $msg")
    return msg
  }
}
svr.start
```

```
for (i:=0; i<50000; i++) {
  echo("send " + svr.sendAsync(i))
}
```

Links

- Fan site
  - http://www.fandev.org/

Acknowledgements

- Flickr for creative commons images
  - Taberandrew
- Andrew and Brian Fan's benevolent dictators
  - Great docs out of the gate

Backup

Serialization

- Read/write objects to a stream
- Used to pass messages between threads
- Tree based not graph based (ugggh circular refs mean stack overflow)
- Syntax
  - Easy to read
  - Efficient
  - Purely declarative / Fan is a complete superset of serialization format

```
@serializable
class Address {
  Str street; Str city; Str state
}
create one:

address := Address {
  street = "1801 Varsity Drive"
  city = "Raleigh"
  state = "NC”
}
```

```
out.writeObj(address)

results in:
AddressExample_0::Address{
street="1801 Varsity Drive"
city="Raleigh"
state="NC"
}
```

FWT

- Fan widget toolkit built atop SWT
- Makes heavy use of the serialization format

```
Window {
  title = "FWT Demo"
  bounds = Rect {x = 100; y = 100; w = 200; h = 75}
  Button {text = "Hello world"; onAction = |,| {echo("hi")}}
}.open
```

A button instance will be created and the “add” method called on Window