



Stretching Java

No Fluff, Just Stuff

By Bruce A. Tate
J2Life, LLC



- **Introduction**
- **Continuations**
- **JVM alternatives**
- **Metaprogramming**
- **Idioms and Abstractions**
- **Conclusion**



- **Introduction**
- Continuations
- JVM Alternatives
- Metaprogramming
- Idioms and Abstractions
- Conclusion



- Author and speaker
 - 6 books, including
 - Better, faster, lighter Java; Developer Notebooks
 - Founder of Bitter series at Manning
- Independent consultant
 - Focus on design reviews: Lightweight development strategies in Java and Ruby
 - Customers include FedEx, Great West Life, IHS, many others
- Frequent speaker
 - NoFluff series for 4th year; TSS symposium; keynote for JavaGroupen; JavaZone; international JUGs; etc.
- Programmer, consultant, sales specialist for IBM
 - Databases, OO infrastructure, Java
- CTO, director, architect for startups
 - Pervado, IronGrid, Alloquent



- Java has an established community
 - And powerful code base
- Dynamic languages have productivity
 - And important new ideas
- How do you compromise to get the best of both?



- Introduction
- **Continuations**
- JVM Alternatives
- Metaprogramming
- Idioms and Abstractions
- Conclusion



- Web programming is not stateful
- User can disrupt expected flow
 - with back button
 - Or previous windows in browser history
- Applications are made of disjointed requests
- State must be managed manually

->All takes more effort than it should!



- The rest of the program
- A frozen point in time
- A copy of the call stack
 - With local variable values
 - And current point of execution



```
irb(main):018:0> def loop
irb(main):019:1>   for i in 1..4
irb(main):020:2>     puts i
irb(main):021:2>     callcc {|c| return c} if i==2
irb(main):022:2>   end
irb(main):023:1> end
=> nil
irb(main):024:0> c=loop
1
2
=> #<Continuation:0x2ab16f8>
irb(main):025:0> c.call
3
4
=> 1..4
```



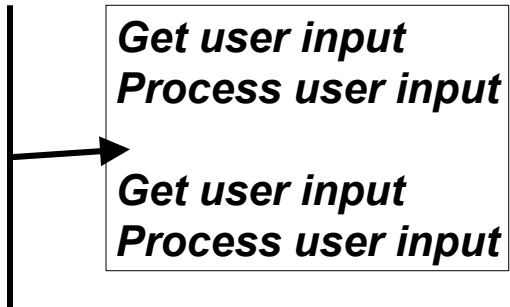
- Use continuations to automate
 - State management
 - Back button
 - Threading support
- Java does not support native continuations
 - Though some are talking about supporting them



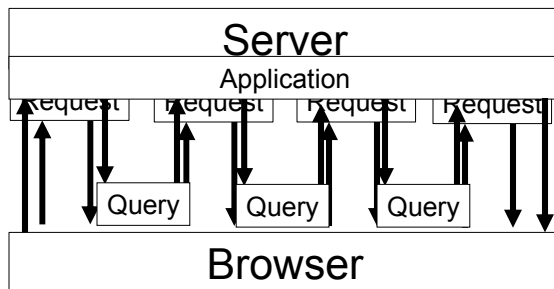
- State gets stored with continuation
- No need to store/restore state
- Very natural programming style

Get user input
 Restore state
 Process user input
 Store state

Restore State
 Get user input
 Process user input
 Store state



- Back button
- Threading
- Inversion of control





- Seaside
 - Developed by Avi Bryant
 - In a language called Squeak
 - A derivative of Smalltalk



```
go
| shipping billing creditCard |
cart _ WASToreCart new.
self isolate:
    [[self fillCart.
     self confirmContentsOfCart]
     whileFalse].

self isolate:
    [shipping _ self getShippingAddress.
     billing _ (self useAsBillingAddress: shipping)
               ifFalse: [self getBillingAddress]
               ifTrue: [shipping].
     creditCard _ self getPaymentInfo.
     self shipTo: shipping billTo: billing payWith:
     creditCard].

self displayConfirmation.
```



- Spring WebFlow
 - Implement a state machine
 - Specify a machine using Web Flow rules
 - Nodes are pages or methods
 - Transition on events (failure, success, etc)
 - Store a state in a dictionary like a continuation
- Rife
 - Implements native Java continuations
 - Get a stack trace + current line
 - Compute offset to current line of code
 - Do byte code injection to skip to current line of code
 - Force all instance variables to implement Clonable
- Cocoon 2
 - Implement Java on Rhino (JavaScript engines)
 - Use custom code to implement continuations in Rhino



```
public void processElement() {
    Template template = getHtmlTemplate("game");
    int answer = 0; int guesses = 0;
    answer = RandomNumbers.nextInt(101);
    while (mGuess != answer) {
        print(template);
        pause();           // Capture continuation!
                           // code to validate
    number of guesses
        guesses++;
        if (answer < mGuess) {
            template.setBlock("indication",
"lower");
        } else if (answer > mGuess) {
            template.setBlock("indication",
"higher");
        }
    }
    // code to load success template
}
```



```

<body>
  <p>Guess a number from 0 to 100</p>
  <!--V 'warning'--><!--/V-->
  <!--B 'invalid'-->
  <p><font color="#990000">Invalid guess.</font></p>
  <!--/B-->
  <p><i><!--V 'indication'--><!--/V--></i></p>
  <!--B 'lower'-->The answer is lower.<!--/B-->
  <!--B 'higher'-->The answer is higher.<!--/B-->
  <form
    action="[/V 'SUBMISSION:FORM:perform_guess']/"
    method="post">
    <!--V 'SUBMISSION:PARAMS:perform_guess'-->
    <input type="text"
      name="guess"
      value="[/V 'PARAM:guess'][/V]" /><br />
    <input type="submit" value="Guess" /><br />
  </form>
  <script language="javascript">
    document.forms[0].guess.focus();
  </script>
</body>

```



```

<webflow id="myFlow" start-state="displayForm">

  <view-state id="displayForm" view="myForm">
    <transition on="submit" to="processSubmit">
      <action bean="formAction" method="bindAndValidate"/>
    </transition>
  </view-state>

  <action-state id="processSubmit">
    <action bean="formAction"/>
    <transition on="success" to="finish"/>
  </action-state>

  <end-state id="finish" view="myFormSuccess"/>

</webflow>

```



- Continuations in Rhino or with byte code enhancement
 - Much more natural programming style
 - Any complex Java logic
 - Very few requirements (e.g. Clonable in Rife)
- State machine
 - Persist state (long-duration workflows)
 - Good integration with existing MVC engines
 - XML is good for tooling



- Introduction
- Continuations
- **JVM Alternatives**
- Metaprogramming
- Idioms and Abstractions
- Conclusion



- Dated language
- Lots of safety
 - Checked exceptions
 - Static typing
- Not truly OO
- Lacks higher abstractions
 - Continuations
 - Closures
 - Parallel assignments
- Difficult Metaprogramming



- Dynamic lanugages
- Full OO
- More, and more powerful abstractions



- We have significant Java investments
 - People
 - Code
 - Infrastructure
- And someone stuck their neck out to pick Java



- Sun's traditional view:
 - JAVA == Language!
 - Enhancements go into language, not JVM
 - Generics
 - Autoboxing
 - A few minor exceptions
- An alternative view: Java == JVM
 - This has been the Microsoft view
 - As Java wanes, this will become Sun's view too



- Why wouldn't you?
 - Politics
 - Want consistent strategy and code
- Why would you?
 - The right tool for the job
 - Easier sell to management
- Prediction: JVM will become much more important for dynamic languages



- Coarse-grained integration
 - Web services, XML + remoting, database, etc.
 - Architectural layers are in appropriate language
- Scripting for special cases
 - User customization
 - Test cases
 - Ant builds!
 - Operating system scripts
- Fine-grained integration
 - Call or even subclass Java objects directly
 - Requires excellent



- Groovy (natively on JVM)
- Ruby (JRuby)
- Python (Jython)
- Scheme (several)
- JavaScript (Rhino)
- NetRexx
- Some experimental langs (Nice, Pnuts)



- Groovy
 - Excellent marketing
 - Good integration with Java, in theory
- Negatives
 - Lexical scoping broken
 - Very buggy, unprofessional
 - Major bugs in late betas (0/2 yields error in beta 10; closures nearly unusable in beta 9)
 - Steve Yegge (Wyvern game inventor)



- “Groovy itself is ugly. It has no sense of aesthetics, no purity of vision. It’s a Perl-style grab bag of features thrown in from other languages.
- ...they didn’t start off with a parser generator—they did some hacked up hand job...didn’t they go to school?
- ...reeks of amateurism: the slipshod quality, the documentation that focuses on marketing rather than features, the lack of focus and attention to detail. It’s not professional quality work.
- ...is unrelentingly buggy, and slow as molasses in January...
- ...and generally uses the Principle of Most Surprise...
- That’s very likely the last I’ll look at Groovy, because even if they miraculously fix their horrible problems, it’ll still be an ugly language. That, and I’m upset that they put me through this [comparison]



- Python for JVM
- Strengths
 - Very strong implementation
 - Has the critical Python features
 - Subclass Java with Python directly
- Weaknesses
 - Back level (2.1 vs. 2.4)
 - May soon be remedied; new grant money to bring it up



- Strengths
 - Strong implementation
 - Great language
 - Test cases getting closer
 - You can view the base Ruby test cases
- Weaknesses
 - Can't subclass Java classes
 - Several stops and starts; jury is still out
 - Won't run Rails yet
 - Still very young; not really usable yet



- Strengths
 - Stable
 - Great tools
 - Performance
- Weaknesses
 - JavaScript not a great language



- End user scripting
 - Like JavaScript in browser
 - Many examples
- Politics: Introducing a dynamic language
- Build/deploy tools
 - JUnit tests in scripting languages
 - Test fixtures (set up databases, etc)
 - It's a matter of time until Ant uses a scripting language
 - Deploy scripts using JVM language would be useful



- Introduction
- Continuations
- JVM Alternatives
- **Metaprogramming**
- Idioms and Abstractions
- Conclusion



- Java is full of repetition
 - Crosscutting concerns
 - Idioms
- Other logic clutters main line logic
- Many examples



- Metaprogramming
 - Writing programs that write programs
- Dynamic languages can directly modify or add to a class
- Dynamic typing allows flexible usage of concepts with less effort
 - Like getters and setters



- Ruby getters/setters

```
Class Person
  attr_accessor :name
  attr_accessor :email
end
```

```
def name
  @name
end

def name=(value)
  @name=value
end
```

- attr is a method
 - Works like Metaprogramming tag
 - Creates getters, setters and property



- Og is OR mapper
- Replaces attr Metaprogramming tags
 - With property
- Example:

```
class Post
  property :title, String
  property :body, String
  property :author, String
  property :create_time, Time
  property :hits, Fixnum
end
```



- Persistence for Rails
 - Highly dynamic
- Base class. At run time:
 - Queries database table design
 - Adds attributes to object
 - Adds specific database methods to object
- Extends active record
 - Include inheritance
 - Relationships via macros



```
class Trail < ActiveRecord::Base  
  belongs_to :location
```

```
@@columns
```

← Metadata

```
@name  
@description  
@difficulty  
@location
```

← Attributes

```
save  
find  
find_first  
find_all
```

← Behavior

```
end
```



- Very common, and increasingly important
 - Hibernate
 - Spring
 - JBoss
 - EJB
 - JDO



- Code generation (EJB, velocity)
- Byte code injection (JDO, RIFE)
 - Code generation after run time
- Proxies
 - Put an object in front of another
 - With the same interface
 - Clients call the new object
 - Proxy handles metaprogramming extensions
- Aspect oriented programming (AspectJ, JBoss, Spring, others)
 - Separate mainline code from crosscutting concerns
 - Define points of attachment
 - Attach crosscutting concerns to mainline code with configuration



- It's a natural progression
 - JavaDoc
 - Marker interfaces
 - Code
- It uses other transparency techniques
 - Byte code injection
 - Reflection



- Use AOP to introduce mix-ins
- Dynamically improve methods
 - Introduce capabilities
- Refactor crosscutting concerns
 - Point cuts (no native support in Ruby/Python/Groovy)
- AOP Refactored: Ramnivas Ladad



- Introduction
- Continuations
- JVM Alternatives
- Metaprogramming
- ***Idioms and Abstractions***
- Conclusion



- Ruby supports code blocks
 - For resource management
 - For iteration of blocks
- Example
- Brief demo

```
result=db.query("select * from words")
result.each {|row| puts row}
```



- Spring handles the details:
 - Executes SQL (CRUD + stored procs)
 - Catches and maps exceptions
 - Manages resources
- Developer implements a JDBC template
 - Passing in SQL
 - And a callback handler
- Result is much less code



```
JdbcTemplate template = new JdbcTemplate(dataSource);
final List names = new LinkedList();

template.query("SELECT USER.NAME FROM USER",
    new RowCallbackHandler() {
        public void processRow(ResultSet rs)
            throws SQLException {
            names.add(rs.getString(1));}});
```



- Naming matters
 - Take advantage of consistency
- Examples
 - Views get called automatically
 - Based on name of controller, method
 - id, table_id for keys, foreign keys
 - name for the name field
- Ruby has other advantages
 - Named and default parameters



- Few and poor
- AOP uses consistent naming for point cuts
 - Regular expression matches
- Annotations have promise
 - Mark defaults and name parameters
- XML formats should specify defaults
- Wrapper classes should specify defaults



- Introduction
- Continuations
- JVM Alternatives
- Metaprogramming
- Idioms and Abstractions
- **Conclusion**



- You can extend Java by borrowing from dynamic languages
- Continuation servers
- Scripting engines
- Metaprogramming
- Inversion of control (closures)



- Ruby
 - Programming Ruby by Dave Thomas
 - Rails
- Continuations
 - Seaside
 - Rife examples (rifers.org)
 - Cocoon
 - Workflow
- AOP
- springframework.com
- JVM Language Comparisons, and links
 - opal.cabochon.com/~stevey/sokoban



- Evaluations!
- J2Life, LLC
 - bruce.tate@j2life.com
 - Design reviews and architecture
 - Persistence and performance consulting
 - Project jump starts
 - Process and product consulting