Spring ROO

Internals and add-ons

Speaker

Software Architect / Developer
ProNetics / Sourcesense

Chairman
JugSardegna Onlus

Founder
SpringFramework Italian User Group

Committer - Contributor
OpenNMS - MongoDB - MagicBox

Author
Spring 2.5 Aspect Oriented Programming

Agenda

- Demo for unbelievers-

- Roo Internals -

- Maven Roo support -

- Add On Anatomy -

- Roo Domain Driven Design -

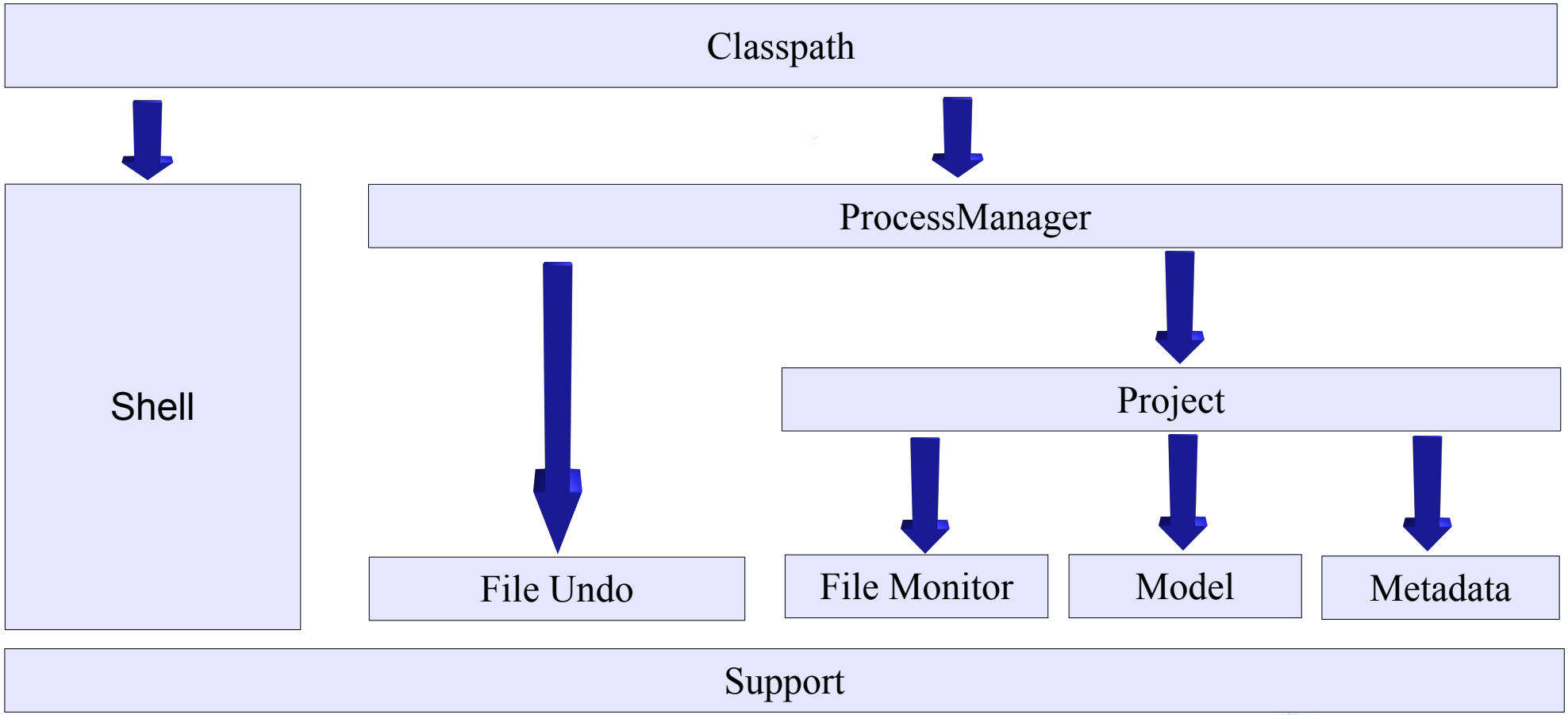
- AspectJ Intertype Declaration -

- SpringSurf Add-on -

5 Minutes demo for Unbelievers

Roo internals

Core modules



Roo in its essence is an interactive shell

and a ProcessManager

The `Shell` receives the commands

and a `ProcessManager`

coordinates the operations

required from the user

- The Shell has a very good usability -

(tab, context awareness, hiding, hints)

- Background monitoring of externally-made changes -

- Full interaction with Spring Tool Suite -

The `Shell`

parses and validates the user input

and publishes the status about current task

STARTING, *STARTED*, *USER_INPUT*, *PARSING*, *EXECUTING*,
EXECUTION_RESULT_PROCESSING,
EXECUTION_COMPLETE, *SHUTTING_DOWN*

The `Shell` passes the `ParseResult` object

to the execute method

of the `ProcessManager`

(Strategy pattern)

`ProcessManager` delivers:

- State publication model -

- Polling on filesystem resources -

- Execute CommandCallbacks for requested operations -

- "Transaction-like" context for above operations -

Commands

During the bootstrap ROO

reads all bean classes

of type `CommandMarker` (Tag interface)

The `Shell` stores the metadata information (`@CliAvailabilityIndicator`) read from all `CommandMarker` beans

In this way the `Shell` knows all available commands and `MethodTarget` related to user input

The Command class (`CommandMarker`)

uses the related Operations class.

The Operations class

performs the real work

on Java and AspectJ classes, XML and template files.

Maven ROO support

sometimes maven is your friend :)

Roo uses Maven under the hood for:

building the entire project and managing dependencies

```
roo> project --topLevelPackage com.proj.roo.addon
```

add-on development

```
roo> project --topLevelPackage com.proj.roo.addon --template ROO_ADDON_SIMPLE
```

eclipse / STS

```
roo> perform eclipse
```

packaging

```
roo> perform assembly
```

Add-On anatomy

Security add-on in the example

```
@ScopeDevelopmentShell
public class SecurityCommands implements CommandMarker{

    private SecurityOperations securityOperations;

    public SecurityCommands(SecurityOperations securityOperations) {
        Assert.notNull(securityOperations, "Security operations required");
        this.securityOperations = securityOperations;
    }

    @CliAvailabilityIndicator("security setup")
    public boolean isInstallSecurityAvailable() {
        return securityOperations.isInstallSecurityAvailable();
    }

    @CliCommand(value = "security setup",
                help = "Install Spring Security into your project")
    public void installSecurity() {
        securityOperations.installSecurity();
    }
}
```

Security add-on

```
@ScopeDevelopmentShell
public class SecurityCommands implements CommandMarker
…
public SecurityCommands(SecurityOperations securityOperations) {
        Assert.notNull(securityOperations, "Security operations required");
        this.securityOperations = securityOperations;
}
```

`@ScopeDevelopmentShell` : Indicates a class that should be instantiated when the ROO development Shell is used.

`CommandMarker` : Tag Interface, the Command class must implement this interface

Commands

```
@CliAvailabilityIndicator("security setup")
public boolean isInstallSecurityAvailable() {
    return securityOperations.isInstallSecurityAvailable();
}
```

`@CliAvailabilityIndicator`: Annotates a method that can indicate whether a particular command is currently available or not.

This annotation must only be applied to a public no-argument method that returns primitive boolean.

The method should be inexpensive to evaluate, as this method can be called very frequently. If expensive operations are necessary to compute command availability, it is suggested the method return a boolean field that is maintained using the observer pattern.

Commands

```
@CliCommand(value = "security setup",
            help = "Install Spring Security into your project")
public void installSecurity() {
   securityOperations.installSecurity();
}
...
```

`@CliCommand`

`value`: one or more strings which must serve as the start of a particular command in order to match this method (these must be unique within the entire application)

`help`: help message for this command (the default is a blank String, which means there is no help)

Operations

```
@ScopeDevelopment
public class SecurityOperations

    private FileManager fileManager;
    private PathResolver pathResolver;
    private MetadataService metadataService;
    private ProjectOperations projectOperations;

....
```

The SecurityOperations uses the above convenient Roo API to fulfill its purpose:
- Find/write and read any resources (Java, XML, JSP, Properties, AJ) on filesystem
- Read metadata
- Handle the POM file

Recap, add-on should:

- Make a "Command" class and implement `CommandMaker` -
- Delegate methods through to an "Operations" object -
- Annotate methods with `@CliCommand` -
- Annotate method parameters with `@CliOption` -
- Optionally use `@CliAvailabilityIndicator` if desired -
- Throw any exceptions to abort and rollback changes -
- Use JDK logging or return objects for console output -

1.0.1 Base Add-Ons

- Backup
- Bean Info
- Configurable
- Data On Demand
- Email
- Entity
- Dynamic Finder
- Java Bean
- JMS
- JPA
- Logging
- Maven
- Pluralization
- Property Editor
- Property File
- Security
- Integration Test
- ToString
- Web (various)

Roo Domain Driven Design

Demo for unbelievers
Roo Internals
Maven Roo support
Add On Anatomy
Roo Domain Driven Design
AspectJ Intertype Declaration
SpringSurf Add-on

Roo uses a pragmatic design approach.

Following the Domain Driven Design principles,

Roo avoids

Anemic Objects and prefers Rich Domain Objects

that follows Object Oriented principles.

This means encapsulation, immutability and

proper role of domain objects

Do you remember the controller-entities interaction of this slide ? Roo works in a similar way

JAVADAY ROMA III EDIZIONE

DDD UI

```
@Controller("customerController")
public class CustomerController {

   @RequestMapping("/customer.create.page")
   public ModelAndView create(HttpServletRequest req) {
      return new ModelAndView("customer/create", "result", UiUtils.getCustomer(req).createOrder());
   }

   @RequestMapping("/customer.order.page")
   public ModelAndView order(HttpServletRequest req) {
      return new ModelAndView("customer/order", "order", UiUtils.getOrder(req));
   }

   @RequestMapping("/customer.items.page")
   public ModelAndView items(HttpServletRequest req) {
      return new ModelAndView("customer/items", "items", UiUtils.getOrder(req).getOrderItems());
   }
...
}
```

I controller risultanti saranno completamente stateless e senza dipendenze, e con una semplice chiamata sulla entità.

Roo by default uses two layers:

An entity layer (similar to domain layer)

and a web layer (REST).

Roo achieves separation of concern

of these two layers

through AspectJ ITD-based architecture.

- REST Controllers work directly with Domain entities -

- OpenEntityManagerInViewFilter -

- JPA EntityManager bound to the thread for the entire processing of the request -

- allow for lazy loading in web views -

- original transactions already being completed -

- Service layer is optional, it can be created and used for business logic spanning multiple entities -

AspectJ ITD

(where Roo shines)

Demo for unbelievers
Roo Internals
Maven Roo support
Add On Anatomy
Roo Domain Driven Design
AspectJ Intertype Declaration
SpringSurf Add-on

Do you remember the concept of this slide ?

JAVADAY ROMA III EDIZIONE

Spring AOP – Introductions

Una introduction mi permette di decorare un oggetto a runtime, aggiungendogli interfacce e relativa implementazione.
Questo permette sia di evitare la duplicazione di una implementazione, sia di simulare l' ereditarietà multipla che java non ha.

```
@Aspect
public class ParallelepipedIntroduction {

    @DeclareParents(value = "org.springaop.chapter.four.introduction.Box",
        defaultImpl = Titanium.class)
    public Matter matter;

    @DeclareParents(value = "org.springaop.chapter.four.introduction.Box",
        defaultImpl = Cube.class)
    public GeometricForm geometricForm;
}
```

The Spring Introductions in the Aspect world

are called Inter-Type Declarations (ITD) .

Roo uses AspectJ ITD

instead of Spring Introductions (proxy based).

With ITD Roo performs active generation.

The ITD enables ROO through the add-on

to write the custom logic

into <name_class>_Roo_<addOn>.aj files

related to the corresponding java class

AspectJ compiler does the rest

by linking the pieces of the puzzle :)

For example the add-ons Entity, JavaBean, ToString generates the proper logic in corresponding .aj files.

*_Roo_Entity.aj

*_Roo_JavaBean.aj

*_Roo_ToString.aj

With AspectJ ITD

Roo can implement

an effective Object Oriented

Domain Driven Design

application

Alfresco SpringSurf add-on

Demo for unbelievers
Roo Internals
Maven Roo support
Add On Anatomy
Roo Domain Driven Design
AspectJ Intertype Declaration
SpringSurf Add-on

"Spring Surf is a view composition framework

for Spring MVC that plugs into

your existing Spring applications.

It provides

a scriptable and content-centric approach

to building web applications "

Surf is born from the alfresco community

need of a lightweight approach

to create application services

and

user interface extensions

instead of JSF or Struts

Spring Surf is an official

Spring extension by Alfresco.

(http://www.springsource.org/extensions/se-surf)

Roo enables scaffolding for

Surf pages, templates and components.

The Spring Surf Roo add-on

shows one of the many possibilities

that Roo provides with its API

Spring-Surf mini demo

```
magicbox:surf-app max$ roo
    ____  ____  ____
   / __ \/ __ \/ __ \
  / /_/ / / / / / / /
 / _, _/ /_/ / /_/ /
/_/ |_|\____/\____/    1.0.1.RELEASE [rev 601]

Welcome to Spring Roo. For assistance press TAB or type "hint" then hit ENTER.
roo> surf

surf addon install                  surf component create
surf component list                 surf component property create
surf component resource create      surf content association create
surf page association create        surf page association list
surf page create                    surf page list
surf report page                    surf site create
surf template create                surf template instance create
surf template instance list         surf template list
surf template region list           surf webscript list
roo> surf
```

Q & A

Project resources

- Home
- http://www.springsource.org/roo
- Contains links to all other resources

- Forum
- http://forum.springsource.org

- Issues
- http://jira.springframework.org/browse/ROO

- Twitter
- #roo hash key
- follow @schmidtstefan & @benalexau

Thanks for the attention !

Massimiliano Dessì
desmax74 at yahoo.it

massimiliano.dessi at pronetics.it

http://twitter.com/desmax74

http://jroller.com/desmax

http://www.linkedin.com/in/desmax74

http://www.slideshare.net/desmax74

http://wiki.java.net/bin/view/People/MassimilianoDessi

http://www.jugsardegna.org/vqwiki/jsp/Wiki?MassimilianoDessi