Why
Java Sucks & C# Rocks

ZhaoJie @ SNDA
April, 2010

About me

- Name: 赵劼（老赵，Jeffrey Zhao）
- Blog: http://blog.zhaojie.me/
- Twitter: @jeffz_cn
- Email: jeffz@live.com

Experiences

- Pascal, VB 5 & 6, Delphi before 2002
- Java programmer from 2002 to 2004
- C# programmer since 2004

I'm going to talk about ...

- Languages features
- Programming paradigms
- Usage patterns
- Why we should throw Java language away

I'm NOT going to make you ...

- Believe CLR is better than JVM
- Believe .NET is better than Java platform
- Get rid of Java language right away
- Use C# instead of Java language
- Use .NET instead of Java platform

Timeline

- 2002 - Java 1.4 & C# 1.0
- 2004 - Java 5.0 & C# 2.0
- 2006 - Java 6
- 2008 - C# 3.0
- 2010 - Java 7 & C# 4

So Java programmers, hold your breath ...

Let’s start from the very beginning ...

Is Java a pure object-oriented language?

No.

Primitive types are not objects.

In Java you can NOT ...

```
ArrayList list = new ArrayList();
list.add(5); // cannot compile
int i = (int)list.get(0); // cannot compile

int hash = 3.hashCode(); // cannot compile
```

You have to ...

```
ArrayList list = new ArrayList();
Integer five = Integer.valueOf(5);
list.add(five);
int i = ((Integer)list.get(0)).intValue();

Integer three = Integer.valueOf(3);
int hash = three.hashCode();
```

In C# ...

- No primitive types
- All types are subtypes of Object.
- Just value types and reference types
- Programmers can build custom value types
- Value types can be assigned to Object variables without explicit casting.

So you can always ...

```
ArrayList list = new ArrayList();
list.Add(5);
int i = (int)list[0];

int hash = 3.GetHashCode();
```

Thank God!

Here comes Java 5.0!

Finally you can ...

```
ArrayList list = new ArrayList();
list.add(5); // auto boxing

// auto unboxing
int i = (Integer)list.get(0);

// you still can’t do this!
int hash = 3.hashCode();
```

Java 5.0 also brings ...

- Annotation - allows language constructs to be tagged with additional data
- Enumerations - the “enum” keyword creates a typesafe, ordered list of values
- Varargs - make passing an array as parameter easier then before
- Enhanced “for” loop - simplified iteration

But most of these features were already provided by C# 1.0 ...

... so who is the copy cat?

OK, let's forget it and look deep inside ...

Annotations in Java are interfaces.

Attributes in C# are classes.

Differences?

Let’s start coding!

Annotation's shortages

- Strange convention
- Anemic object
- Hard to unit test

Well, Java 5.0 also brings Generics, but ...

Type erasure: type information will be lost after compiling!

What's the output?

```
List<String> a = new ArrayList<String>();
List<Integer> b = new ArrayList<Integer>();

bool sameType = a.getClass() == b.getClass();
System.out.println(sameType);
```

And you can’t do this

```
public class MyClass<E> {
    public static void myMethod(Object item) {
        if (item instanceof E) { // Compiler error
            ...
        }
        E item2 = new E(); // Compiler error
        E[] iArray = new E[10]; // Compiler error
    }
}
```

And “use-site variance”

is strange and stupid.

Well, maybe I'm biased.

I can give you a separate talk about it.

Let keep moving.

The same time Java released 5.0 ...

C# 2.0 comes with

- Better generics (comparing to Java 5.0)
- Anonymous methods (hello, closure!)
- “yield” keyword - iterator creation can’t be more easier

Why closure matters?

And “yield” is only about iterator?

Let’s see some samples.

So closure and yield are ...

- Increasing productivity dramatically
- Coming with new programming patterns
- Important features for async / parallel / reactive programming
- The primitives for Fibers - lightweight computation units

Two years after C# 2.0,

Java 6 released with ...

Nothing!

Nothing!!

Nothing!!!

Give me a break and leave me alone ...

OK, I admit it ...

Java 6 is cool because of ...

- pluggable annotations (JSR 269): customized compile-time processing (Project Lombok)
- dramatic JVM improvements: compiler performnace, start-up time, GC, JIT...
- scripting language support (JSR 223): Rhino JavaScript for Java included

But we're talking about language, so ...

Let's move on to C# 3.0, which brings us ...

LINQ

(Language Integrated Query)

Only LINQ?

Yes! The great LINQ!

LINQ is made up of ...

- Extension method - safely add methods to types without breaking anything
- Lambda expression - make C# a better functional language
- Anonymous type - compile time auto generated types
- Expression tree - language-level code in the form of data

Now tell me, which one do you prefer?

BBCode to HTML

```
// Java
Util.stripWhites(
    Util.stripXss(
        Util.bbToHtml(bbCode)))

// C#
bbCode.BBToHtml().StripXss().StripWhites()
```

Sorting an array

```
// C#
users.Sort((u1, u2) => u1.Age - u2.Age);

// Java
Arrays.sort(
    users,
    new Comparator<User>() {
        public int compare(User u1, User u2) {
            return u1.Age - u2.Age;
        }
    });
```

JSON Output (C#)

```
List<User> userList = ...;
var json = new {
    errorCode = 0,
    message = "OK",
    users = userList.Select(u => new {
        name = u.FirstName + " " + u.LastName,
        age = (DateTime.Now - u.BirthData).Year
    }),
};
```

JSON Output (Java)

```
JSONObject obj = new JSONObject();
obj.put("errorCode", 0);
obj.put("message", "OK");

JSONArray userArray = new JSONArray();
for (User u: userList) {
    JSONObject objUser = new JSONObject();
    objUser.put("name", /* get full name */);
    objUser.put("age", /* calculate age */);
    userArray.add(objUser);
}

obj.put("users", userArray);
```

Is C# just full of Syntactic Sugar?

I don’t think so.

It really changed my mind

- Real-world Functional Programming
- Write declarative code (anti-for)
- Make code / API more and more elegant

Please write a program to index a list of keywords (like books).

Explanation

- Input: List<String>
- Output: Map<Character, List<String>>
- The key of map is ‘a’ to ‘z’
- Each list in the map are sorted

Let's see the Java code first ...

```
List<String> keywords = ...;
Map<Character, List<String>> result = new HashMap<>();

for (String k: keywords) {
    char firstChar = k.charAt(0);

    if (!result.containsKey(firstChar)) {
        result.put(firstChar, new ArrayList<String>());
    }

    result.get(firstChar).add(k);
}

for (List<String> list: result.values()) {
    Collections.sort(list);
}
```

```
List<String> keywords = ...;
Map<Character, List<String>> result = new HashMap<>();

for (String k: keywords) {
    char firstChar = k.charAt(0);

    if (!result.containsKey(firstChar)) {
        result.put(firstChar, new ArrayList<String>());
    }

    result.get(firstChar).add(k);
}

for (List<String> list: result.values()) {
    Collections.sort(list);
}
```

Imperative code

"How" to do

Quite easy, ah?

Guess how does C# code look like?

```
List<string> keywords = ...;
var result = keywords
    .GroupBy(k => k[0])
    .ToDictionary(
        g => g.Key,
        g => g.OrderBy(k => k).ToList());
```

```
List<string> keywords = ...;
var result = keywords
    .GroupBy(k => k[0])
    .ToDictionary(
        g => g.Key,
        g => g.OrderBy(k => k).ToList());
```

Declarative code

"What" to do

Amazing, isn’t it?

What about the Java community?

Well, the Functional
Java project is out!
But ...

... do you really want to write code like this?

```
keywords
   .groupBy(
     new F<String, Character> {
       public Character f(String s) { return s.charAt(0); }
     })
   .toMap(
     new F<Grouping<Character, String>, Character> {
       public Character f(Grouping<Char, String> g) {
         return g.getKey();
       }
     },
     new F<Grouping<Character, String>, List<String>> {
       public List<String> f(Grouping<Character, String> g) {
         return g
           .orderBy(
             new F<String, String> {
               public String f(String s) { return s; }
             })
           .toList();
       }
     });
```

It’s really a pain without lambda expressions.

Fortunately, Java 7 will provide a sort of lambda syntax.

```
keywords
    .groupBy({ String k => k.charAt(0) })
    .toMap(
        { Grouping<Character, String> g => g.getKey() },
        { Grouping<Character, String> g =>
          g.orderBy({ String c => c }).toList() });
```

Much better,

but still noisy because of lacking ...

Type Inference

C# is an FPL because of

- First class functions (as .NET delegates)
- Elegant lambda expression syntax
- Enough type inference for most cases
- Full set of functional primitives in BCL
- Great extensibility (by extension method)

Fixed-point combinator

```
static Func<T, TResult> Fix<T, TResult>(
    Func<Func<T, TResult>,
    Func<T, TResult>> f)
{
    return x => f(Fix(f))(x);
}

var fib = Fix<int, int>(f => x =>
    x <= 1 ? 1 : f(x - 1) + f(x - 2));
```

Is it really useful in real world programming?

Yes, of course.

I can’t live without the functional features now.

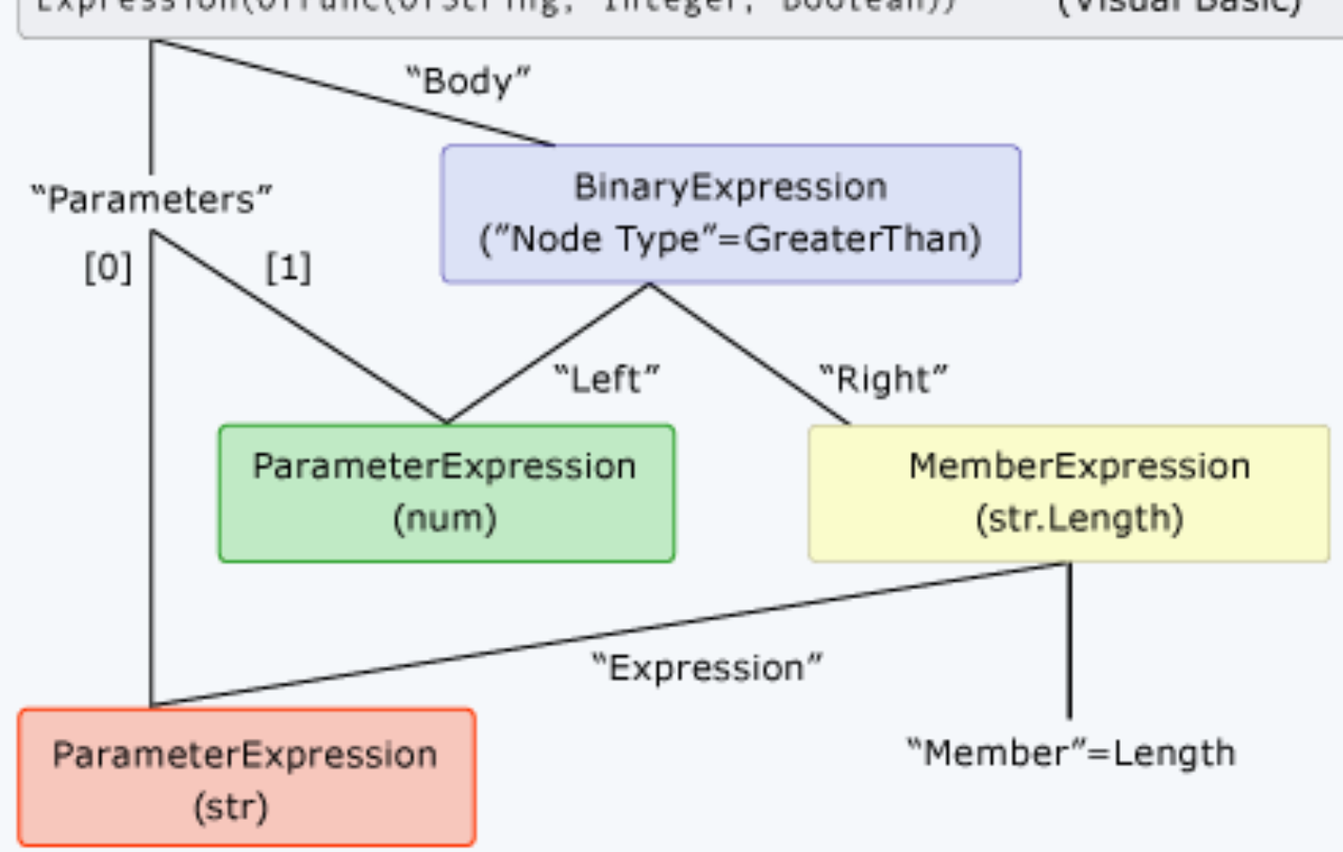
Let me prove it to you.

Expression Trees

```
(str,num)=>    num>  str.Length    (C#)
Function(str,num)  num>  str.Length    (Visual Basic)
```

The Expression Tree:

```
Expression<Func<string, int, bool>>                (C#)
Expression(OfFunc(OfString, Integer, Boolean))     (Visual Basic)
```



Code as Expressions

- Serializable code
- Consistent coding style
- Strong typing
- Compile-time verification
- Generate dynamic method easily

Many projects use
Expressions.
Let me show you sth.

Java 7 is coming

- Automatic Resource Management
- Collection Literals
- Improved Type Inference for Generics Instance Creation
- Language support for JSR 292
- Simplified Varargs Method Invocation
- Support for Strings in Switch Statements

Most of them are provided by C# now

What's new in C# 4

- Default parameters
- Naming parameters
- Language build-in dynamic support
- Convariance / Contravariance

Taste a bit

Ruby-like Markup Builder

```
dynamic b = new XmlMarkupBuilder();
var persons = new [] {
    new Person("Tom", 10),
    new Person("Jerry", 8)
};
XElement xml =
    b.persons(
        from p in persons
        select b.person(p.Name, age: p.Age));
```

Ruby-like Markup Builder

```
dynamic b = new XmlMarkupBuilder();
var persons = new [] {
    new Person("Tom", 10),
    new Person("Jerry", 8)
};
XElement xml =
    b.persons(
        from p in persons
        select b.person(p.Name, age: p.Age));
```

```
<persons>
  <person age=”10”>Tom</person>
  <person age=”8”>Jerry</person>
</persons>
```

Convariance & Contravariance

- Improving the generics type system
- More powerful and safer than Java's use-site variance
- Really a brain fucker

Summary

- (maybe I should put a table here...)

C# always make me happy when coding ...

... and it brings us lots of useful code patterns

But Java keeps me away from the great JVM

Java is popular since it's ...

- Fast - designed for the JVM, as C is designed for the machine
- Stable - it's always one and only one way to do things
- Easy - most of the features are quite easy to learn

But what's wrong with Java?

Java is too noisy

- The only way to work is not simple
- Write more and do less
- Little room for smart people.

Java is weird ...

```
int a = 1000, b = 1000;
System.out.println(a == b); // true
```

```
Integer c = 1000, d = 1000;
System.out.println(c == d); // false

Integer e = 100, f = 100;
System.out.println(e == f); // true
```

… and becoming even more

```
// C#
dynamic o = ...;
int i = o.SomeMethod(true, “hello”);
```

```
// Java
Object o = ...;
Object[] args = new Object[] { true, “hello” };
InvokeDynamic.<int>SomeMethod(o, args);
```

If JVM is machine ...

- Java byte code is the machine code
- Java language is the ASM.

Would you like to program with ASM?

What should we do?

Let's use C# instead!

Just kidding

We can't use C# since ...

- JVM is powerful
- Java libraries are great
- Platform migration is a nightmare

So choose another language!

Languages on JVM

- JRuby
- Jython
- Groovy
- Clojure
- Scala

All these languages are

- More powerful
- Run on JVM
- Interop with Java seamlessly
- Mature enough
- Successful in real world

Scala makes me write Java program again.

Why Scala?

- a statically-compiled language
- a pure object-oriented languages
- clean & powerful generic support (including convariance & contravariance)
- enough functional features
- actor-based programming model
- really scalable

What do you think?

Contact me via

- Email: jeffz@live.com
- Twitter: @jeffz_cn
- Blog: http://blog.zhaojie.me/

Show me your idea ...

... and prove it to me

Future Talks

- Why F# matters
- Why Scala matters (TBD.)
- Generics in Java, Scala, C# & F#
- The beauty of Parallel Library
- Real-world reactive programming

Thanks