ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

New in the World of Spring:

A Look at Spring 2.0 and Related Projects

Colin Sampaleanu
Interface21

Spring

About Me

- Spring Framework core developer since mid-2003
- Founder and Principal Consultant at Interface21, a unique consultancy devoted to Spring Framework and Java EE
  - Training, consulting and support
  - “From the Source”
  - http://www.interface21.com

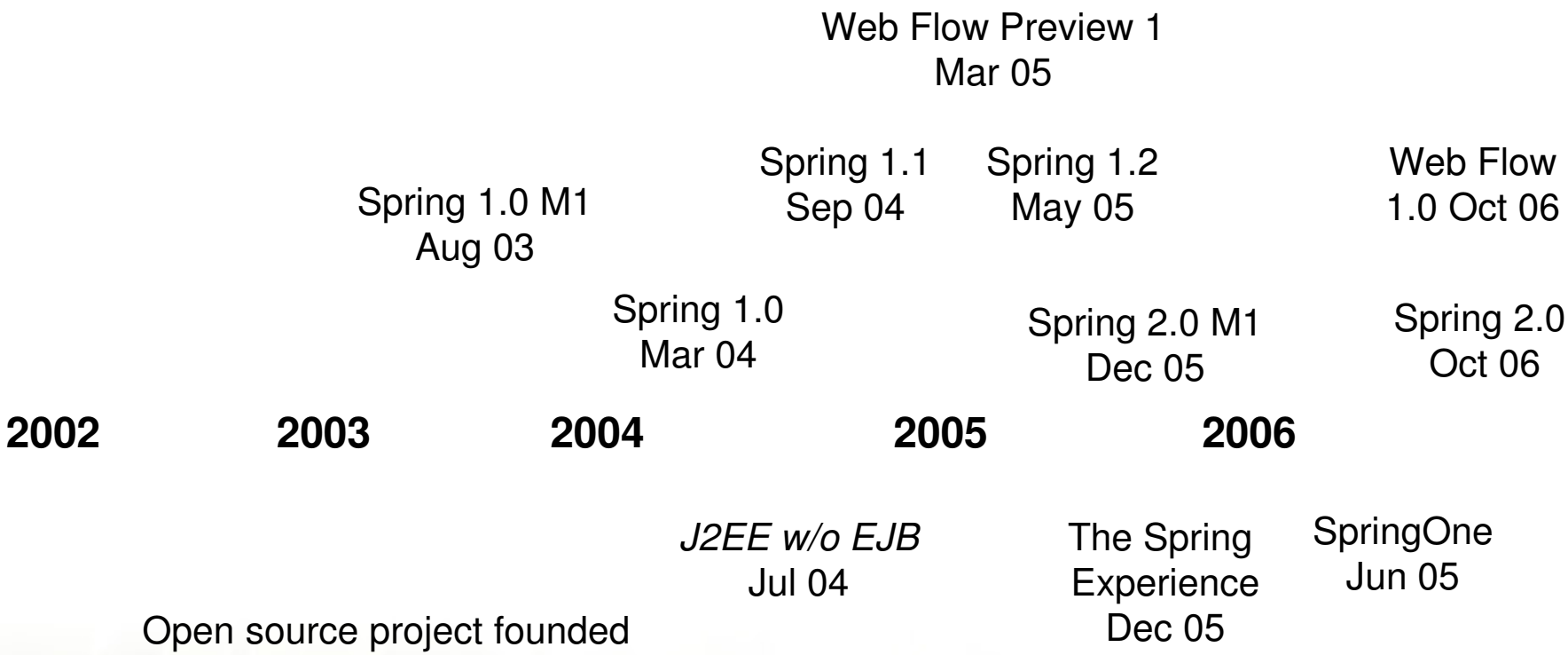
Where is Spring Today?

Interface21

Where is Spring today?

- **The favourite and most trusted Java application framework**
- **Most powerful technology for enabling POJO-based application development**
- Widely adopted across most industries and proven in many demanding applications

How did we get here?

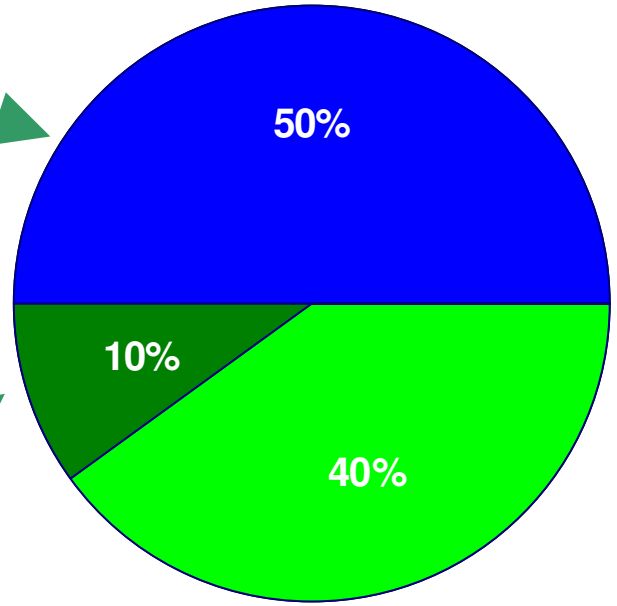


Who's using Spring?

- Spring is used in countless applications, large and small
- Very well established in the enterprise space, including
  - Banking / Financials
  - Telecom
  - Insurance
  - Government
  - Airline industry
  - Defence
  - …
- … all the way down to mom & pop development shops

Focus: Banking

- 5 out of the world's 10 largest banks are Spring users and Interface21 clients
- At least one more is also using Spring on multiple projects



Source: *The Economist* – "The world's biggest banks" (2005)

Focus: Banking: Voca

- Part of UK's Critical National Infrastructure
  - They process Direct Debits, Direct Credits and Standing Orders to move money between banks
  - Over 5 billion transactions worth €4.5 trillion in 2005
  - Some 15% of Europe's Direct Debits and Direct Credits are handled by Voca
  - Over 70% of the UK population use Direct Debits to pay household bills; Direct Credits are used to pay over 90% of UK salaries
  - Over 72 million items on a peak day
  - They have never lost a payment
- Rolling out Spring-based implementation of this infrastructure to replace legacy version!

Usage Summary

- It's known
- It's trusted
- It's used
- It adds value

- *…Everywhere*

- Analyst conclusion (Forrester – *Health of Open Source*)
  - *A majority of [enterprise Java] users interviewed by Forrester use Spring*

ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

Spring 2.0 Preview

Interface21

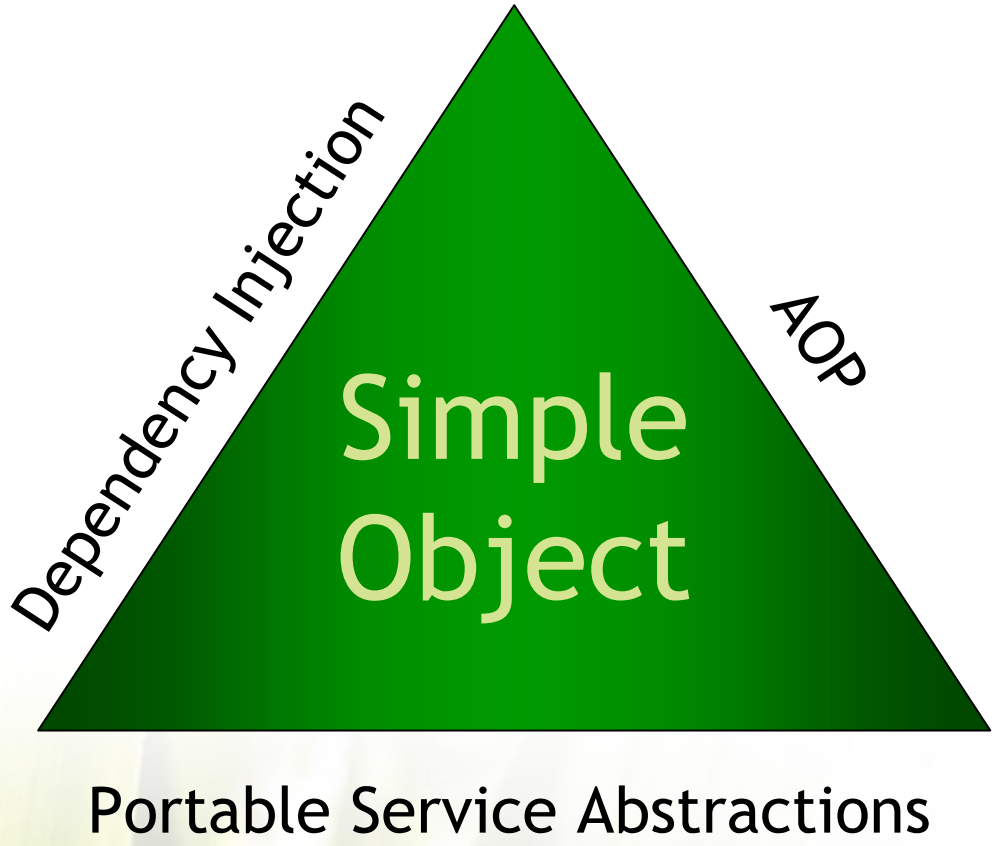
Spring

Spring has always delivered great value based on 3 enabling technologies…

… working together to enable POJO-based software development to be agile, fun, and productive

… there's a bit of magic here, it's adds up to more than the sum of the parts

Enabling technologies



Spring 2.0 – taking it to the next level

- Builds on the solid base established by Spring 1.x
- Pursues vision of POJO-based development
- Adds new capabilities and makes many tasks more elegant
  - make Spring more ***powerful***
  - … while ***simplifying*** common tasks
- … with full backwards compatibility

Spring 2.0: Backwards compatible

- Huge user base demands full backwards compatibility
  - Old code still works
- Runs on existing infrastructure
  - Java 1.3, Java 1.4, Java 5
  - A myriad of Java EE app server versions, old and new (WebSphere, WebLogic, JBoss, OC4J, etc.).
  - Simple Servlet engines (TomCat, Resin, etc.)
  - Standalone app (Swing/RCP) environments

Spring 2.0: New Features (1)

- Major enhancements and new features across the board, especially…
  - Simpler, more extensible XML configuration
  - Enhanced AOP functionality and integration with AspectJ
    - Leveraging new XML config

Let's take a look at these two area first…

XML

XML Configuration in Spring 2.0

- Ability to define new XML tags to produce one or more Spring bean definitions
- Tags out of the box for common configuration tasks
- Problem-specific configuration
  - Easier to write and to maintain
- XML schema validation
  - Better out of the box tool support
  - Code completion for free
- Exploits the full power of XML
  - Namespaces, schema, tooling

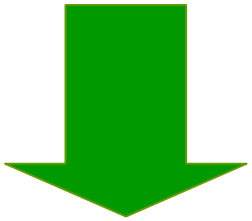
XML Configuration in Spring 2.0

- Backward compatibility
  - Full support for <beans> DTD
  - Complete interoperability between classic and extended configuration
- Choices
  - Flexible, generic configuration
  - Targeted, problem-specific configuration
- Ideal for third parties developing Spring components, and very large projects

XML Configuration in Spring 2.0

A JNDI lookup...

```
<bean id="dataSource" class="...JndiObjectFactoryBean">
  <property name="jndiName" value="jdbc/StockData"/>
  <property name="resourceRef" value="true"/>
</bean>
```



```
<jee:jndi-lookup id="dataSource"
    jndiName="jdbc/StockData" resourceRef="true"/>
```

Code Completion in IDE

Extended Configuration Options

- Out-of-the-box namespaces:
  - `<jee:*/>`
    - JEE related configuration
  - `<util:*/>`
    - Load Properties instances, create constants, etc.
  - `<aop:*/>`
    - Simplified standard AOP configuration
    - Expose new AspectJ-style advice
  - `<tx:*/>`
    - Transaction simplification - DSL for concise definitions
    - Improved tool support (code-asist) for transactional advice
    - Enable annotation-driven transactions in a single line
      - `<tx:annotation-driven />`

XML Configuration Best Practices

- Standard `<bean>` tags
  - Still a great solution
  - General configuration tasks
  - Application-specific components
    - DAOs, Services, Web Tier
- Custom tags
  - Infrastructure tasks
    - JNDI, Properties, AOP, Transactions
    - Supplied with Spring
  - 3rd party namespaces
    - Whatever **YOU** or **YOUR USERS** need

AOP

AOP in Spring 2.0

- AOP is important…
  - How do we make Spring AOP better?

- Simplified XML configuration using <aop:*/> tags
- Closer AspectJ integration
  - Pointcut expression language
  - AspectJ-style aspects in Spring AOP
  - @AspectJ-style aspects in Spring AOP
    - Fully interoperable with ajc compiled aspects
- Spring ships with AspectJ aspects for Spring/AspectJ users
  - Dependency injection on any object even if it isn't constructed by the Spring IoC container

Spring 2.0 aims for Spring AOP

- Build on strengths, eliminate weaknesses
- Preserve ease of adoption
  - Still zero impact on development process, deployment
  - Easier to adopt
- Benefit from the power of AspectJ
- Provide a comprehensive AOP roadmap for Spring users, spanning
  - Spring AOP
  - AspectJ
- Remains fully backwards compatible with Spring 1.x releases

Spring 2.0 and AspectJ

- Spring and AspectJ are still distinct projects
- Spring just uses the AspectJ pointcut parsing and matching APIs
  - using AspectJ as a library, not as a weaving engine
- Gives the same syntax and semantics across Spring AOP and AspectJ
  - perfect if you are going to use both
  - or start out with Spring AOP, and then want to introduce AspectJ at some point

Pointcut Expressions

- Spring can use AspectJ pointcut expressions
  - In Spring XML
  - In @AspectJ aspects
  - In Java code (with Spring `ProxyFactory`)
- New `AspectJExpressionPointcut` will become the most used Spring AOP Pointcut implementation

What's so good about AspectJ pointcut expressions?

- Go far beyond simple wildcarding
- AspectJ views pointcuts as first-class language constructs
  - Can compose pointcuts into expressions
  - Can reference named pointcuts, enabling reuse
  - Can perform argument binding…
  - Can express complex matching logic *concisely*
- Well documented in many books/articles

AOP is *about* pointcuts

- Pointcuts give us the tool to think about program structure in a different way to OOP
- Without a true pointcut model we have only trivial interception
  - Does not achieve aim of modularizing crosscutting logic
  - **DRY (Don't repeat yourself)**
- Spring AOP has always had true pointcuts
  - But now they are dramatically improved

@AspectJ-style Aspects

```
@Aspect
public class AjLoggingAspect {

    @Pointcut("execution(* *..Account.*(..))")
    public void callsToAccount(){}

    @Before("callsToAccount()")
    public void before(JoinPoint jp) {
       System.out.println("Before [" +
                   jp.toShortString() + "].");
    }

    @AfterReturning("callsToAccount()")
    public void after() {
        System.out.println("After.");
    }
}
```

@AspectJ-style Aspects

```
<!– tell spring-aop to treat any beans that are @AspectJ aspects
    as such -->
<aop:aspectj-autoproxy/>

<bean id="account" class="demo.Account"/>

<!– define a bean that is an @AspectJ aspect, DI as normal… -->
<bean id="aspect" class="demo.ataspectj.AjLoggingAspect"/>
```

POJO Methods as Advice

```
public class JavaBeanPropertyMonitor {

   private int getterCount = 0;
   private int setterCount = 0;

   public void beforeGetter() {
      this.getterCount++;
   }

   public void afterSetter() {
      this.setterCount++;
   }
```

Applying Pointcuts: Via XML

INTERFACE21

```
<aop:config>
  <aop:aspect bean="javaBeanMonitor">
    <aop:before
        pointcut=
          "execution(public !void get*())"
        method="beforeGetter"/>
    <aop:after
        pointcut=
          "execution(public void set*(*))"
        method="afterSetter"/>
  </aop:aspect>
</aop:config>
<!- just a regular bean -->
<bean id="javaBeanMonitor" class="..."/>
```


Spring

AOP in Spring 2.0

- Many other enhancements
  - @Configurable annotation for dependency injection of domain objects
  - Full integration with new “tx” XML namespace or @Transactional annotation
  - Use of old interceptors

New Transaction Configuration: XML

Using Spring 2.0's concise "tx" namespace:

```
<aop:config>
   <aop:advisor pointcut="execution(* *..OrderService+.*(..))"
                advice-ref="txAdvice"/>
</aop:config>

<tx:advice id="txAdvice">
   <tx:attributes>
      <tx:method name="get*"/>
      <tx:method name="process*" propagation="REQUIRES_NEW"/>
   </tx:attributes>
</tx:advice>

<bean id="transactionManager"
   class="org.springframework.transaction.jta.JtaTransactionManager"/>

<bean id="orderService" class="foo.OrderServiceImpl"/>
```

More Powerful

More Concise

Less Chance of Error (Code-Completion in IDE)

Transaction configuration via Annotations

Code example:

```
@Transactional
public class OrderServiceImpl implements OrderService {

   @Transactional(readOnly=true)
   public List getOrdersForCustomer(Customer customer) {…}

   public void processOrder() {…}
}
```

Spring 2.0 configuration with “tx” namespace:

```
<tx:annotation-driven/>
```

That’s it!...

Spring will automatically detect @Transactional annotations

Other Enhancements

Spring 2.0: New Features (2)

- Additional scoping options for beans
  - Backed by HttpSession, HttpRequest, etc.
  - Pluggable backing store
    - Not tied to web tier
- Ability to define any named bean in a scripting language such as Groovy or JRuby
  - Named bean conceals both configuration and implementation language
  - Allows for DI, AOP and dynamic reloading
- JPA (Java Persistence Architecture) Integration
- JdbcTemplate simplification for Java 5

Spring 2.0: New Features (3)

- MVC Simplification: Intelligent defaulting
- New JSP form tags
- Spring Portlet MVC, an MVC framework for JSR-168 Portlets

- Asynchronous JMS facilities enabling message-driven POJOs
- Customizable task execution framework for asynchronous task execution
- CommonJ TimerManager implementation
  - Great for WebLogic/Websphere users

Code break...

Spring 2.0 - Summary

- Builds on the solid base established by Spring 1.x
- Pursues vision of POJO-based development
- Adds new capabilities and makes many tasks more elegant
  - make Spring more ***powerful***
  - … while ***simplifying*** common tasks
- … with full backwards compatibility

- Provides a new foundation for future work
  - Most capable and flexible container on the market
  - The road ahead is exciting, with much more to come!

Hot on the Heels of Spring 2.0...

Spring Web Flow 1.0 is ready

Spring Web Flow 1.0

- A full spring subproject
- Part of Spring's web stack
- Capture a logical flow of your web application as a self contained module, at a ***higher level***
  - In a declarative fashion
  - Essentially a black box, including sub flows
  - Representing a user conversation
  - Introduces new scope: Flow Scope
  - Highly portable across lower level UI frameworks
  - Highly manageable
- Available at http://www.springframework.org

Spring-LDAP

Spring-LDAP 1.1

- A full spring subproject
- Simplifies LDAP operations, based on the pattern of Spring's JdbcTemplate
- Encapsulates nasty boilerplate plumbing code traditionally required for LDAP
  - Resource management and cleanup
  - Exception handling
  - Iterating
- Lower level access to data
- Higher level mapping of domain objects
- Available at http://www.springframework.org/ldap

Spring-OSGi

OSGi, What is it?

- Industry driven framework specification, with multiple implementations
- Dynamic component model, based around the idea of *bundles*

- Classloading (for isolation & versioning)
- Lifecycle control and definition
- Service Registry
- Standard Services
- Security Model

Why do we Need Spring-OSGi?

- Spring-OSGi is an integration library for Spring in OSGi environments
- *For those that need it*, allows a more powerful component programming model
  - Without Spring having to re-invent the wheel
  - ApplicationContexts become bundles, able to import and export services, with full isolation, integrated into OSGi lifecycles
- Project moving along rapidly, with large amount of interest and involvement from vendors such as BEA, Oracle, IBM, members of the OSGi foundation, and the general public

- Find it at http://www.springframework.org/osgi

If you like SpringForward…

… you'll love The Spring Experience

The Spring Experience 2006
December 7th – 10th, Hollywood Florida

by Interface21 and NoFluffJustStuff Java Symposiums

- World-class technical conference for the Spring community
- Experience 3 full days, 55 sessions across 5 tracks
    1. Core Spring 2.0
    2. Core Enterprise 2.0
    3. Core Web 2.0
    4. Domain-Driven Design
    5. Just Plain Cool
- Enjoy five-star beach resort and amenities
- Converse with core Spring team and industry experts
    - Rod Johnson, Adrian Colyer, Ramnivas Laddad, Juergen Hoeller
    - Eric Evans, Luke Hohmann, Eamon McManus
- **Register at http://www.thespringexperience.com**

THE SPRING EXPERIENCE
December 7-10, 2006
Hollywood, Florida

- Questions?