
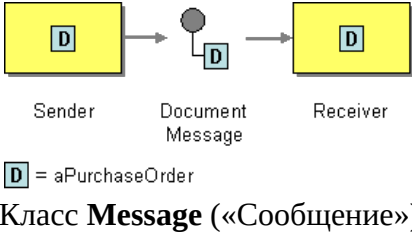

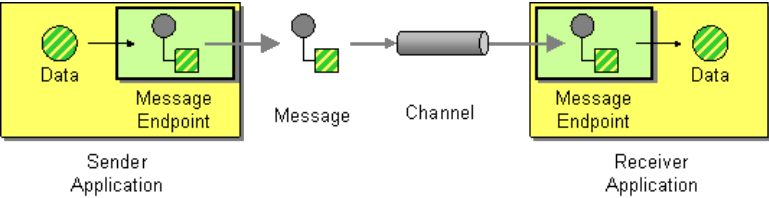
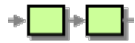


Основные шаблоны интеграции модуля **PyPipeline-ESB**

EIP	Схема шаблона интеграции	Пример кода
 <u>Message</u>	 Класс Message («Сообщение»)	<pre>class Message: def __init__(self): self.headers = {} self.body = None def __str__(self): return "\tHeaders: " + str(self.headers) + "\n\tBody: " + str(self.body)</pre>
<u>Message Exchange</u>	<p>Класс Exchange («Обмен»).</p> <p>Пакеты данных, которые передаются по шине, инкапсулированы в Exchange.</p> <p>Сущность «Обмен» имеет идентификатор (id), словарь свойств, и входящее (in) и исходящее (out) сообщения. Входящее и исходящее сообщения имеют тип Message</p>	<pre>import uuid class Exchange: def __init__(self): self.id = uuid.uuid4() self.in_msg = None self.out_msg = None self.properties = {} def __str__(self): return "Id: " + str(self.id) + "\nProperties: " + str(self.properties) + "\nIn Msg:\n" + str(self.in_msg) + "\nOut Msg:\n" + str(self.out_msg)</pre>
 <u>Message Endpoint</u>	 Компоненты Source («Источник») и Destination («Приёмник»). <p>Компоненты - это блоки, которые либо производят, обрабатывают или потребляют данные, которые передаются по шине.</p>	<pre>class Source: def __init__(self, plumber, params): self.plumber = plumber self.chain = None self.params = params def start(self): raise NotImplementedError("Should implement start method") def stop(self): raise NotImplementedError("Should implement start method") class Destination: def __init__(self, plumber, params): self.plumber = plumber self.params = params def process(self, exchange): raise NotImplementedError("Must implement process method")</pre>



Pipes & Filters



DslPipelineBuilder (конвейер/труба) представляет собой собственно шину интеграции.

Это последовательность компонентов, начиная с Источника и заканчивая произвольным количеством Приёмников, с набором EIP-функций обработчиков между ними.

```
builder = DslPipelineBuilder()

pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .to(MsgModifier) \
    .process(lambda ex: print(ex.in_msg.body)) \
    .build()
```

```
pipeline.start()
time.sleep(4)
pipeline.stop()
```

```
class MsgModifier(Destination):
    def process(self, exchange):
        exchange.in_msg.body += " modified"
```

This is exchange 0 modified
This is exchange 1 modified
This is exchange 2 modified
This is exchange 3 modified

Plumber

Plumber (Менеджер трубы). Может быть использован для регистрации нескольких конвейеров PipelineBuilders, сосредоточив таким образом, управление в единой точке.

Когда конвейер строится путем регистрации builder в менеджере «трубы», тогда этому конвейеру и всем его компонентам предоставляется экземпляр менеджера. Это можно использовать для выполнения сложных действий внутри компонента, таких как запуск другого конвейера, который зарегистрирован в менеджере (Plumber).

```
plumber = Plumber()
builder1 = DslPipelineBuilder()
builder2 = DslPipelineBuilder()
pipeline1 = builder1.source(Timer, {"period": 1.0}) \
    .to(MsgModifier).process(lambda ex: print(ex.in_msg.body))
pipeline2 = builder2.source(Timer, {"period": 2.0}) \
    .to(MsgModifier).process(lambda ex: print(ex.in_msg.body))
plumber.add_pipeline(pipeline1)
plumber.add_pipeline(pipeline2)
```

```
plumber.start()
time.sleep(5)
plumber.stop()
```

```
class MsgModifier(Destination):
    def process(self, exchange):
        exchange.in_msg.body += " modified"
```

This is exchange 0 modified
This is exchange 0 modified
This is exchange 1 modified
This is exchange 2 modified
This is exchange 1 modified
This is exchange 3 modified
This is exchange 4 modified
This is exchange 2 modified



Message Filter

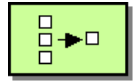
«Фильтр сообщений» (EIP-метод **filter**).

```
builder = DslPipelineBuilder()

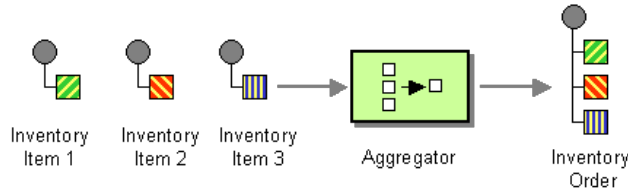
pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .filter(filterEven) \
    .process(lambda ex: print(ex.in_msg.body))
```

```
def filterEven(ex):
    counter = ex.in_msg.body.split()[3]
    return int(counter) % 2 == 0
```

This is exchange 0
This is exchange 2
This is exchange 4
This is exchange 6
This is exchange 8



Aggregator



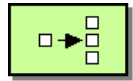
«Агрегатор» (EIP-метод **aggregate**). Собирает отдельные сообщения, пока не будет получен полный набор связанных сообщений. Публикует их как 1 исходящее.

```
builder = DslPipelineBuilder()

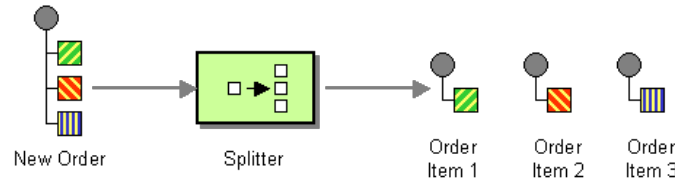
pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .aggregate({"method": aggregFn, "count": 5, "timeout": 2}) \
    .process(lambda ex: print(ex.in_msg.body))
```

```
def aggregFn(old, current):
    if old is not None:
        current.in_msg.body += old.in_msg.body
    return current
```

This is exchange 1
This is exchange 3
This is exchange 5
This is exchange 7
This is exchange 9



Splitter



EIP-метод **split** «расщепляет» одно сообщение на несколько составных.

```
builder = DslPipelineBuilder()

pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .split({"method": splitFn, "count": 5, "timeout": 2}) \
    .process(lambda ex: print(ex.in_msg.body))
```

```
def splitFn(exchange):
    exchanges = []
    for s in exchange.in_msg.body.split():
        ex = exchange.plumber.create_exchange()
        ex.in_msg = Message()
        ex.in_msg.body = s
        exchanges.append(ex)
    return exchanges
```

This is exchange 0
This is exchange 1
This is exchange 2
This is exchange 3

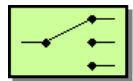
Multicast

EIP-метод **multicast** отправляет каждое сообщение Источника одновременно нескольким Приёмникам для отдельной обработки.

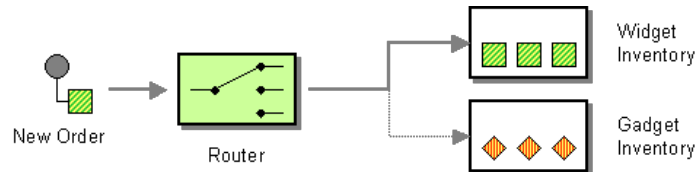
This is exchange 0 P11
THIS IS EXCHANGE 0 P12
This is exchange 0 P21
THIS IS EXCHANGE 1 P11
THIS IS EXCHANGE 1 P12
This is exchange 1 P21
THIS IS EXCHANGE 2 P11
THIS IS EXCHANGE 2 P12
This is exchange 2 P21
THIS IS EXCHANGE 3 P11
THIS IS EXCHANGE 3 P12
This is exchange 3 P21

```
builder = DslPipelineBuilder()

pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .multicast({}) \
    .pipeline().process(lambda ex: print(ex.in_msg.body + " P11")) \
    .process(to_upper) \
    .process(lambda ex: print(ex.in_msg.body + " P12")) \
    .end_pipeline() \
    .pipeline().process(lambda ex: print(ex.in_msg.body + " P21")) \
    .end_pipeline() \
    .end_multicast()
```



Content-Based Router

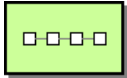
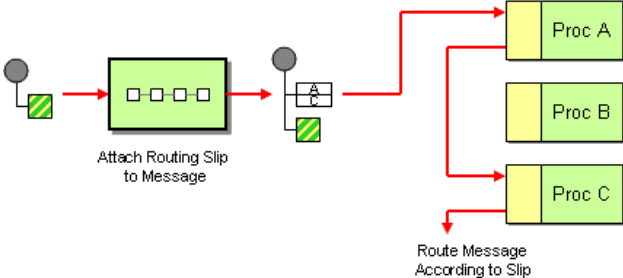

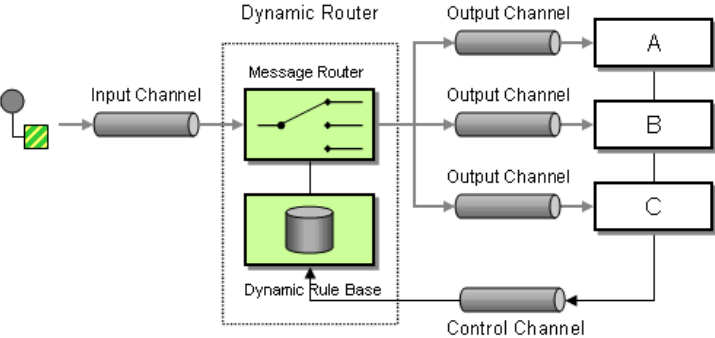


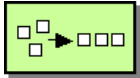
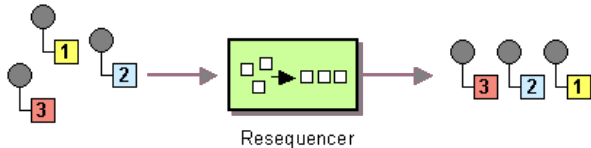

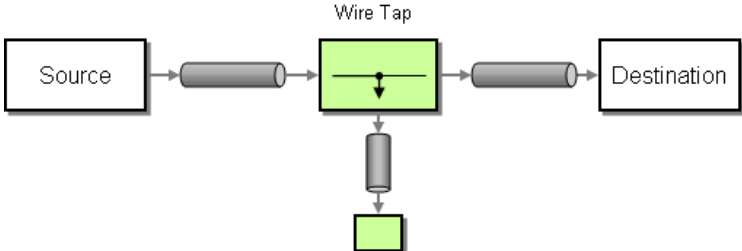
Маршрутизирует сообщение на основании его содержания.

This is exchange 0 %0
This is exchange 0 Last
This is exchange 1 %1
This is exchange 1 Last
This is exchange 2 Otherwise
This is exchange 2 Last
This is exchange 3 %0
This is exchange 3 Last

```
builder = DslPipelineBuilder()

pipeline = builder \
    .source(Timer, {"period": 1.0}) \
    .content_based_router() \
    .when(lambda ex: int(ex.in_msg.body.split()[-1]) % 3 == 0) \
    .pipeline().process(lambda ex: print(ex.in_msg.body + " %0")) \
    .end_pipeline() \
    .when(lambda ex: int(ex.in_msg.body.split()[-1]) % 3 == 1) \
    .pipeline().process(lambda ex: print(ex.in_msg.body + " %1")) \
    .end_pipeline() \
    .otherwise().pipeline() \
```

		<pre>.process(lambda ex: print(ex.in_msg.body + " Otherwise")) \ .end_pipeline() \ .end_content_based_router() \ .process(lambda ex: print(ex.in_msg.body + " Last"))</pre>
 <u>Routing Slip</u>	 <p>EIP-метод routing_slip добавляет к сообщению последовательность шагов обработки.</p>	<pre>builder = DslPipelineBuilder() pipeline = builder \ .source(Timer, {"period": 1.0}) \ .routing_slip({"method": slip}) \ .process(lambda ex: print(ex.in_msg.body)) def slip(exchange): return [(D1, {}), (D2, {})] class D1(Destination): def process(self, exchange): print(exchange.in_msg.body + " D1") class D2(Destination): def process(self, exchange): print(exchange.in_msg.body + " D2")</pre> <p>This is exchange 0 D1 This is exchange 0 D2 This is exchange 0 This is exchange 1 D1 This is exchange 1 D2 This is exchange 1 This is exchange 2 D1 This is exchange 2 D2 This is exchange 2 This is exchange 3 D1 This is exchange 3 D2 This is exchange 3</p>
 <u>Dynamic Router</u>	 <p>EIP-метод dynamic_router - это динамический маршрутизатор сообщений.</p>	<pre>builder = DslPipelineBuilder() pipeline = builder \ .source(Timer, {"period": 1.0}) \ .dynamic_router({"method": slipFn}) \ .process(lambda ex: print(ex.in_msg.body)) def slipFn(exchange): if Property.slip_endpoint not in exchange.properties: return D1, {} elif exchange.properties[Property.slip_endpoint] == D1: return D2, {} else: return None class D1(Destination): def process(self, exchange): print(exchange.in_msg.body + " D1") class D2(Destination): def process(self, exchange): print(exchange.in_msg.body + " D2")</pre> <p>This is exchange 0 D1 This is exchange 0 D2 This is exchange 0 This is exchange 1 D1 This is exchange 1 D2 This is exchange 1 This is exchange 2 D1 This is exchange 2 D2 This is exchange 2 This is exchange 3 D1 This is exchange 3 D2 This is exchange 3</p>

 Resequencer	 <p>ЕІР-метод resequencer производит переупорядочение сообщений.</p>	<pre>builder = DslPipelineBuilder() pipeline = builder \ .source(Timer, {"period": 1.0}) \ .process(insert_seq_number) \ .resequencer({"key_extractor": get_key, \ "count": 5, "reverse": True}) \ .process(lambda ex: print(ex.in_msg.headers["seq_num"])) plumber.add_pipeline(pipeline) plumber.start() time.sleep(5) plumber.stop() def insert_seq_number(exchange): exchange.in_msg.headers["seq_num"] = random.randint(0, 100) def get_key(ex): return ex.in_msg.headers["seq_num"]</pre>
Validate	<p>ЕІР-метод validate - валидатор сообщений.</p> <pre>This is exchange 0 Exception in thread Thread-233: raise ValueError("Exchange failed validation") ValueError: Exchange failed validation</pre>	<pre>builder = DslPipelineBuilder() pipeline = builder \ .source(Timer, {"period": 1.0}) \ .validate(filterEven) \ .process(lambda ex: print(ex.in_msg.body))</pre>
 Wire Tap	 <p>ЕІР-метод waretrap - «Прослушка». Сообщение Источника передаётся одновременно на Приёмник и дополнительный канал.</p>	<pre>from pypipeline.components.destination.Log import Log builder = DslPipelineBuilder() pipeline = builder \ .source(Timer, {"period": 1.0}) \ .wiretap((Log, {"name": "wiretap"})) \ .to(Log, {"name": "actual"}) plumber.add_pipeline(pipeline) plumber.start() time.sleep(0) plumber.stop()</pre> <pre>Log: wiretap Id: c12cd2fd-134a-4b7c-9ccf-2a89c5f18c3d Properties: {} In Msg: Headers: {} Body: This is exchange 0 Out Msg: None Log: actual Id: 3fcab9df-13bb-4ab5-a745-22c9168bf798 Properties: {} In Msg: Headers: {} Body: This is exchange 0 Out Msg: None</pre>