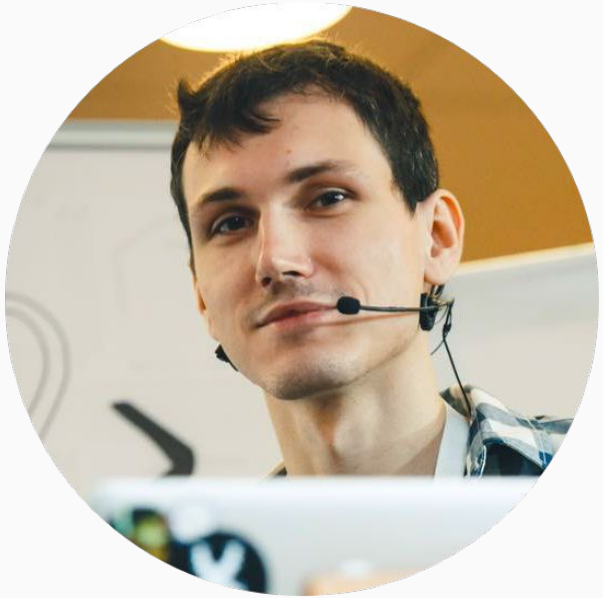
Твой личный Spring Boot Starter



@tolkv

octoberry

РАЗБОР ПОЛЕТОВ

@aatarasoff

DEVELOPERBLOG.INFO

Твой личный Spring Boot Starter

План такой

1. Будем решать задачу
2. Будут сложности
3. Будем бороться с ними
4. Сделаем выводы
5. Ответим на вопросы

Простая задача.
“Супергерой на час”

Попытка №1

К несчастью у нас целая куча сервисов

Слайд Боли

- Boilerplate-боль
  - бизнес-логика
  - куча непонятного, но нужного
- Конфигурация-боль
  - много настроек
  - сложная композировать настройки
- Имплементация-боль
  - не DRY

К чему это ведёт?

- Низкая скорость создания
- Высокий порог вхождения
- Нужно держать в голове детали

Spring Boot Starter

- Мини-фреймворк
- Высокий уровень абстракции
- Всё своё несёт собой

Принципы

- композиция через аннотации
- согласованность технологий
- изменение поведения через конфигурацию

Простой пример

```
@SpringBootApplication
public class Application {
     @PostConstruct
     public void init(){
         ..boilerplate...
          business logic
         ..boilerplate...
     }
     public static void main(String[] args) {
          SpringApplication.run(Application.class, args);
     }
}
```

Простой пример

```
@SpringBootApplication
@EnableThriftServer
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Добавим пару аннотаций

```
@EnableHystrix
@EnableHystrixDashboard
@SpringBootApplication
@EnableThriftServer
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Или не пару


```
@EnableZuulProxy
@EnableDiscoveryClient
@EnableHystrix
@EnableHystrixDashboard
@SpringBootApplication
@EnableThriftServer
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Пишем свой...

стартер

7 простых шагов

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

7 простых шагов

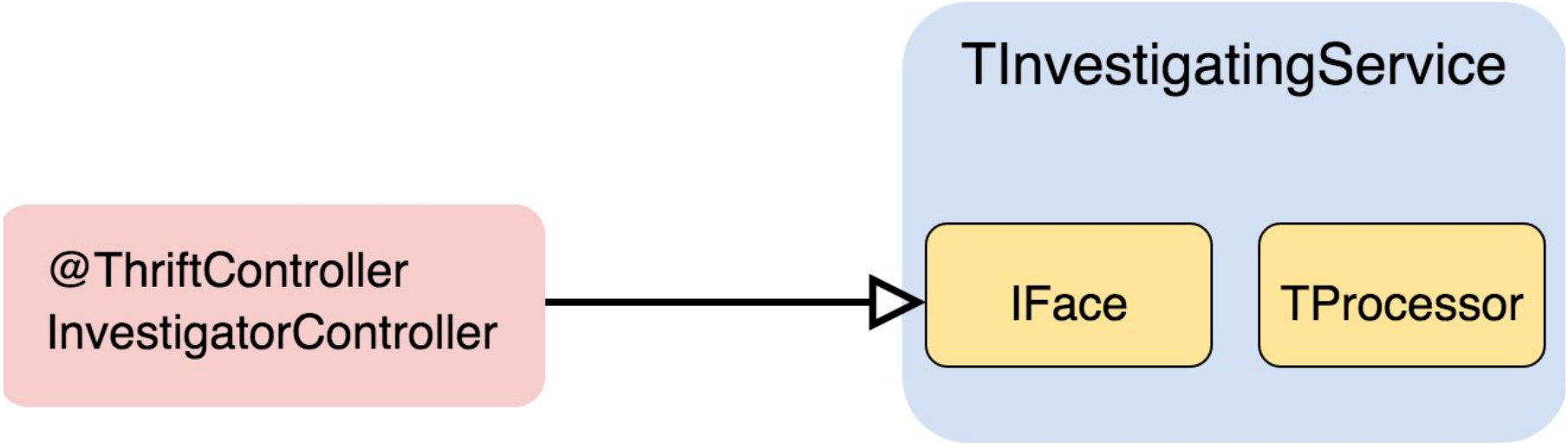
1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

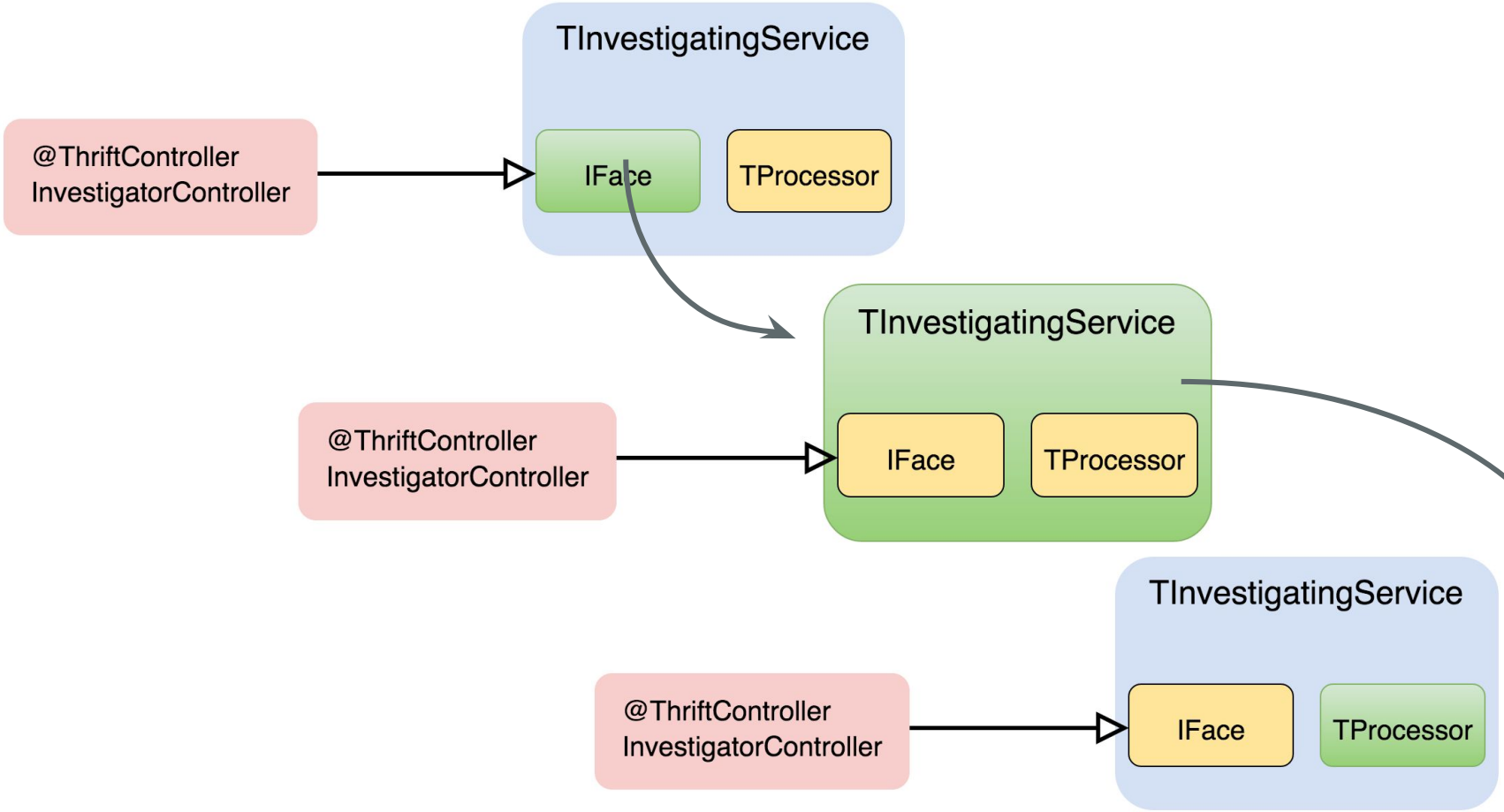
41.4 Creating your own starter

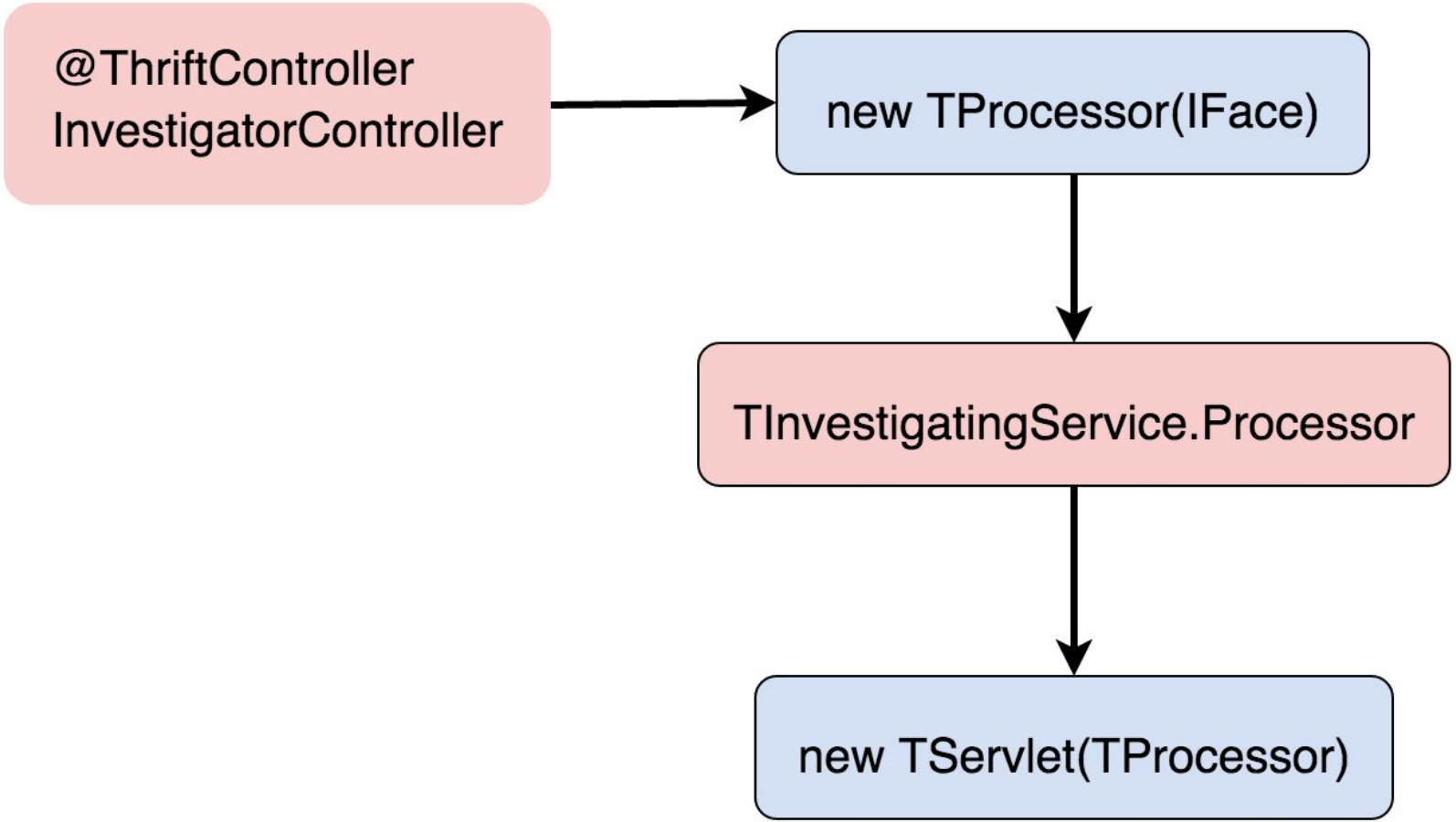
A full Spring Boot starter for a library may contain the following components:

- The `autoconfigure` module that contains the auto-configuration code.
- The `starter` module that provides a dependency to the autoconfigure module as well as the library and any additional dependencies that are typically useful. In a nutshell, adding the starter should be enough to start using that library.

Схема работы автоконфигуратора




TInvestigatingService
@ThriftController
InvestigatorController
IFace
TProcessor
TInvestigatingService
@ThriftController
InvestigatorController
IFace
TProcessor
TInvestigatingService
@ThriftController
InvestigatorController
IFace
TProcessor


@ThriftController
InvestigatorController
new TProcessor(IFace)
TInvestigatingService.Processor
new TServlet(TProcessor)

7 простых шагов

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

7 простых шагов

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

7 простых шагов

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

7 простых шагов

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

Зачем нужен spring.provides?

spring-boot / spring-boot-starters / spring-boot-starter-web / src / main / resources / META-INF / **spring.provides**

dsyer Add META-INF/spring.provides to starters 81

1 contributor

1 lines (1 sloc) | 51 Bytes **Raw** **Blame** **His**

```
1  provides: spring-webmvc,spring-web,jackson-databind
```

Зачем нужен spring.provides?

dsyer

It's for IDEs. They can make suggestions about imports for things that are not on the classpath yet by parsing and caching that data.

Что ещё нужно сделать

Добавим метрики, обработку ошибок и логирование:

```
proxyFactory.addAdvice(new MetricsThriftMethodInterceptor(gaugeService))
proxyFactory.addAdvice(new ExceptionsThriftMethodInterceptor())
proxyFactory.addAdvice(new LoggingThriftMethodInterceptor())
```

```
@Slf4j
public class LoggingThriftMethodInterceptor implements MethodBeforeAdvice, AfterReturningAdvice {
      @Override
      public void before(Method method, Object[] args, Object target) throws Throwable {
            log.info("Thrift method {}.{}() is called with args: {}",
                  target.getClass().getSimpleName(), method.getName(), args
            );
      }

      //afterReturning method is implemented here %)
}
```

Что ещё можно улучшить

```
@ThriftClient(serviceId = "greeting-service", path = "/api")
TGreetingService.Client client;
```

```
greeting-service:                          #service name
  endpoint: http://localhost:8080/api      #direct endpoint
  ribbon:                                  #ribbon
      listOfServers: localhost:8080        #hardcoded list
  path: /service                           #general path
  connectTimeout: 1000                     #default=1000
  readTimeout: 10000                       #default=30000
```

Как распространять?

- Не прячьте свой труд. Используйте GitHub.
- Делитесь своими разработками со всеми. Используйте Bintray и jcenter.

Парадокс (де)централизации микросервисов

Чтобы эффективно работать с **распределенными** приложениями, нужно иметь очень хорошие **централизованные** библиотеки/инструменты

Например: логирование, health-чеки, метрики,
обработку программных ошибок

Парадокс (де)централизации микросервисов

Чтобы эффективно работать с **распределенными** приложениями, нужно иметь очень хорошие **централизованные** библиотеки/инструменты

Но: не выносите бизнес-логику или доменные объекты

7 простых шагов к счастью

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был один
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

7 простых шагов к счастью

1. Напиши “простой” тест. Разберись в технологии.
2. Напиши микросервис так, как если бы он был
3. Создай автоконфигуратор и переведи сервис на него
4. Добавь точки расширения
5. Не забудь про конфигурацию по-умолчанию
6. Напиши тесты на автоконфигурацию
7. Создай и распространяй стартер

Ссылки

Полная версия: https://github.com/aatarasoff/spring-thrift-starter

Для grpc: https://github.com/lavcraft/grpc-spring-boot-starter

Спасибо! Готовы ответить на ваши вопросы

@tolkv

@aatarasoff