


TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

AOP: Myths and Realities


Ramnivas Laddad
Author, AspectJ in Action
ramnivas@aspectivity.com



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Speaker Introduction


- **Author of books and articles**
 - AspectJ in Action: Practical Aspect-Oriented Programming (Manning, 2003)
- **Speaker at many professional conferences**
 - JavaOne, No Fluff Just Stuff, Software Development, EclipseCon, O'Reilly OSCON etc.
- **Active involvement in AspectJ since its early form**
- **Over a decade of industry experience**
 - Java, J2EE, AspectJ, UML, networking, C++, and XML



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Outline


- **Technological myths**
- **Cultural myths**
- **Adoption myths**



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Outline

- **Technological myths**
- **Cultural myths**
- **Adoption myths**




TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth

AOP is good for tracing, but nothing else

Reality


Tracing is just a "hello world" example



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Tracing: A Poster Child of AOP


- **Crosscutting nature of problem is quickly apparent**
- **Selecting trace points is easy**
 - All public methods
`execution(public * *.*(..))`
 - All exception handlers
`handler(SQLException)`
 - All calls to certain kinds of methods
`call(public * Remote+.*(..) throws RemoteException+)`



TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Crosscutting Concerns Abound **Only a little practice with AOP will show you that it is as universally applicable as OOP**


- **System wide crosscutting**
 - Security
 - Transaction management
 - Thread safety
- **Implementation-specific crosscutting**
 - Business rules
- **Micro, class- or package-level crosscutting**
 - Class-level exception handling
 - Class-level resource life-cycle management



TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Myth
 AOP does not solve any new problems


Reality
 Correct; AOP isn't about solving new problems, it is about solving problems the right way

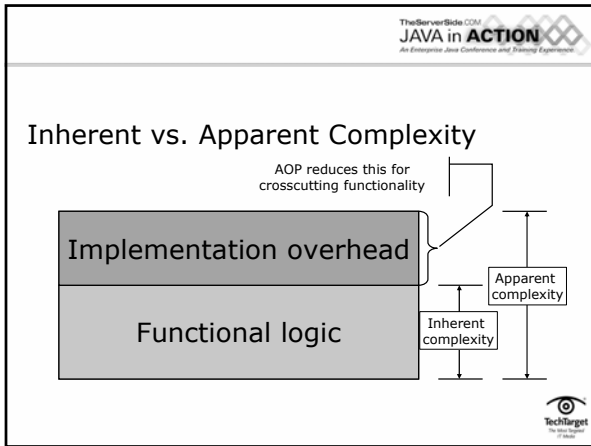


TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

AOP Isn't a New Computational Theory

- **Better methodologies and languages help solve problems in a pragmatic way**
- **AOP merely helps implementation of crosscutting concerns while keeping us sane**
- **AOP helps reduce gap between inherent complexity and apparent complexity**






This slide contains two text boxes on a grid background. The top box is titled 'Myth' and contains the text 'Debugging with aspects is a hard task'. The bottom box is titled 'Reality' and contains the text 'Debugging with aspects is as easy as debugging classes'. The top right corner features the 'TheServerSide.COM JAVA in ACTION' logo, and the bottom right corner features the 'TechTarget' logo.

The slide has a title 'Debugging Needs Right Tools: With AOP or OOP' and a single bullet point: '• Time for a demo...'. The top right corner features the 'TheServerSide.COM JAVA in ACTION' logo, and the bottom right corner features the 'TechTarget' logo.

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
Application frameworks obviate AOP


Reality
Application frameworks provide a substitute for AOP in some cases and leave you hanging in the remaining ones



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Frameworks Implementing Crosscutting Concerns: Approach


- **Target domain-specific problems and provide specific solution to address them**
- **For example, EJB helps modularizing:**
 - Distributed objects
 - Persistence
 - Transaction management
 - Authentication and authorization
 - Concurrency control



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Frameworks Implementing Crosscutting Concerns: Problems


- **Lacks a fundamentally systematic approach**
 - Requires learning new methods for each solution
 - Concerns other than those directly addressed by solution are harder to implement
 - In EJB, how do you implement business rules? data driven authorization?
 - Forces trade-offs
- **What if container services do not meet your needs or get in your way?**
- **What if you don't have a framework?**



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Frameworks Implementing Crosscutting Concerns: Fundamental Issue


- **Application concern space is N-dimensional**
- **Framework solution space is n-dimensional ($n < N$), and dimensions are fixed**
 - Mismatch
 - Ad-hoc support for unsupported dimension
- **AOP can augment frameworks to implement missing dimensions**



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Application Frameworks: The AOP Way


- **Provide a generic support for crosscutting concerns**
 - Easy to extend
 - Easy to substitute with custom solutions
- **Provides pre-specified aspects targeting common domain-specific concerns**
- **Good examples:**
 - Spring
 - JBoss/AOP



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
Many design patterns obviate AOP


Reality
Design patterns often exhibit crosscutting nature and benefit from AOP



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

OO Design Patterns: Strengths and weaknesses


- **Strengths**
 - Well understood
 - Provide good decoupling within limits of OOP
- **Weaknesses**
 - Reusable concepts, but copy-paste implementation
 - Pattern implementation tangled with application code
 - Pattern implementation scattered throughout system
 - Pattern density reduces comprehensibility



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Chain of Responsibility


- **General idea: Add a chain of handlers around main processing**
- **Strengths:**
 - Easy to add coarse-grained processing
- **Weaknesses:**
 - Works when there is already infrastructure in place
 - No servlet filters before 2.3
 - Should infrastructure always be in place?
 - Over-design vs. under-design dilemma
 - Overexposes context
 - The whole request and response object



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Chain of Responsibility: The Players

- **Handler interface**
- **Handler implementation**
- **Chain processing logic**
- **Chain configuration logic**



Chain of Responsibility: Handler Interface

```
public interface Filter {  
    public void init(FilterConfig filterConfig);  
    public void destroy();  
    public void doFilter(ServletRequest request,  
                        ServletResponse response,  
                        FilterChain chain);  
}
```



Chain of Responsibility: Handler Implementation

```
public class PerformanceMonitorFilter implements Filter {  
    public void doFilter(ServletRequest req, ServletResponse res,  
                        FilterChain chain)  
        throws IOException, ServletException {  
        long startTime = System.nanoTime();  
        chain.doFilter(req, res);  
        long endTime = System.nanoTime();  
  
        monitorAgent.record(((HttpServletRequest) req).getRequestURI(),  
                           endTime - startTime);  
    }  
  
    // empty implementations for init() and destroy()  
}
```



Chain of Responsibility: Handler Processing

```
public class FilterChainImpl implements FilterChain {  
    private int currentPos, totalFilters;  
    private Filter[] filters;  
    private Servlet servlet;  
    ...  
    public void doFilter(ServletRequest request, ServletResponse response)  
        throws IOException, ServletException {  
        if(currentPos < totalFilters) {  
            Filter currentFilter = filters[currentPos++];  
            currentFilter.doFilter(request, response, this);  
        }  
        servlet.doService(request, response);  
    }  
}
```



Chain of Responsibility: Handler Configuration

```
<web-app>
...
<filter>
  <filter-name>monitor</filter-name>
  <filter-class>
    com.aspectivity.servlet.filter.PerformanceMonitorFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>monitor</filter-name>
  <url-pattern>/*.do</url-pattern>
</filter-mapping>
...
</web-app>
```



Chain of Responsibility: Aspect Implementation

```
public aspect ServletPerformanceMonitor {
  Object around(HttpServletRequest request)
    : execution(* HtpRequest.do*(..)) && args(request,..) {
    long startTime = System.nanoTime();
    Object retValue = proceed();
    long endTime = System.nanoTime();
    monitorAgent.record(request.getRequestURI(), endTime - startTime);
    return retValue;
  }
}
```



Chain of Responsibility: Merits of the AOP solution

- **Same basic implementation reusable across**
 - Servlets
 - Web services
 - EJBs
 - Your business logic
- **No anticipating of potential needs**
 - Add aspects only when you need them



Observer: The OO Way

- **General idea: Decouple observers from subjects, collaboration through notifications**
- **Strengths:**
 - Avoids direct coupling to observers
 - Modularizes observation logic
- **Weaknesses:**
 - Duplicated subject's infrastructure
 - Observer management and notification
 - Fixed notifications regardless of observers needs



Observer: The AO Way

- **General idea:**
 - Reusable protocol aspect
 - System-specific customization aspect
- **No need to anticipate events**
- **Flexibility in establishing the relationship**
- **Flexibility in specifying the observed events**



Dynamic Proxy: The OO Way

- **General idea: Proxy the target object and add crosscutting behavior to the invocation objects**
- **Strengths:**
 - Crosscutting behavior modularized
- **Weaknesses:**
 - Forces complex reflection-based programming model
 - Requires creating objects through a factory
 - Otherwise, need to replace all 'new' for the proxy objects (a crosscutting concern)
 - Creates object identity problem
 - Weakens static type checking



Dynamic Proxy: The AO Way

- **General idea: An aspect with an advice to add crosscutting behavior**
- **Simple programming model**
 - The same basic AO machinery used
- **No modification to object creation logic**
 - No forced use of factory
- **No replacement of original objects**
 - No object identity problems
- **No weakening of static typing**



Thoughts on Use of Design Patterns

- **AOP transforms reusable concepts into reusable implementation**
- **Not all design patterns can be improved by AOP**
- **Use design patterns, if you are not using AOP**
 - Then consider refactoring design patterns using AOP
- **Even AOP has some design patterns of its own**



Myth

AOP makes evolution hard to manage

Reality

AOP makes evolution easy to manage



Evolution with AOP

- Allows easily integrating new concerns
- Allows easily modifying existing concerns



Concurrency Control: Conventional

```
public class Account {
    private ReadWriteLock _lock = new ReentrantWriterPreferenceReadWriteLock();

    public void credit(float amount) {
        try {
            _lock.writeLock().acquire();
            ... business logic
        } catch (InterruptedException ex) {
            throw new InterruptedException(ex);
        } finally {
            _lock.writeLock().release();
        }
    }
}
```



Concurrency Control: Conventional

```
public float getBalance() {
    try {
        _lock.readLock().acquire();
        ... business logic
    } catch (InterruptedException ex) {
        throw new InterruptedException(ex);
    } finally {
        _lock.readLock().release();
    }
}

... debit(), setBalance() etc. with identical concurrency control
}
```



Concurrency Control: AOP

```

public class Account {
    public void credit(float amount) {
        ... business logic
    }

    public float getBalance() {
        ... business logic
    }
    ... debit(), setBalance()

    private static aspect ConcurrencyControl extends ReadWriteSynchronization perTypeWithin(Account) {
        public pointcut readOperations() : execution(* Account.get*(..)) || execution(* Account.toString(..));
        public pointcut writeOperations() : execution(* Account.*(..)) && !readOperations();
    }
}

```



Evolution: Use Simple Synchronization

```

private static aspect ConcurrencyControl
    extends SimpleSynchronization perTypeWithin(Account) {
    ... Unchanged! ...
}

```



Evolution: Use Java 5 Read-Write Locks

- **Simply change** `ReadWriteLockSynchronization`
 - No changes required to `Account.java`
- **Or, change the base aspect**

```

private static aspect ConcurrencyControl
    extends Java5ReadWriteSynchronization perTypeWithin(Account) {
    ... Unchanged! ...
}


```



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Tracking Evolution


- **Crosscutting diff tool demo...**



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Outline


- **Technological myths**
- **Cultural myths**
- **Adoption myths**



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
Aspects make program flow hard to understand

Reality
AOP stretches program flow transparency in a new way



Understanding Program Flow: Wants vs. Needs

- **Do you need to know everything happening in program flow?**
 - All at the same time?
 - Even when you aren't really concerned about what it actually means?
- **Tools help, if you insist**



Myth

AOP is an overkill for what can be achieved with well designed interfaces

Reality

Well designed interfaces are all good, but they are no substitute for AOP



Fallacies of Swapping Through Interfaces

- **Jakarta Commons Logging**
 - Abstracts logging: Log4j, Standard logging
 - When log4j was first released, how many of us wrote a wrapper?
 - First version of log4j: 2000
 - First version of commons logging: 2003
- **Where are the wrappers for the concurrency APIs?**
- **How do you respond to a new API?**
 - Let's use it...?
 - or—
 - Let's write an abstract wrapper to meet future needs...?



Well-designed Interfaces

- **Requirements**
 - Loose enough to allow wide variety of implementations
 - Tight enough to allow uniform treatment of all implementations
 - Least common denominator (LCD) problem!
- **Anticipating future requirement is tough/impossible**
 - Over-design vs. under-design



Myth

AOP implementations don't need a new language

Reality

All AOP implementations have a new language



The Only Choice is the Flavor of the Language

- **Extend language**
`pointcut loggedOps() : execution(* Account.*(..));`
- **Use annotations**
`@Pointcut("execution(* Account.*(..))")
public void loggedOps() {}`
- **Use XML**
`<pointcut name="loggedOps"
expression="execution(* Account.*(..))"/>`



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
Dynamic languages obviate AOP

Reality
Dynamic languages still need AOP

TechTarget
The Best Target of All

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Dynamic Languages

- **Dynamic languages allow for some AOP-ish features**
 - Open classes
 - Method redefinition
- **However,**
 - Lack ways to express crosscutting functionality
 - Ruby 2.0 is considering **adding** AOP support

TechTarget
The Best Target of All

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
Annotations make AOP useless


Reality
Annotations simplify using AOP in some cases
and
AOP makes annotations useful!

TechTarget
The Best Target of All

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

AOP and Annotations

- **Two relationships**
 - AOP as annotation consumer
 - AOP as annotation supplier
- **Third relationship**
 - Annotations to express AOP constructs




TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

AOP as Annotation Consumer

- **Defining pointcut**

```
pointcut transactedOps() : @annotation(Transactional);  
  
pointcut transactedOps(Transactional tx) : @annotation(tx);  
  
pointcut transactedRequiredNewOps(Transactional tx)  
: @annotation(tx) && if(tx.value() == REQUIRED_NEW);
```



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

AOP as Annotation Supplier

- **Avoiding annotation clutter**

```
declare @method  
: * Account.*(..)  
: TransactionAttribute(Required);
```



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Annotations to Express AOP Constructs

```
@Aspect public class BankLoggingAspect {
    private static Logger _logger = Logger.getLogger("banking");

    @Pointcut("execution(* *.* (..) && !within(BankLoggingAspect)")
    public void loggedOperations() {}

    @Before("loggedOperations()")
    public void log(JoinPoint.StaticPart thisJoinPointStaticPart) {
        Signature sig = thisJoinPointStaticPart.getSignature();
        _logger.log(Level.INFO, sig.getDeclaringType().getName(),
            sig.getName(), "Entering");
    }
}
```

TechTarget
The Java Experts

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth

Aspects can't be unit tested

Reality

Aspects can be unit tested quite easily

TechTarget
The Java Experts

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Unit Testing with AOP

- **Testing business classes**
 - Switch off aspects to test business logic only
 - or —
 - Test classes the normal way
- **Testing aspects**
 - Mock objects to simulate the environment
- **Many of the lessons from OO apply**
 - Cleaner separation of interfaces and implementations
 - Dependency injection

TechTarget
The Java Experts

TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Outline

- Technological myths
- Cultural myths
- Adoption myths

TechTarget
 The Way You Think

TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Myth
 AOP promotes sloppy design

Reality
 Every methodology/language needs to be used carefully; AOP is no exception

TechTarget
 The Way You Think

RED MEAT Lynndie-ing javac from the Z axis from the secret files of max cannon

I've been banging my head against this code for 6 hours, and I'm convinced that this hashmap is possessed by evil spirits.

Weird. And you're taking into account the new side effects of equals()? Side-effects of .equals()?

Yeah, we've got an aspect in place that causes .equals to return false if it's called from within any class that begins with the letter 'P'

I hate you more than ever, AspectJ.

Wait until you see what it's done to your database.

Understanding is the Key

- **It takes experience to use any technology optimally**

- No one learned OOP in a day! (or months, or years, or decades!)

- **Making mistakes is a part of the game**

- Overuse of inheritance in OOP – learned only by overusing!



Myth

Aspects can be added in badly designed systems

Reality

AOP encourages and rewards good design



Good Design is Still Needed

- **It is easy to implement aspects when**

- Classes and methods implement clear responsibilities
- Classes and interfaces use natural inheritance hierarchy
- Teams use consistent naming conventions

- **It is easy to test classes and aspects when**

- Coupling is minimized
- Inversion of Control is utilized



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
AOP adoption requires subscribing to the AOP way

Reality
Many aspects pave the way for incremental adoption

TechTarget
The Java Experts
®

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Many "Starter" Aspects:
No-risk, Non-production, Fun to Learn

- **Tracing aspects**
 - Find out problem during development
 - Allow trace-enabled QA
- **Policy enforcement**
 - Enforce policies at compile-time
 - Detect violations during development and QA
 - Swing thread safety
- **Testing aspects**
 - Inject faults to test "sad paths" and improve code coverage

TechTarget
The Java Experts
®

TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Myth
AOP is too complex


Reality
...except for all alternatives

TechTarget
The Java Experts
®

TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

No Rocket Science


- **AOP isn't trivial, but it is no rocket science**
- **Don't believe me or other AOP proponents**
 - Use AspectJ for one full day



TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Conclusion


- **AOP isn't a panacea, but a very helpful tool in implementing and thinking about crosscutting concerns**
- **AOP may be seen as hard since we are so used to OOP**
- **OOP can be stretched, but**
 - Resistance is futile, you will be assimilated!
- **Don't believe what I said**
 - Try it out... and form *your own* opinion



TheServerSide.COM
JAVA in ACTION
 An Enterprise Java Conference and Training Experience

Resources

- **All about AOP**
 - <http://aosd.net>
- **AOP tools**
 - AspectJ: <http://eclipse.org/aspectj>
 - Spring: <http://springframework.org>
 - JBoss/AOP: <http://www.jboss.org/developers/projects/jboss/aop>
- **AOP design patterns project**
 - <http://www.cs.ubc.ca/labs/spl/projects/aodps.html>
- **AspectJ in Action**
 - <http://manning.com/laddad>
- **AOP in Ruby**
 - <http://www.rubygarden.org/ruby?AspectOrientedRuby>




TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Break sponsored by



TheServerSide.COM
JAVA in ACTION
An Enterprise Java Conference and Training Experience

Questions

Ramnivas Laddad
Author, AspectJ in Action
ramnivas@aspectivity.com
<http://ramnivas.com>

Training and consulting available

