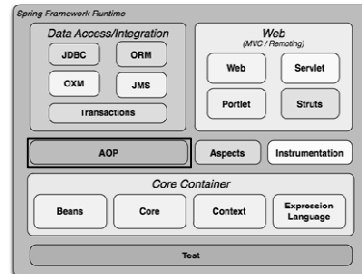


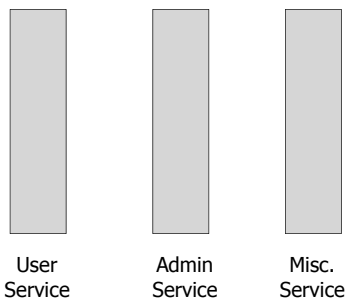
## CS520 Web Programming Spring – Aspect Oriented Programming

Chengyu Sun  
California State University, Los Angeles

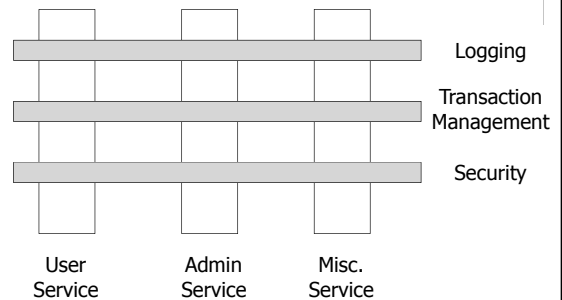
## Spring Framework



## Concerns



## Cross-Cutting Concerns



## An Example of Transaction Rollback

```
catch( SQLException e ) {  
    System.err.println( e.getMessage() );  
    if( c != null ) {  
        try {  
            c.rollback();  
        } catch( SQLException ex ) {  
            System.err.println( ex.getMessage() );  
        }  
    }  
}
```

## Aspect Oriented Programming

- ◆ Separate out cross-cutting concern code into their own classes or modules, called aspects.

## Example: Logging

- ◆ LoggedTask1 and LoggedTask2
  - Both implement LoggedTask interface
  - LoggedTask1 – mixes application code and logging code
  - LoggedTask2 – only has application code

## How Does it Work?

```
class Foo {  
    Object bar() {...}  
}
```

run some code before bar() is called

run some code after bar() is called

## Proxy – Subclass

```
class Foo1 extends Foo {  
    void bar()  
    {  
        // some code before super.bar()  
        ...  
        Object o = super.bar();  
        // some code after super.bar();  
        ...  
        return o;  
    }  
}
```

## Proxy – Wrapper Class

```
class Foo2 {  
    private Foo foo;  
    void bar()  
    {  
        // some code before super.bar()  
        ...  
        Object o = foo.bar();  
        // some code after super.bar();  
        ...  
        return o;  
    }  
}
```

## Some AOP Terminology

- ◆ Target
- ◆ Proxy
- ◆ Proxy Interface
- ◆ Advice

## Create Proxies Automatically - ProxyFactoryBean

```
<bean id="loggedTask2" class="cs520.spring.aop.LoggedTask2" />  
  
<bean id="loggedTask2WithAdvice"  
    class="org.springframework.aop.framework.ProxyFactoryBean">  
    <property name="proxyInterfaces">  
        <list>  
            <value>cs520.spring.log.LoggedTask</value>  
        </list>  
    </property>  
    <property name="target" ref="loggedTask2" />  
    <property name="interceptorNames">  
        <list>  
            <value>loggingAdvice</value>  
        </list>  
    </property>  
</bean>
```

## More AOP Terminology

- ◆ Join point: a point in the execution of the application where the advice can be plugged in
- ◆ Pointcut: A predicate that determines join points
- ◆ Introduction: adding new methods and/or fields to existing classes
- ◆ Weaving
  - *Compile time, class load time, or runtime*

## Spring AOP

- ◆ Advices are written in Java
- ◆ Pointcuts are defined in XML or annotation
- ◆ Supports only method join points
- ◆ Aspects are woven in at runtime
- ◆ Advisor = Advice + Pointcuts

## Advice Types

Type	Interface
Around	org.aopalliance.intercept.MethodInterceptor
Before	org.springframework.aop.BeforeAdvice
After	org.springframework.aop.AfterReturningAdvice
Throws	org.springframework.aop.ThrowsAdvice

## Use Interceptor

```
public class LoggingInterceptor implements MethodInterceptor {  
    public Object invoke( MethodInvocation invocation ) throws Throwable  
    {  
        // do something before method invocation  
        ...  
  
        Object result = invocation.proceed();  
  
        // do something after method invocation  
        ...  
  
        return result;  
    }  
}
```

## Configure Pointcuts

- ◆ NameMatchMethodPointcutAdvisor
- ◆ RegExpPointcutAdvisor

```
<bean id="loggingAdvisor"  
    class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">  
    <property name="mappedName" value="doSomething*" />  
    <property name="advice" ref="loggingInterceptor" />  
</bean>
```

## AOP Annotations

- ◆ Spring Framework Reference Documentation, Chapter 7.2

## About AOP

◆ Good??

◆ Bad??