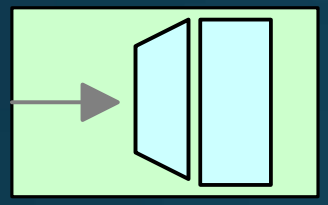


Gateways

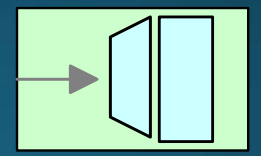
Express Spring Integration

Service Activator Review

- A message endpoint for connecting a Spring object or bean to a message channel.
  - The object or bean serves as a service.
  - The service is triggered (or activated) by the arrival of a message into the channel.
- The service activator configuration must specify the message channel that it polls for messages, and the class of the service bean.
  - If a service produces results, the output is sent to an output channel.
- The argument to a service method could be either a Message or an arbitrary type.
  - When the service method has no arguments, the service activator is referred to as ***event-style Service Activator***.

SI Gateway

- A Spring Integration gateway serves as a façade to a Spring Integration system.
  - It hides the SI API or any messaging API from the application.
  - The gateway is defined by an interface.
  - SI implements the gateway interface under the covers.
- There are two types of gateways.
  - Synchronous gateways cause the application to block while the SI system processes a request.
  - Asynchronous gateways allows the application to continue and retrieve results from the SI processes later.
- In EIP diagrams, gateways are represented by the following icon:



The Interface

- Applications must provide an interface to make requests of the SI system.
  - The interface should be devoid of SI API to keep the application decoupled from SI.
  - Under the covers, SI will implement the interface with a org.springframework.integration.gateway.GatewayProxyFactoryBean .

```
public interface ShipService {
        Confirmation ship(Order order);
}
```


Gateway Configuration & Use

- Spring Integration provides the implementation of the gateway based on its configuration.

```
<int:gateway id="shipService"
 service-interface="com.intertech.ShipService"
      default-request-channel="requestChannel" default-reply-channel="replyChannel" />
```

- The application can then call on the gateway as it would any other Spring bean.

```
ClassPathXmlApplicationContext context = new
   ClassPathXmlApplicationContext("/META-INF/spring/si-components.xml");
ShipService service = context.getBean("shipService", ShipService.class);
Confirmation confirm = service.ship(myOrder);
```


Asynchronous Gateway

- The previous example demonstrated a synchronous gateway.
  - The application code would need to block until SI replies back through the call to ship().
  - Gateways can be made to be synchronous or asynchronous.
- The gateway interface should be altered to return an instance of java.util.concurrent.Future<?>.

```
public interface ShipService {
        Future<Confirmation> ship(Order order);
}
```

  - The application code still remains loosely coupled from the SI API.
  - The application will need to get information from the Future at a point of its choosing by calling get( ).

```
ShipService service = context.getBean("shipService", ShipService.class);
Future<Confirmation> future = service.ship(myOrder);
// other application work here
Confirmation confirm = future.get(5000, TimeUnit.SECONDS);
```

You are ready to tackle Lab 8

- In Lab 8, you create a gateway to separate an application from a Spring Integration system that translates an English string to a Pig Latin string.
- You initially implement the gateway in a synchronous fashion, but change it to work asynchronously using a Java Future.

Thank you

- I hope you have enjoyed this Spring Integration Tutorial Series from ***Intertech***.
- If you found the series helpful, please let others know about it.
- Could you use some training in Spring or Spring MVC?
  - Contact Intertech's training organization and sign up for a class today!
  - http://www.intertech.com/Training/Java/Frameworks/Spring
- If you need some assistance on your Spring or Java project…
  - Check out Intertech's consulting services and request more information
  - http://www.intertech.com/Consulting/Spring-Framework-Consulting