Introduction to Spring Integration

Mark Fisher
http://www.springsource.com

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- The Spring Integration Core API
- Configuration Options
- Channel Adapters
- Message Routing
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Loose Coupling

- **Challenges**:
  - Requirements evolve
  - Technologies change
  - Systems need to be integrated
- **Solutions**:
  - Define generic abstractions
    - Interfaces, Channels
  - Encapsulate implementation details
    - Strategies, Messages
  - Take advantage of polymorphism

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Separation of Concerns

- **Challenges**:
  - Infrastructural resources must be located and configured
  - Duplicated logic is scattered across components
  - Many responsibilities are tangled within a component
  - Testing becomes prohibitive
- **Solutions**:
  - Dependency Injection
    - Provide resources to the components
    - Isolate components from the environment
  - Aspect Oriented Programming
    - Modularize cross-cutting concerns
    - Minimize scattering and tangling
  - Delegate to templates for generic behavior

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Layered Architecture

Web Services

MVC

RMI

Services

Data Access

Infrastructure

Email

RDBMS

JMS

Domain Model

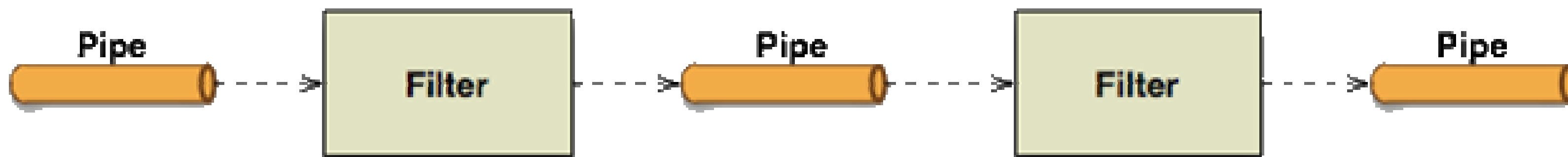
Layers enforce *separation of concerns*

Interface contracts promote *loose coupling*

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Pipes and Filters Architecture

- Pipes (messaging channels) decouple components
- Facilitates interception and monitoring
- Filter may be a service, transformer, or router
- Enables flexible service orchestration
- Channel adapters connect in/out-bound transports



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

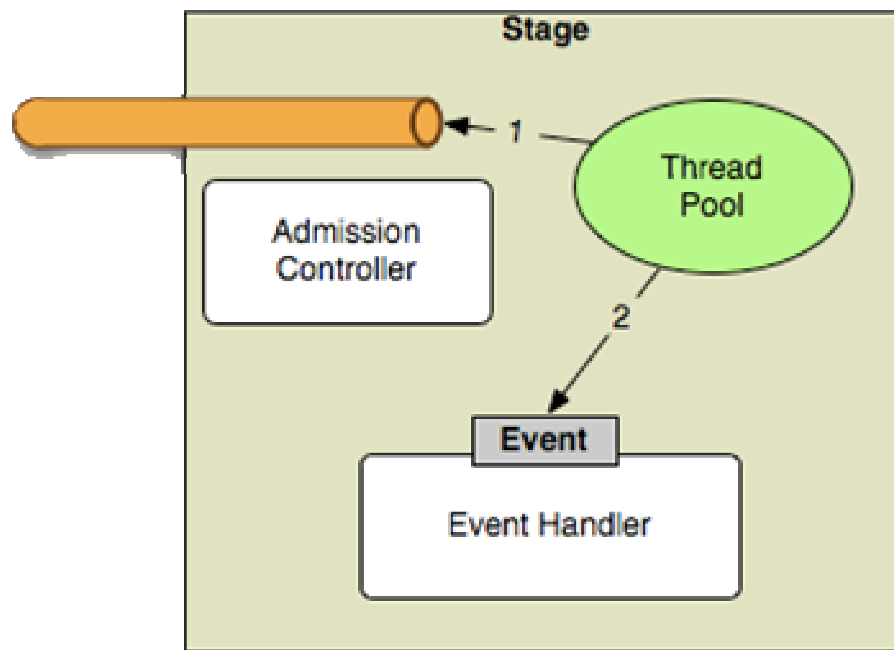
Event-Driven Systems

- Essentially Inversion of Control at runtime
- Framework polls or listens to an event source
- Framework notifies or invokes a service
- Example events
  - File written to a directory
  - JMS Message arrives on a queue/topic
  - Email received
  - Scheduled trigger fires
  - Method invocation or return value is intercepted (AOP)

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Staged Event Driven Architecture (SEDA)

- Alternative to thread-per-request server model
- Controlled number of threads per handler
- Ideal for short-lived tasks and high # of requests



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Goal 1: Keep it Simple and Non-Invasive

- Business components should not be aware of the messaging system or integration concerns
- Thread management and polling should be encapsulated but highly-configurable
- Integration logic (e.g. routing and transformation) should be isolated and testable
  - Annotated type-safe methods on POJOs
  - Dynamic language support
- Custom extension points should be well-defined
  - Strategy interfaces
  - Interceptors and AOP advice

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Goal 2: Maintain Philosophical Consistency

- Event-driven = *runtime* Inversion of Control
  - Framework handles message-listening
  - Framework handles service-invoking
- Core API design based on interfaces
- Highly customizable via strategy and template method patterns
- Provide a framework for testing
- Support multiple metadata formats
  - XML with ‘beans’
  - XSD-based namespace support
  - Annotations

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Goal 3:
Provide Seamless Integration

- Maximize leverage from the Spring foundation
  - Lifecycle management
  - Task execution abstraction
  - Aspect-Oriented Programming
  - Declarative transaction management
  - Dynamic language support
  - Spring remoting
  - JMS support
  - Scheduling

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

The Result

- Simplifies development of integration solutions by relying on proven Spring best practices and well-known Enterprise Integration Patterns
- Facilitates incremental adoption for existing Spring users who are beginning to explore SOA and EDA
- Co-evolves with other Spring portfolio products

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

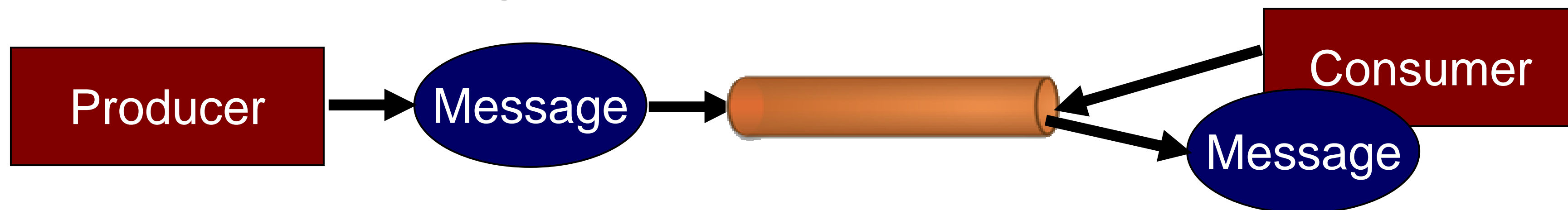
Topics

- Goals and Principles of Spring Integration
- **Enterprise Integration Patterns Overview**
- The Spring Integration Core API
- Configuration Options
- Channel Adapters
- Message Routing
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Message Channel

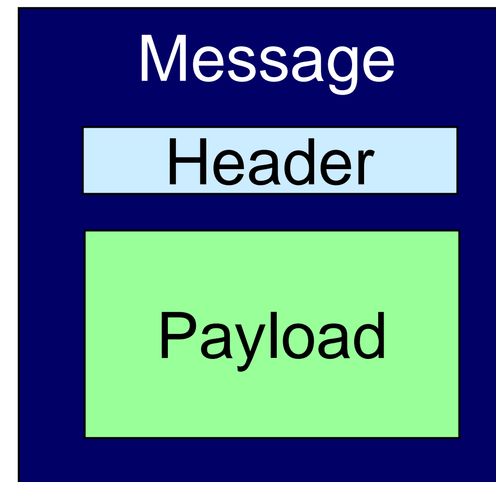
- Decouples producers from consumers
- Enforces data type consistency
- Provides a subscription strategy
  - Point-to-Point Channel
  - Publish/Subscribe Channel
- Enables message-based error handling
  - Invalid Message Channel
  - Dead Message Channel



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Message

- A generic package for data (the Message payload) that can be transported via channels
- A Message Header provides information to other components that consume from channels
  - Message ID
  - Sequence Number
  - Sequence Size
  - Expiration Date
  - Correlation Identifier
  - Return Address



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Message Endpoint

- Provides an abstraction for message producers and consumers
  - Adapts input sources and output targets
  - Handles invocation of local services
- Cleanly separates messaging concerns from business components
  - Acts as a Messaging Gateway for the application
  - Uses a Messaging Mapper to convert between Messages and domain objects
- Supports multiple consumer strategies
  - Polling or Event-driven
  - Selective Consumers
  - Competing Consumers

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Channel Adapter

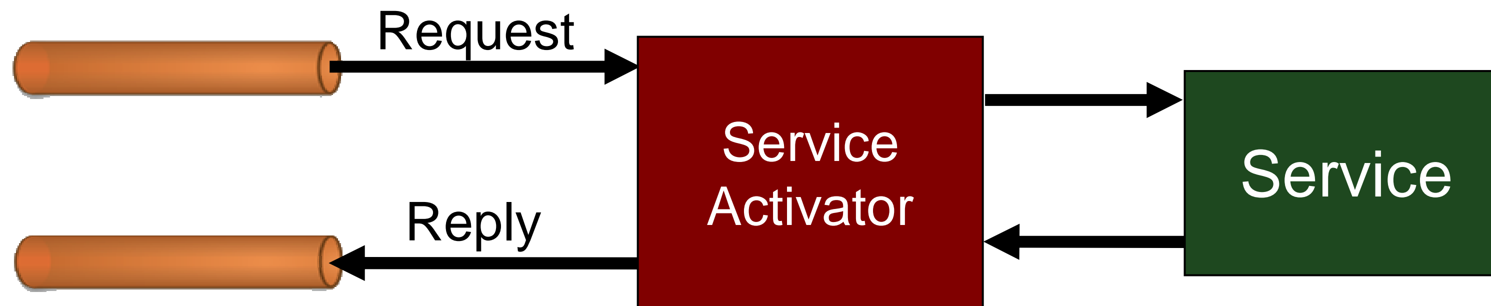
- Connect a source to the messaging system so it can send to a Message Channel

- Connect a target to the messaging system so it can receive from a Message Channel

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Service Activator

- A Message Endpoint that invokes a service
- Supports multiple communication styles
  - one-way and request-reply
  - synchronous and asynchronous
- The service is unaware of the messaging system



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

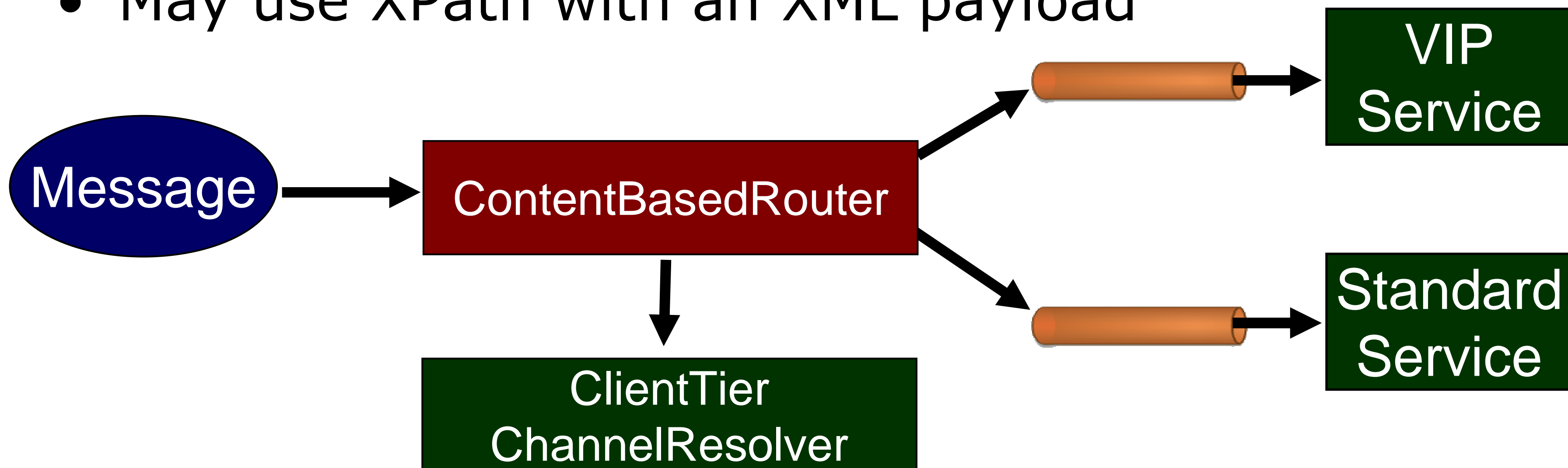
Message Router

- Route messages to message channels
- Isolate routing strategy from business logic
- Provide a dynamic alternative to publish/subscribe channels
- Accommodate complex messaging scenarios
  – Splitter
  – Aggregator
  – Resequencer

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Content Based Router

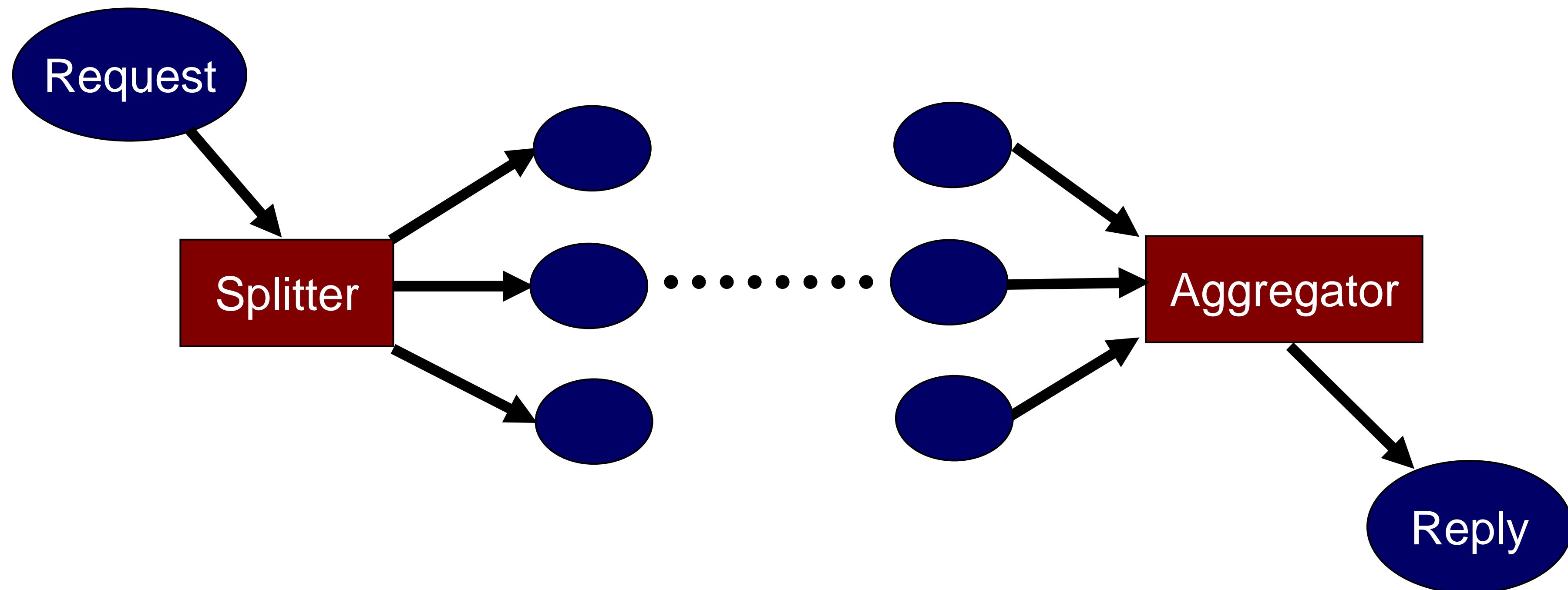
- Determine target channel based on
  - payload type
  - property value
  - custom logic applied to payload
- May define rules with EL or a scripting language
- May use XPath with an XML payload



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Splitter and Aggregator

- Divide coarse-grained message into sub-messages
- Delegate to distributed endpoints as necessary
- Recombine asynchronous reply messages



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

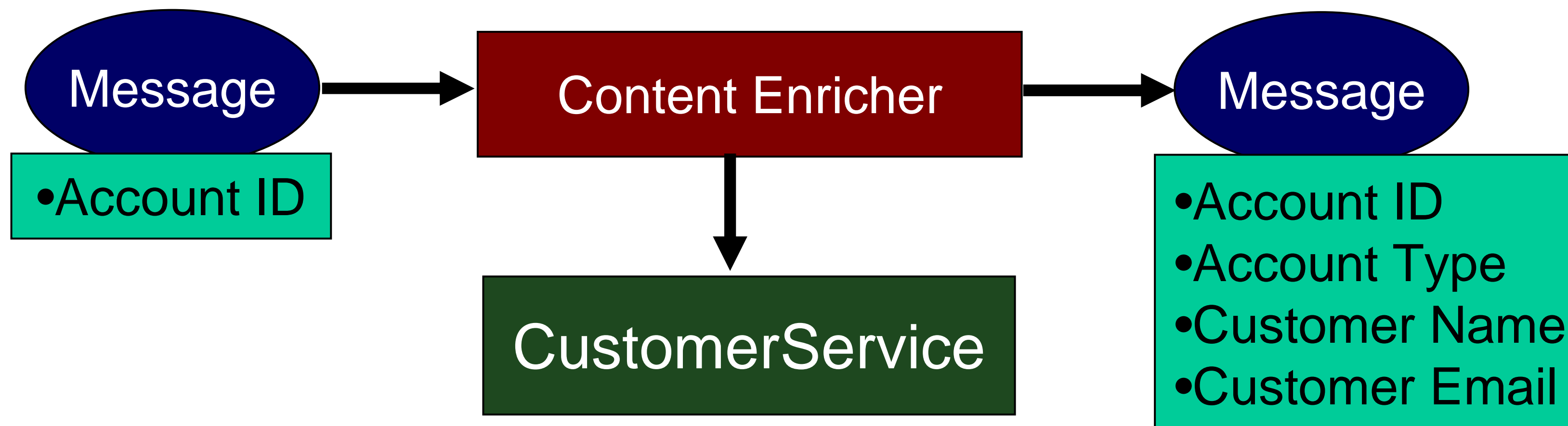
Message Translator

- Convert payload type
- Enrich message content
- Filter message content
- Normalize message format
  - Multiple clients may send multiple versions
  - The application may expect a canonical format

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Content Enricher

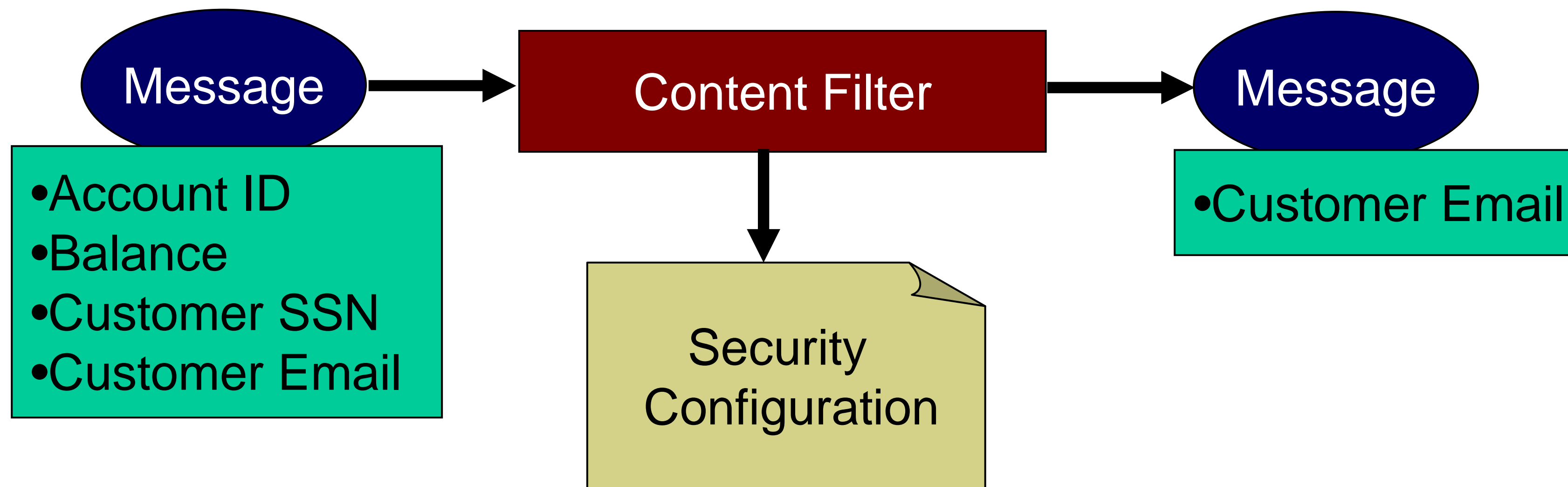
- Invoke a service that provides additional data to the payload object
- Add properties for a downstream adapter



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Content Filter

- Remove unnecessary information to reduce size
- Remove sensitive information for security purposes
- Use a *Claim Check* to save data for later



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

EIP and the Core Principles

- A Message *encapsulates* data
- A Message Channel *decouples* producers and consumers
- A Message Endpoint is an *abstraction* whose implementation may translate, route, or invoke a business service with a Message payload
- A Channel Adapter *encapsulates* the connection details and *decouples* the integrated systems
- Asynchronous invocation *separates* the polling or listening concerns from the business logic
- A message-driven architecture accommodates change and evolving business requirements

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- **The Spring Integration Core API**
- Configuration Options
- Channel Adapters
- Message Routing
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessageChannel

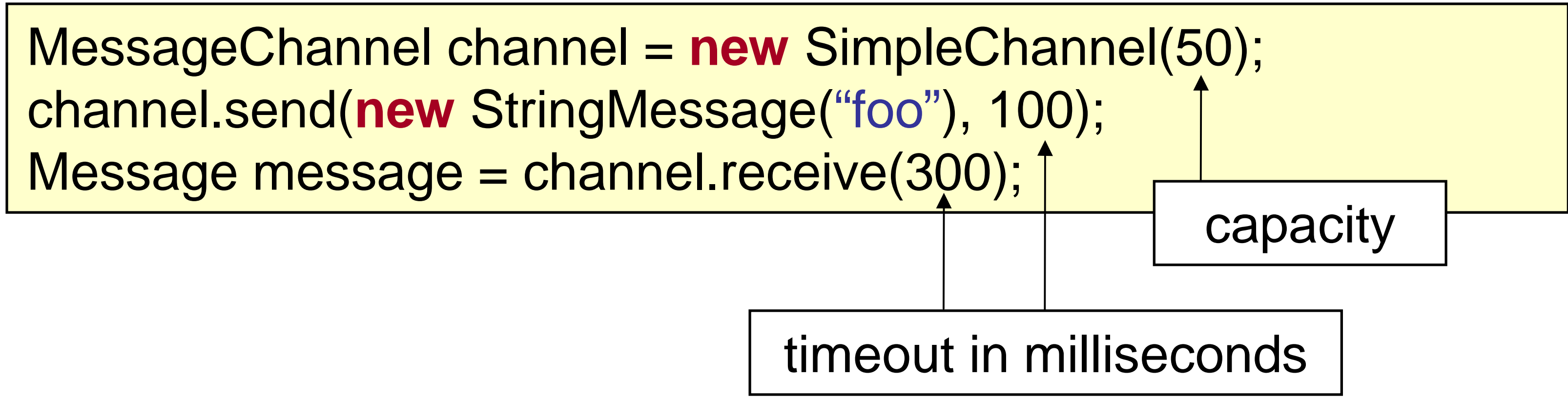
```
public interface MessageChannel {
    boolean send(Message message);
    boolean send(Message message, long timeout);
    Message receive();
    Message receive(long timeout);
}
```

- If capacity is reached, the send method will block until a Message is removed or the timeout elapses
- If empty, the receive method will block until a Message is available or the timeout elapses

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

SimpleChannel

- Default implementation
- Wraps a BlockingQueue



Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessageHandler

- A generic interface defines the simple but common behavior of processing a received Message

```
public interface MessageHandler {
    Message handle(Message message);
}
```

- Many of the internal base messaging components implement this top-level interface
  - Routers, Transformers, Service Invokers
- Implementations do not necessarily return a reply Message (routers, void-returning service invokers)

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessageHandlerChain

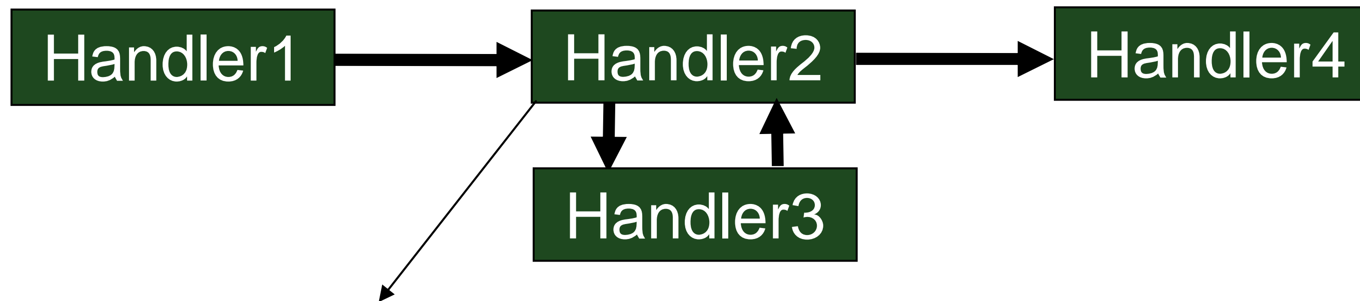
- MessageHandlers can be linked together

```
MessageHandlerChain chain = new MessageHandlerChain();
chain.add(new Handler1());
chain.add(new Handler2());
chain.add(new Handler3());
Message result = chain.handle(new StringMessage("foo"));
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

InterceptingMessageHandler

- An InterceptingMessageHandler can add behavior before and/or after another handler
  - The intercepting handler is responsible for calling *handle* on the next handler (or intentionally not proceeding)




```
public Message handle(Message message, MessageHandler target) {
   // do something before
   message = target.handle(message);
   // do something after
   return message;
}
```

Similar to AOP *around* advice

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessageDispatcher

- Receives from a channel and sends to one or more handlers

```
MessageDispatcher dispatcher =
                              new DefaultMessageDispatcher(channel);
dispatcher.addHandler(new ExampleMessageHandler());
dispatcher.addHandler(new AnotherMessageHandler(), schedule);
dispatcher.start();
```

- Configurable properties of the dispatcher:
  - defaultSchedule (initialDelay, period, fixedRate)
  - receiveTimeout
  - maxMessagesPerTask
  - rejectionLimit
  - retryInterval

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessageBus

- Acts as a registry for
  - MessageChannels
  - MessageEndpoints (handler + policies)
- Provides task execution infrastructure
  - For scheduling MessageDispatchers
  - For MessageEndpoint thread pools
- Manages lifecycle of the registered components (implements Lifecycle itself)

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- The Spring Integration Core API
- **Configuration Options**
- Channel Adapters
- Message Routing
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Configuration Options

- Integration components may be configured in a number of ways
  - Java with direct usage of the API
  - XML with generic 'beans' elements
  - XML with XSD namespace support
  - Annotations
  - AOP

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Java Configuration

```
MessageHandler handler = new ExampleHandler();

DefaultMessageEndpoint endpoint = new DefaultMessageEndpoint();
endpoint.setSubscription(new Subscription("inputChannel"));
endpoint.setDefaultOutputChannelName("outputChannel");
endpoint.setHandler(handler);

MessageBus bus = new MessageBus();
bus.registerChannel("inputChannel", new SimpleChannel());
bus.registerEndpoint("testEndpoint", endpoint);
bus.registerChannel("outputChannel", new SimpleChannel());
bus.start();
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

XML Configuration

```
<bean id="inputChannel"
class="org.springframework.integration.channel.SimpleChannel"/>

<bean id="endpoint"
      class="org.springframework.integration.endpoint.
             DefaultMessageEndpoint">
   <property name="subscription">
      <bean class="org.springframework.integration.
                   scheduling.Subscription"
         <constructor-arg ref="inputChannel"/>
      </bean>
   <property name="defaultOutputChannelName"
             value="outputChannel"/>
   <property name="handler" ref="exampleHandler"/>
</bean>

<bean id="bus"
   class="org.springframework.integration.bus.MessageBus"/>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Schema-Based Configuration

```
<integration:message-bus/>

<integration:channel id="quotes"/>

<integration:endpoint input-channel="quotes"
                      handler-ref="logger"
                      handler-method="log">
   <integration:schedule period="1000"/>
   <integration:concurrency core="5" max="20"/>
</integration:endpoint>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Annotation Configuration

```
@MessageEndpoint(input="inputChannel",
                 defaultOutput="outputChannel")
public class SimpleAnnotatedEndpoint {

    @Handler
    public String sayHello(String name) {
        return "hello " + name;
    }

}
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Messaging and AOP

- Spring AOP provides a non-invasive way to capture method execution “events”
  - Before advice
  - After-Returning advice
  - After-Throwing advice
- Spring Integration includes two interceptors
  - MessagePublishingInterceptor
  - AnnotationAwareMessagePublishingInterceptor

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

MessagePublishingInterceptor

```
MessagePublishingInterceptor interceptor =
                                        new MessagePublishingInterceptor();
interceptor.setDefaultChannel(testChannel);
ProxyFactory pf = new ProxyFactory(testService);
pf.addAdvice(interceptor);
TestService proxiedService = (TestService) pf.getProxy();
String result = proxiedService.test();
```

The return value is also sent to the channel

- The interceptor can be customized or extended
  - Provide a MessageMapper strategy
  - Subclass and implement a channel resolving strategy

```
resolveChannel(MethodInvocation invocation)
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Annotation-Driven Publisher

spring source

- Add the @Publisher annotation to any method of a Spring-managed object

```
@Publisher(channel="testChannel")
public String test() {
    return "testing…";
}
```

- Register a bean post-processor

```
<bean class="org.springframework.integration.config.
                  PublisherAnnotationPostProcessor"/>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Annotation-Driven Subscriber

- Add the @Subscriber annotation to any method of a Spring-managed object

```
@Subscriber(channel="testChannel")
public void test(String input) {
    System.out.println("received: " + input);
}
```

- Register a bean post-processor

```
<bean class="org.springframework.integration.config.
                SubscriberAnnotationPostProcessor"/>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- The Spring Integration Core API
- Configuration Options
- **Channel Adapters**
- Message Routing
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Channel Adapters

- A source adapter connects an external data *provider* to a MessageChannel
- A target adapter connects an external data *consumer* to a MessageChannel
- Examples:
  - JMS, File, Stream, RMI, ApplicationEvent

- In both cases, a message mapping strategy handles the conversion to and from message types

Similar to Spring's JMS MessageConverter

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

JMS Channel Adapters

```
<si:message-bus/>
<si:channel id="inputChannel"/>
<si:channel id="outputChannel"/>

<si:jms-source id="jmsSourceAdapter"
        connection-factory="connectionFactory"
        destination-name="sourceQueue"
        channel="inputChannel"/>

<si:jms-target id="jmsTargetAdapter"
        connection-factory="connectionFactory"
        destination-name="targetQueue"
        channel="outputChannel"/>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

File Channel Adapters

```
<si:message-bus/>
<si:channel id="inputChannel"/>
<si:channel id="outputChannel"/>

<si:file-source id="fileSourceAdapter"
        directory="${java.io.tmpdir}/test-input"
        channel="inputChannel"
        poll-period="10000"/>

<si:file-target id="fileTargetAdapter"
        directory="${java.io.tmpdir}/test-output"
        channel="outputChannel"/>
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Stream Channel Adapters

```
CharacterStreamSourceAdapter inputAdapter =
        new CharacterStreamSourceAdapter(reader);
adapter.setChannel(inputChannel);

CharacterStreamTargetAdapter outputAdapter =
        new CharacterStreamTargetAdapter(writer);

messageBus.registerSourceAdapter("inputAdapter", inputAdapter);
messageBus.registerHandler("outputAdapter", outputAdapter,
                                        new Subscription(outputChannel));
```

Also: ByteStreamSourceAdapter and ByteStreamTargetAdapter

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

ApplicationEvent Adapters

```
ApplicationEventSourceAdapter adapter =
                              new ApplicationEventSourceAdapter();

eventTypes = new ArrayList<Class<? extends ApplicationEvent>>();
eventTypes.add(TestApplicationEvent2.class);
adapter.setEventTypes(eventTypes);
adapter.setChannel(channel);

adapter.onApplicationEvent(new TestApplicationEvent1());
adapter.onApplicationEvent(new TestApplicationEvent2());
message = channel.receive(); // will receive TestApplicationEvent2
```

ApplicationEventTargetAdapter sends message payloads as ApplicationEvents – it implements ApplicationEventPublisherAware

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Delegating Source Adapters

```
MethodInvokingSource source = new MethodInvokingSource();
source.setObject(new ExampleSource());
source.setMethod("retrieve");

PollingSourceAdapter adapter = new PollingSourceAdapter(source);
adapter.setChannel(channel);
adapter.setPeriod(100);
adapter.start();

Message message = channel.receive();
```

- SourceAdapters are registered with the MessageBus which handles scheduling and lifecycle

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Delegating Target Adapters

```
MethodInvokingTarget target = new MethodInvokingTarget();
target.setObject(new ExampleTarget());
target.setMethod("publish");

DefaultTargetAdapter adapter = new DefaultTargetAdapter(target);
bus.registerHandler("adapter", adapter, new Subscription(channel));

channel.send(new StringMessage("foo"));
```

- TargetAdapters are also registered with the MessageBus
  - Handles lifecycle and schedules a dispatcher

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Annotation-Based Channel Adapters

```
@MessageEndpoint
public class SampleAnnotatedEndpoint {
    @Polled(period=5000)
    public String getName() {
        return "World";
    }
    @Handler
    public String sayHello(String name) {
        return "Hello " + name;
    }

    @DefaultOutput
    public void display(String message) {
        System.out.println(message);
    }
}
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- The Spring Integration Core API
- Configuration Options
- Channel Adapters
- **Message Routing**
- Roadmap

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Message Routing

spring source

- Routing can be simple
  - Resolve and send to a single channel based on the Message's payload type
  - Resolve and send to multiple channels based on a property value in the MessageHeader
- Or complex
  - Split a composite Message into its constituent parts and then send each as a new Message to a dedicated handler
  - In a downstream endpoint, aggregate the results into a single composite Message

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Message Routing Interfaces

- The base router implements MessageHandler and delegates to these fine-grained strategies

```
public interface ChannelResolver {
    MessageChannel resolve(Message message);
}
```

```
public interface MultiChannelResolver {
    List<MessageChannel> resolve(Message message);
}
```

```
public interface ChannelNameResolver {
    String resolve(Message message);
}
```

```
public interface MultiChannelNameResolver {
    String[] resolve(Message message);
}
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

PayloadTypeRouter

```
channelMappings.put(String.class, stringChannel);
channelMappings.put(Integer.class, integerChannel);

PayloadTypeRouter router = new PayloadTypeRouter();
router.setChannelMappings(channelMappings);
Message<String> message1 = new StringMessage("test");
Message<Integer> message2 = new GenericMessage<Integer>(123);

router.handle(message1); // will send to 'stringChannel'
router.handle(message2); // will send to 'integerChannel'
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

RecipientListRouter

```
List<MessageChannel> channels = new ArrayList<MessageChannel>();
channels.add(channel1);
channels.add(channel2);

RecipientListRouter router = new RecipientListRouter();
router.setChannels(channels);
Message<String> message = new StringMessage("test");

router.handle(message); // will send to channel1 and channel2
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Routing Annotation Examples

- Routers can also be defined with annotations
  - Low-level (working with message and channel)

```
@Router
public List<MessageChannel> getChannels(Message message) {
   // resolve the channel or channels…
}
```

  - High-level (working with domain objects)

```
@Splitter
public OrderItem[] splitOrders(PurchaseOrder purchaseOrder) {
   // split the purchase order into order items…
}
```

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Topics

- Goals and Principles of Spring Integration
- Enterprise Integration Patterns Overview
- The Spring Integration Core API
- Configuration Options
- Channel Adapters
- Message Routing
- **Roadmap**

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

1.0 Roadmap (1)

- Configuration
  - Generic XML ‘beans’
  - XSD-based namespaces
  - Annotations
- Channels
  - Point-to-Point
  - Publish/Subscribe
  - Support for datatype enforcement and priority
  - Interceptors for send and receive
- Adapters
  - JMS, RMI, HttpInvoker, Hessian/Burlap, and Spring-WS
  - File, FTP, Stream, and Email

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

1.0 Roadmap (2)

- Spring Core Integration
  - AOP
    - intercept and publish a Message before or after method
    - subscribe to a MessageChannel for method input
  - Adapters for Spring ApplicationEvents
  - Transaction management for Message Endpoints
- Spring Portfolio Integration
  - Source/Target Adapters for Spring Web Services
  - Messaging Gateway for Spring MVC
  - Integration with Spring Batch processes
  - Spring Security for Channels and Endpoints

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

Suggested Reading

- Enterprise Integration Patterns
  - Gregor Hohpe and Bobby Woolf (Addison Wesley, 2004)
- Pattern-Oriented Software Architecture, v.4
  - Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt (Wiley, 2007)
- Event-Based Programming
  - Ted Faison (Apress, 2006)
- Java Messaging
  - Eric Bruno (Charles River Media, 2006)
- Enterprise Service Bus
  - David Chappell (O'Reilly, 2004)

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.

DEMO

Copyright 2008 SpringSource. Copying, publishing or distributing without express written permission is prohibited.