
СБЕР
Всех их соберем!
Мы писали DSL в Spring и
грабли ловили

Немного о себе

Зубенко
Евгений

Демо

Шестаков
Максим

Дисклеймер

Дисклеймер

1 Мы не можем давать официальных рекомендаций

Дисклеймер

1 Мы не можем давать официальных рекомендаций

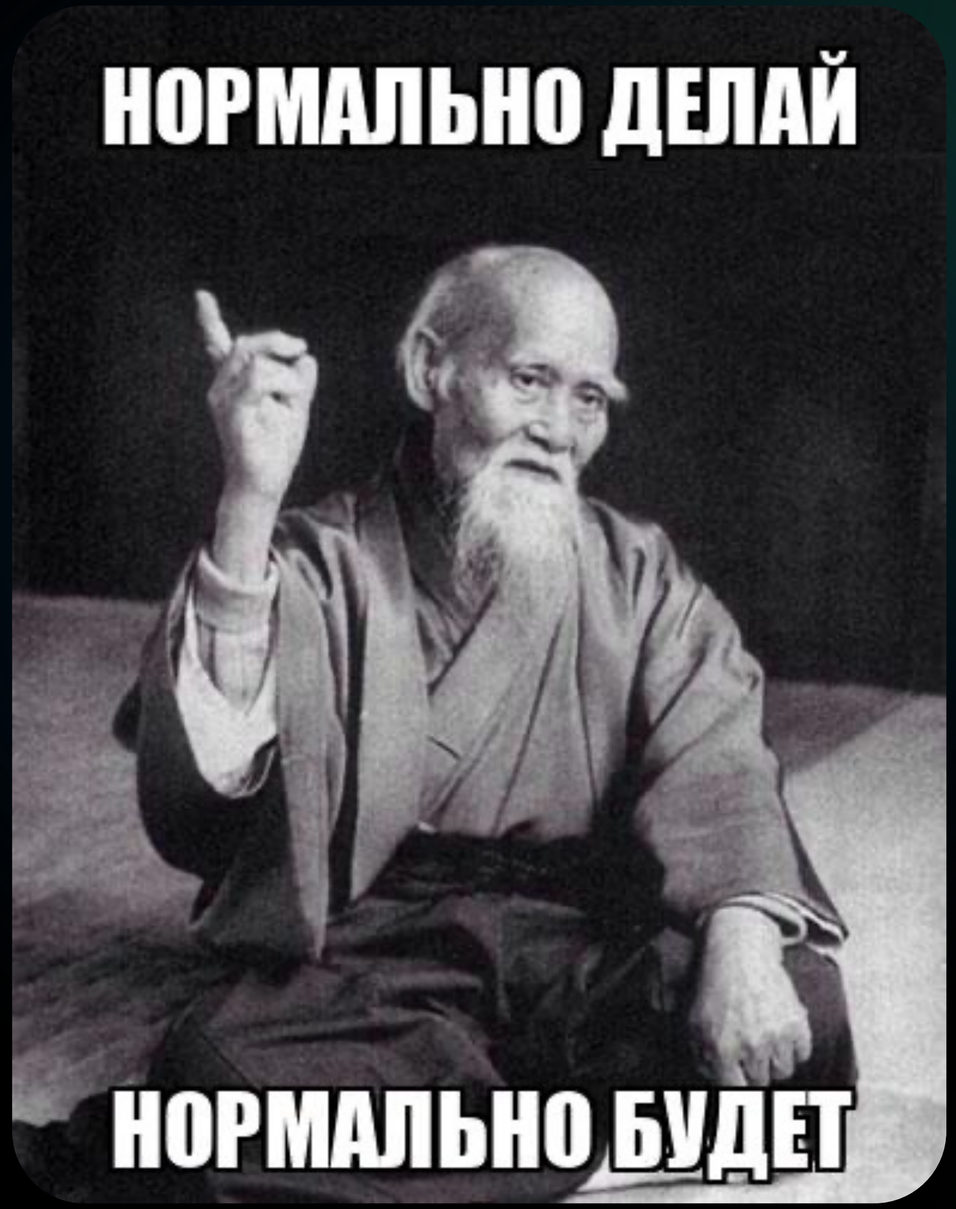
2 Однако, можем поделиться с вами нашим опытом

Дисклеймер

1 Мы не можем давать официальных рекомендаций

2 Однако, можем поделиться с вами нашим опытом

3 Мы не говорим, какое из решений правильное



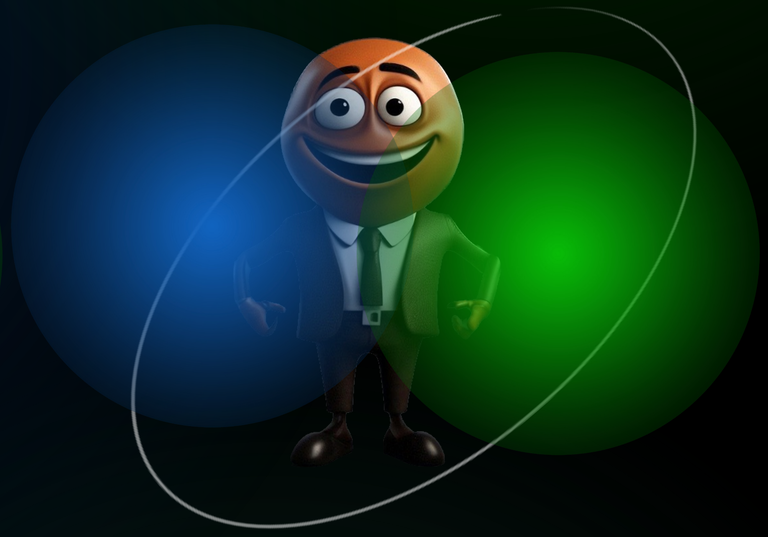
“Хочешь хорошо работать
– пользуйся Spring’ом.

Хочешь, чтобы работало хорошо
– знай, как он устроен внутри.”

Приступим

Старые добрые
XML бины

Java
configuration



Functional
beans

Старые добрые
XML бины

Java
configuration

Functional
beans








Постановка задачи

ARG

String_1
String_2
Int_3
Long_4
...

Кому интересно?

Кому интересно?

1

Тем, кто пишет

Spring-библиотеки, регистрирующей кучу environment-бинов

Кому интересно?

1

Тем, кто пишет

Spring-библиотеки, регистрирующей кучу environment-бинов

2

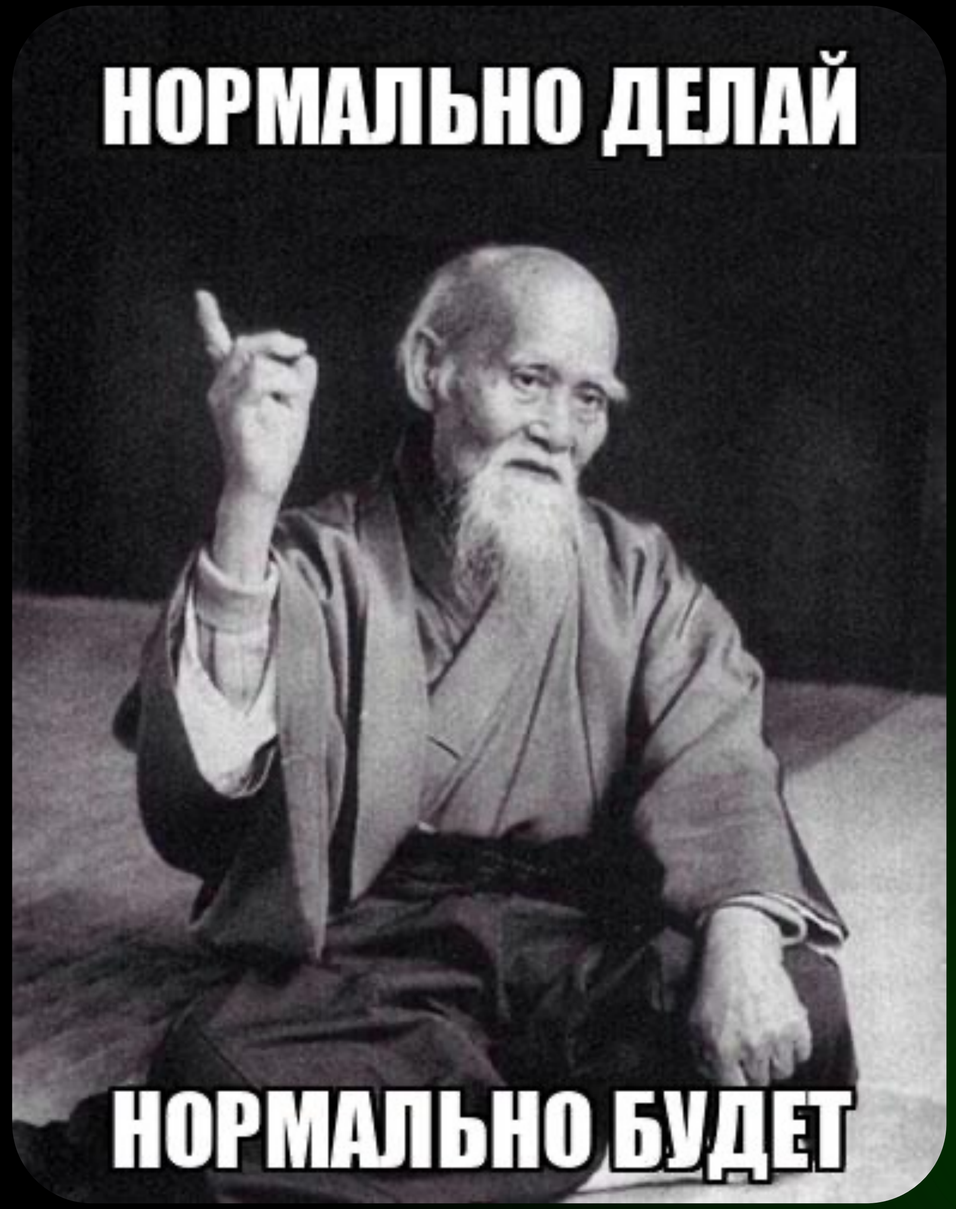
Тем, кто хочет

сделать конфигурацию своего приложения более лаконичной

Мечты


```
@Bean
public IntegrationFlow convert() {
      return f -> f
        .transform(payload ->
              "<FahrenheitToCelsius xmlns=\"https://www.w3schools.com/xml/\">"
            +     "<Fahrenheit>" + payload + "</Fahrenheit>"
            + "</FahrenheitToCelsius>")
        .enrichHeaders(h -> h
            .header(WebServiceHeaders.SOAP_ACTION,
                "https://www.w3schools.com/xml/FahrenheitToCelsius"))
        .handle(new SimpleWebServiceOutboundGateway(
            "https://www.w3schools.com/xml/tempconvert.asmx"))
        .transform(Transformers.xpath("/*[local-
name()=\"FahrenheitToCelsiusResponse\"]"
            + "/*[local-name()=\"FahrenheitToCelsiusResult\"]"));
 }
```

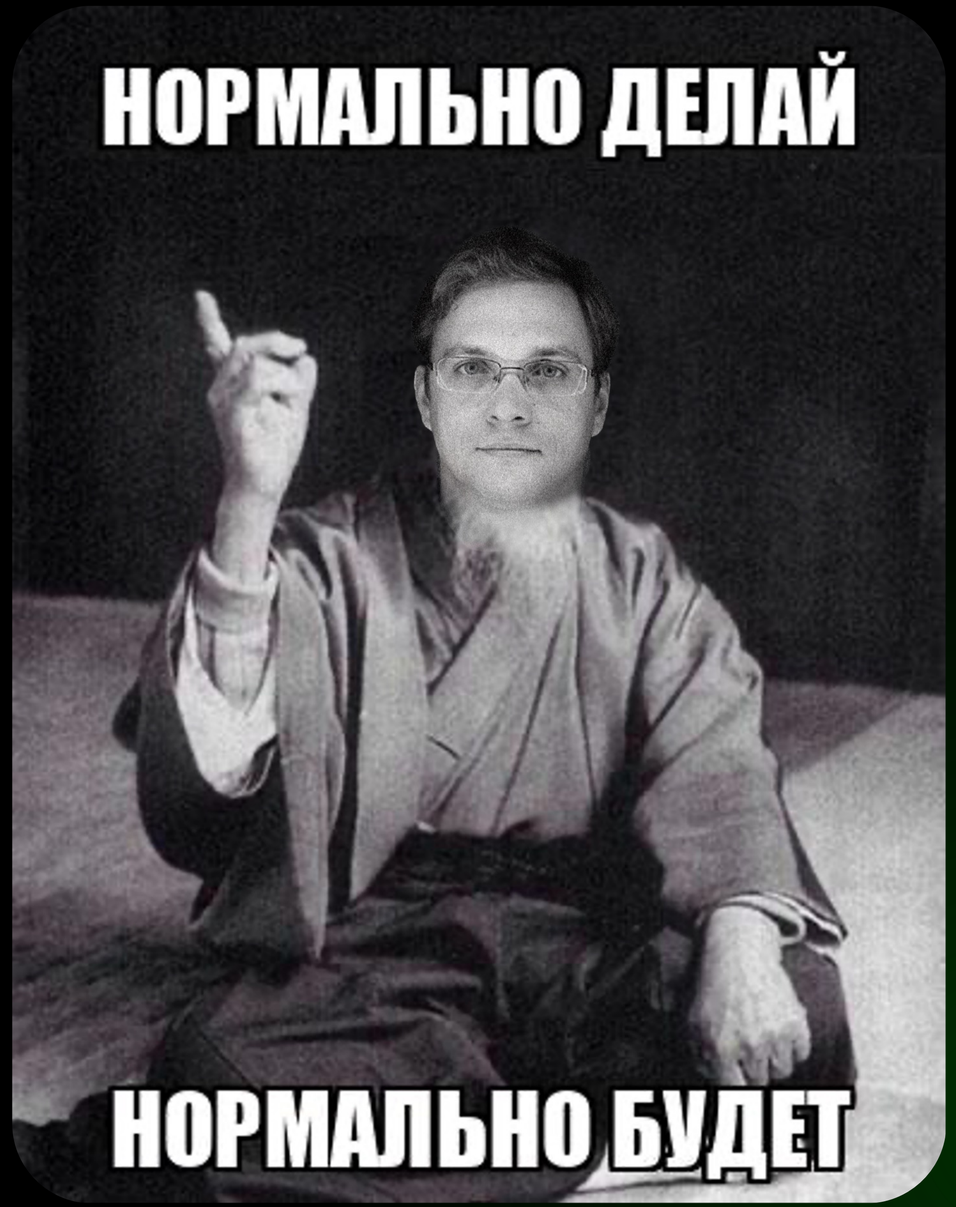

EnterpriseZoo



А помните?...

“Хочешь хорошо работать – пользуйся Spring’ом.

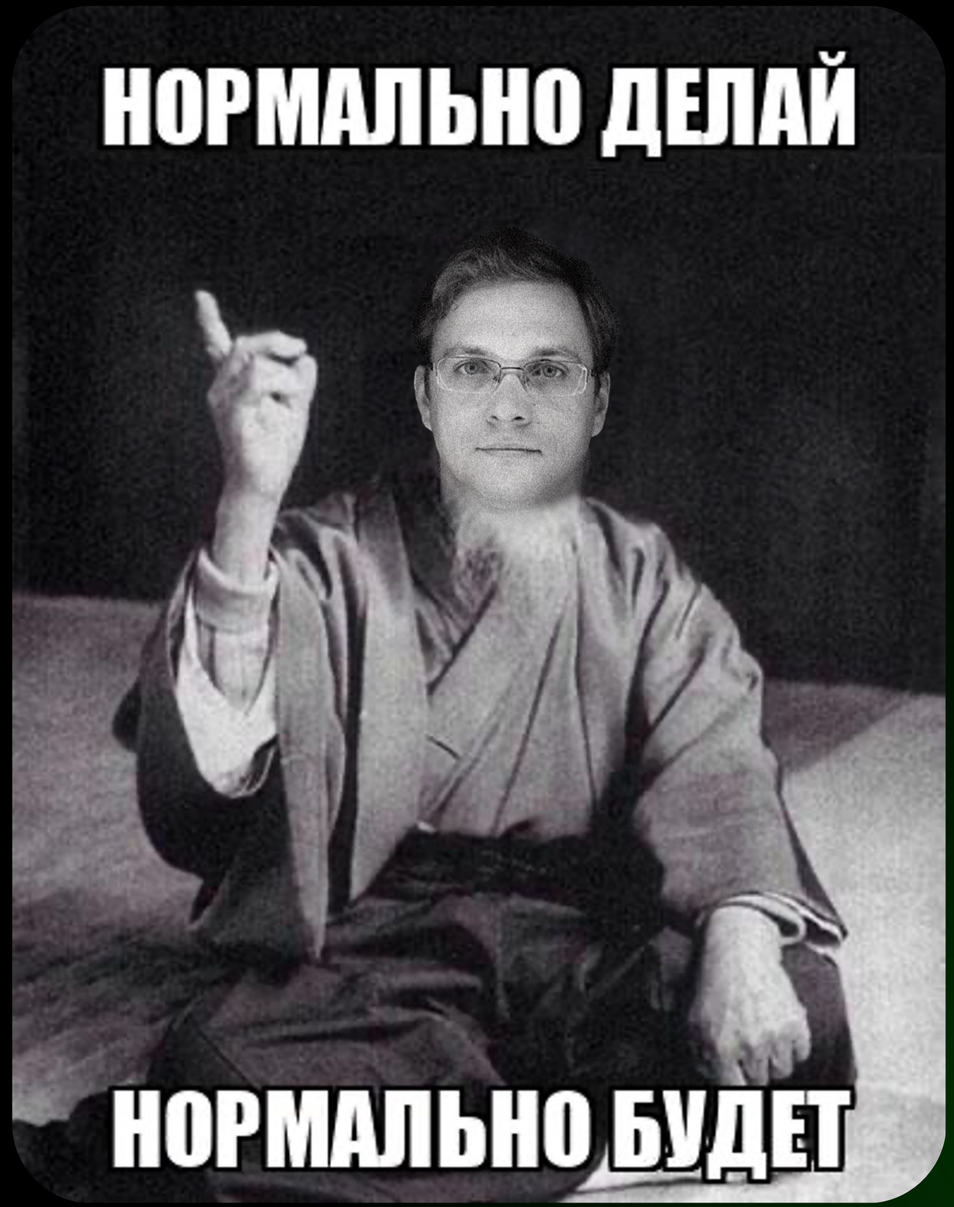
Хочешь, чтобы работало хорошо – знай, как он устроен внутри.”



А помните?...

“Хочешь хорошо работать – пользуйся Spring’ом.

Хочешь, чтобы работало хорошо – знай, как он устроен внутри.”



А помните?...

“Хочешь хорошо работать – пользуйся Spring’ом.

Хочешь, чтобы работало хорошо – знай, как он устроен внутри.”

Постановка задачи

```
DslBean.builder()
  .withString1("string1")
  .withString2("string2")
  .withInt3(3)
  .withLong4(4L)
  ...
  .build()
```

BeanFactoryPostProcessor

Позволяет

настраивать BeanFactory до создания неслужебных bean’ов
(в том числе и регистрировать новые bean’ы в ней)

С помощью них

регистрируются бины окружения в Spring-библиотеках

BeanFactoryPostProcessor

Позволяет

настраивать BeanFactory до создания неслужебных bean’ов
(в том числе и регистрировать новые bean’ы в ней)

С помощью них

регистрируются бины окружения в Spring-библиотеках


EnterpriseZoo

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья


РАННЯЯ
ИНИЦИАЛИЗАЦИЯ

Оказалось, не мы одни такие

GitHub

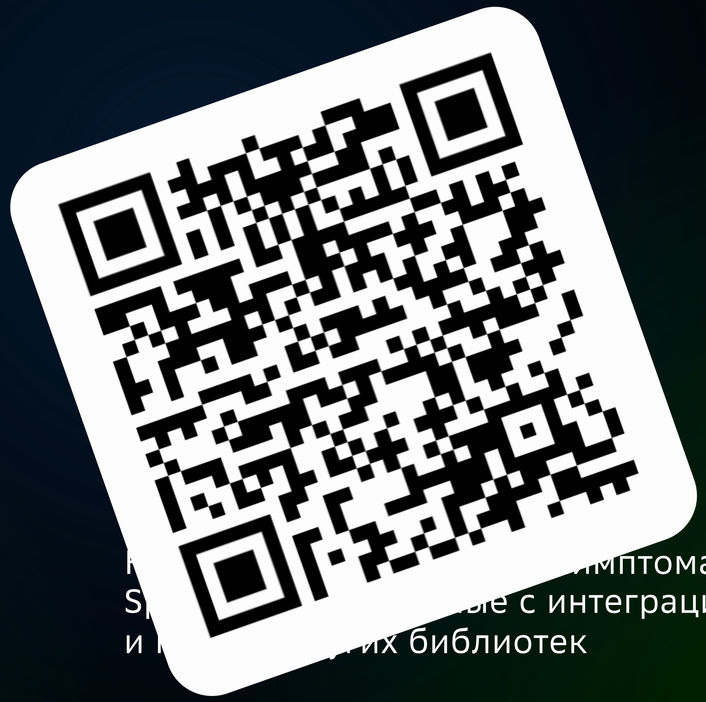
Нашли issue с похожими симптомами в репозитории SpringBoot (связанные с интеграцией с Mockito) и многих других библиотек

Оказалось, не мы одни такие

GitHub

Нашли issue с похожими симптомами в репозитории SpringBoot (связанные с интеграцией с Mockito) и многих других библиотек

Оказалось, не мы одни такие



Оказалось, не мы такие

имптомами в репозитории
ые с интеграцией с Mockito)
их библиотек

Оказалось, не мы такие

Получилось?

Получилось?

1

Использовали

BeanFactoryPostProcessor для создания нашего DSL на базе Spring

Получилось?

1

Использовали

BeanFactoryPostProcessor для создания нашего DSL на базе Spring

2

Столкнулись

с проблемами использования BeanFactory, о которых в сети ни слова

Получилось?

1

Использовали

BeanFactoryPostProcessor для создания нашего DSL на базе Spring

2

Столкнулись

с проблемами использования BeanFactory, о которых в сети ни слова

3

Надо ли делать именно так?

BeanFactoryPostProcessor, возможно, не лучший вариант

А как делают разработчики библиотек в Spring?

Встречайте!

ImportBeanDefinitionRegistrar

Встречайте!

ImportBeanDefinitionRegistrar

- **Регистрирует**

  бины во время старта контекста

Встречайте!

ImportBeanDefinitionRegistrar

- **Регистрирует**

  бины во время старта контекста

- **Импорта**

  достаточно для использования

Встречайте!

ImportBeanDefinitionRegistrar

- **Регистрирует**

  бины во время старта контекста

- **Импорта**

  достаточно для использования

- **Стартеры**

  используют именно этот подход


EnterpriseZoo

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor
- ImportBeanDefinitionRegistrar

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor
- ImportBeanDefinitionRegistrar

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Ну как теперь?

Ну как теперь?

1

Использовали

подход разработчиков стартеров,
ImportBeanDefiitionRegistrar

Ну как теперь?

1

Использовали

подход разработчиков стартеров, ImportBeanDefiitionRegistrar

2

Проблемы

все те же самые, что и с BeanFactoryPostProcessor

Ну как теперь?

1 **Использовали**

подход разработчиков стартеров, ImportBeanDefiitionRegistrar

2 **Проблемы**

все те же самые, что и с BeanFactoryPostProcessor

3 **Двойной импорт**

registrar’а приведет к его двойной работе

Ну как теперь?

1. **Использовали**

   подход разработчиков стартеров, ImportBeanDefinitionRegistrar

2. **Проблемы**

   все те же самые, что и с BeanFactoryPostProcessor

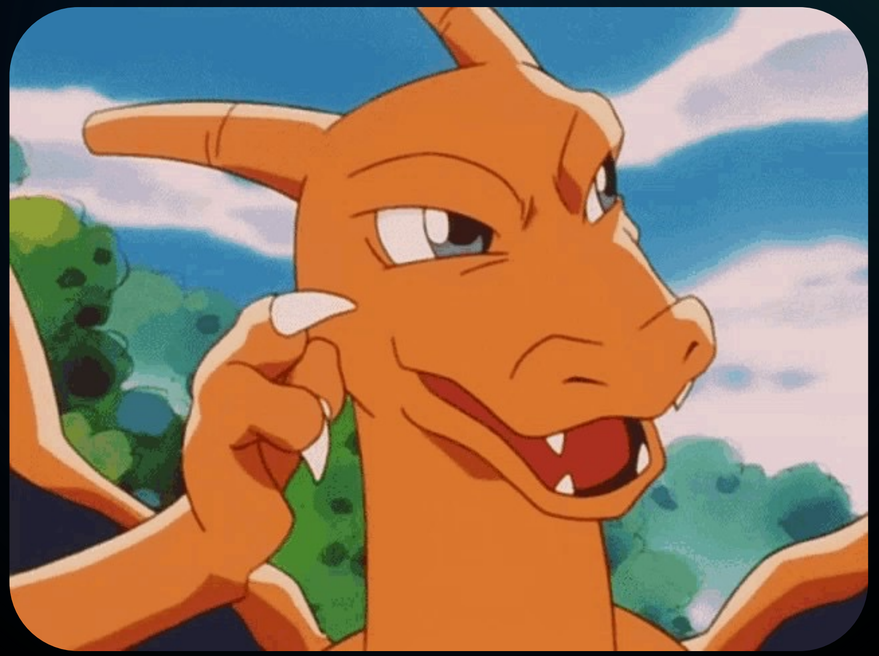
3. **Двойной импорт**

   registrar’а приведет к его двойной работе

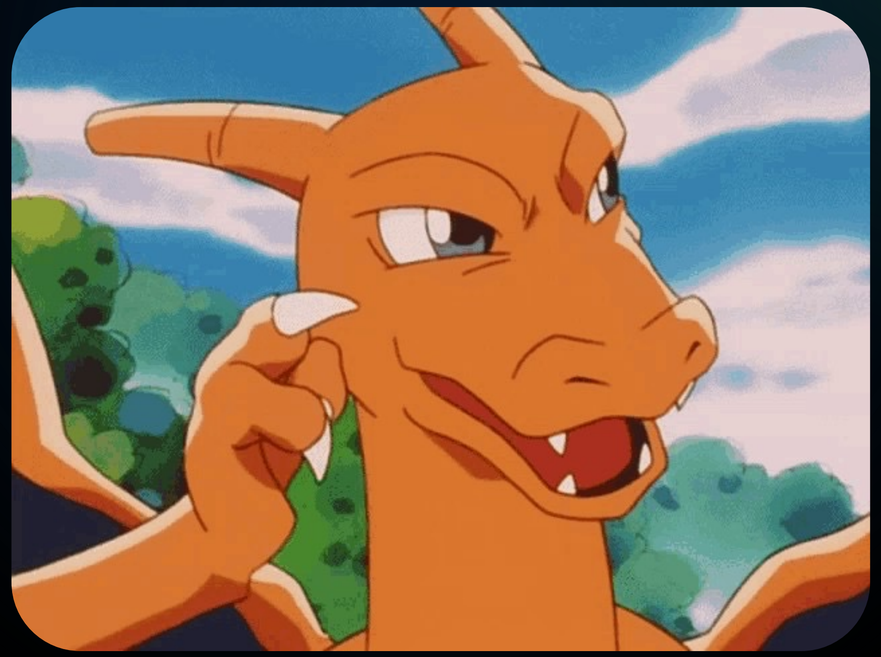
4. **Как-то слишком сложно**

   может можно как-то попроще?

Неужели все?

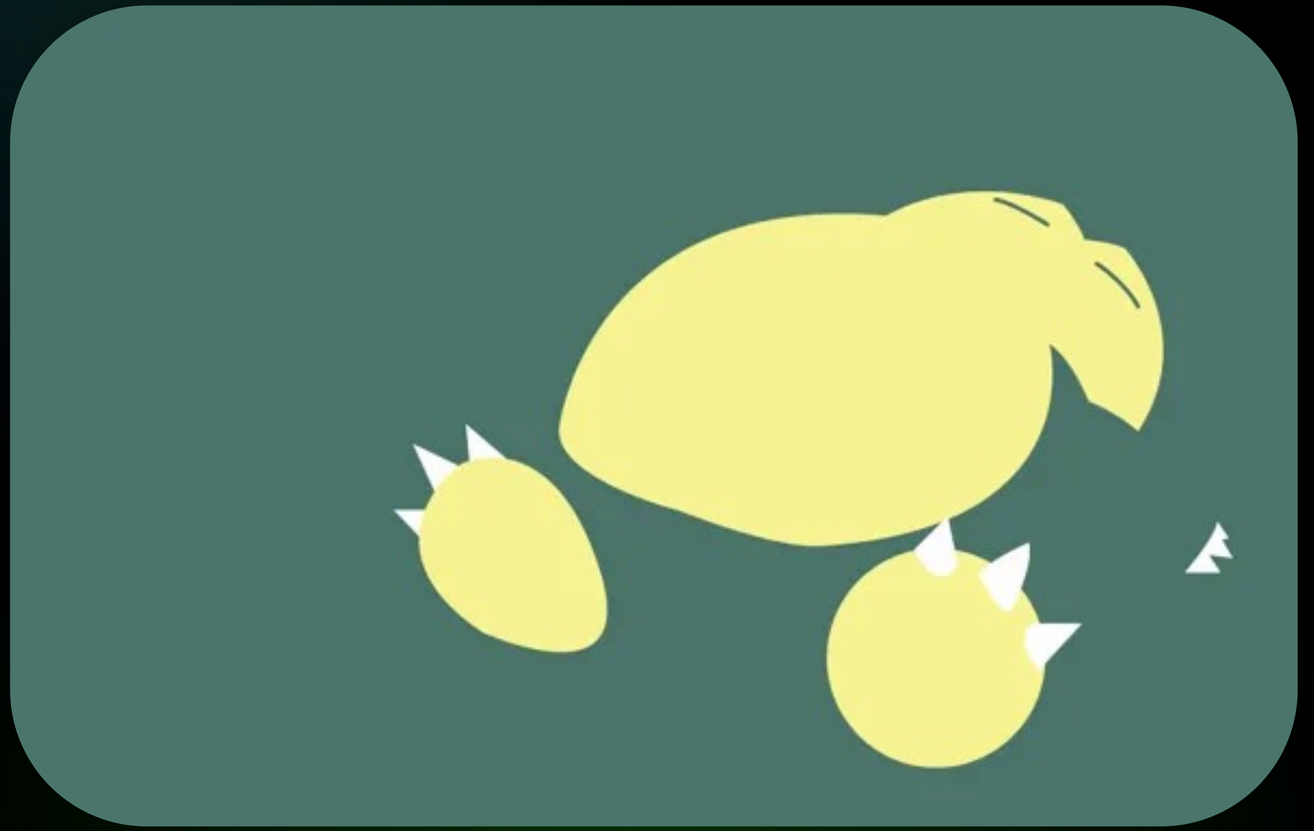


Неужели все?



Нет, не все!

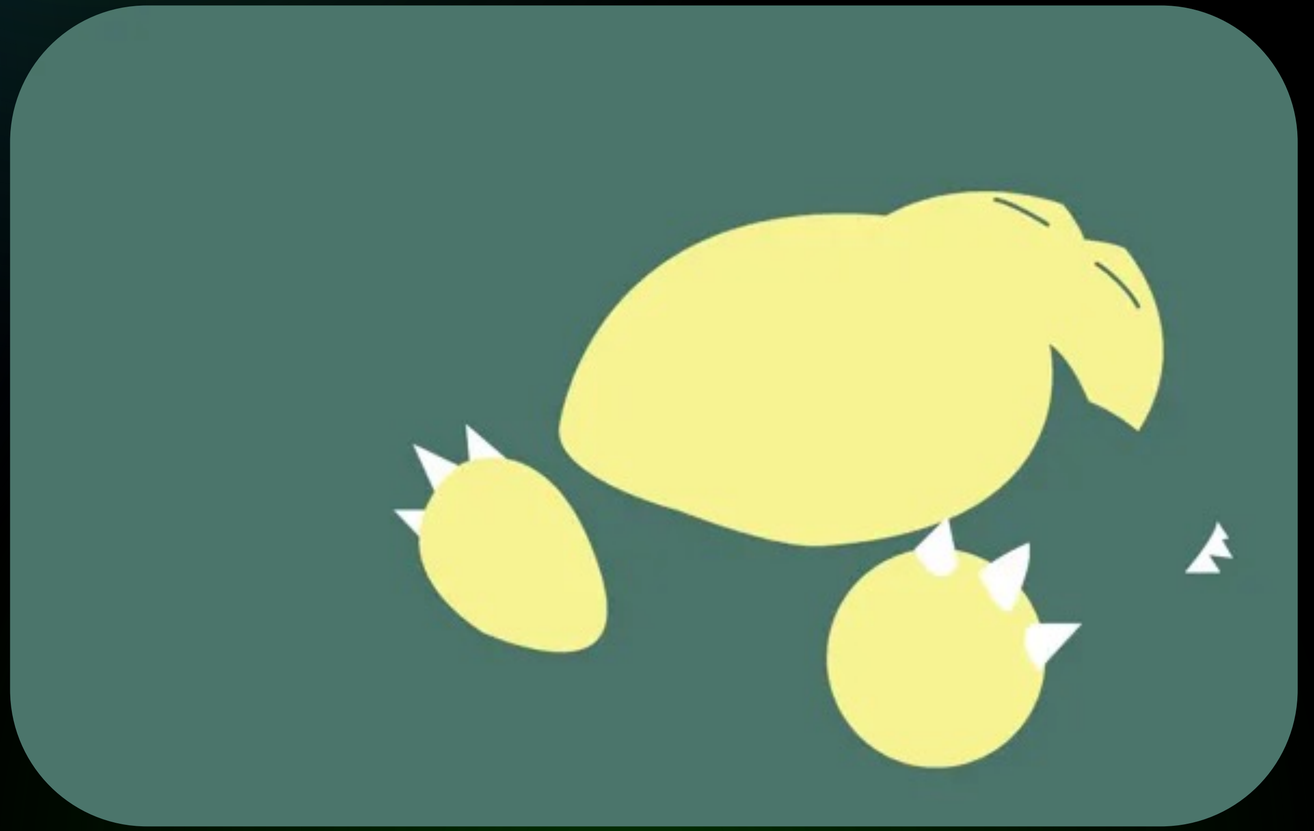
Делаем проще!
Сменим парадигму!



Делаем проще!
Сменим парадигму!

- **Отказываемся**

  от DSL-бинов в пользу утилитных DSL-методов



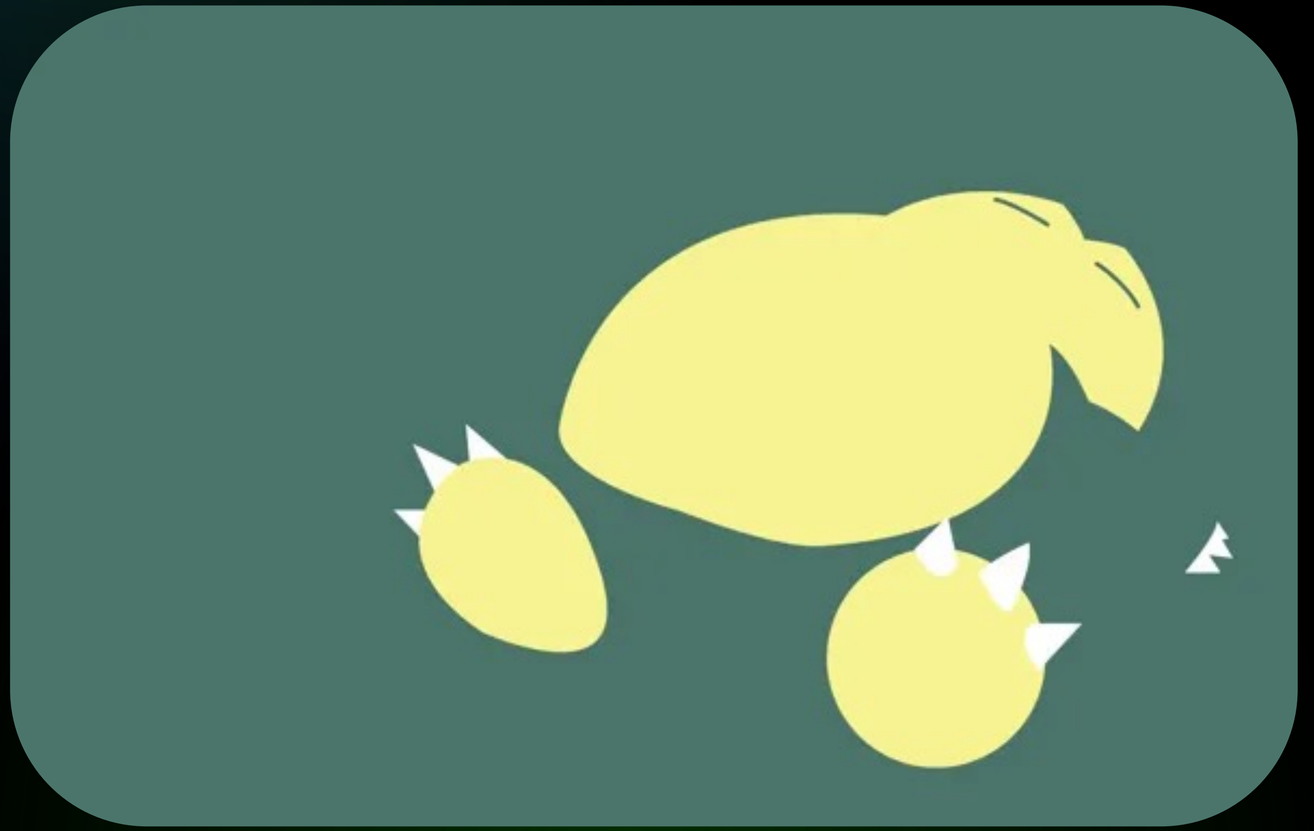
Делаем проще!
Сменим парадигму!

- **Отказываемся**

  от DSL-бинов в пользу утилитных DSL-методов

- **Используем**

  абстрактный ApplicationContextInitializer для заготовки DSL-методов



Делаем проще!
Сменим парадигму!

- **Отказываемся**

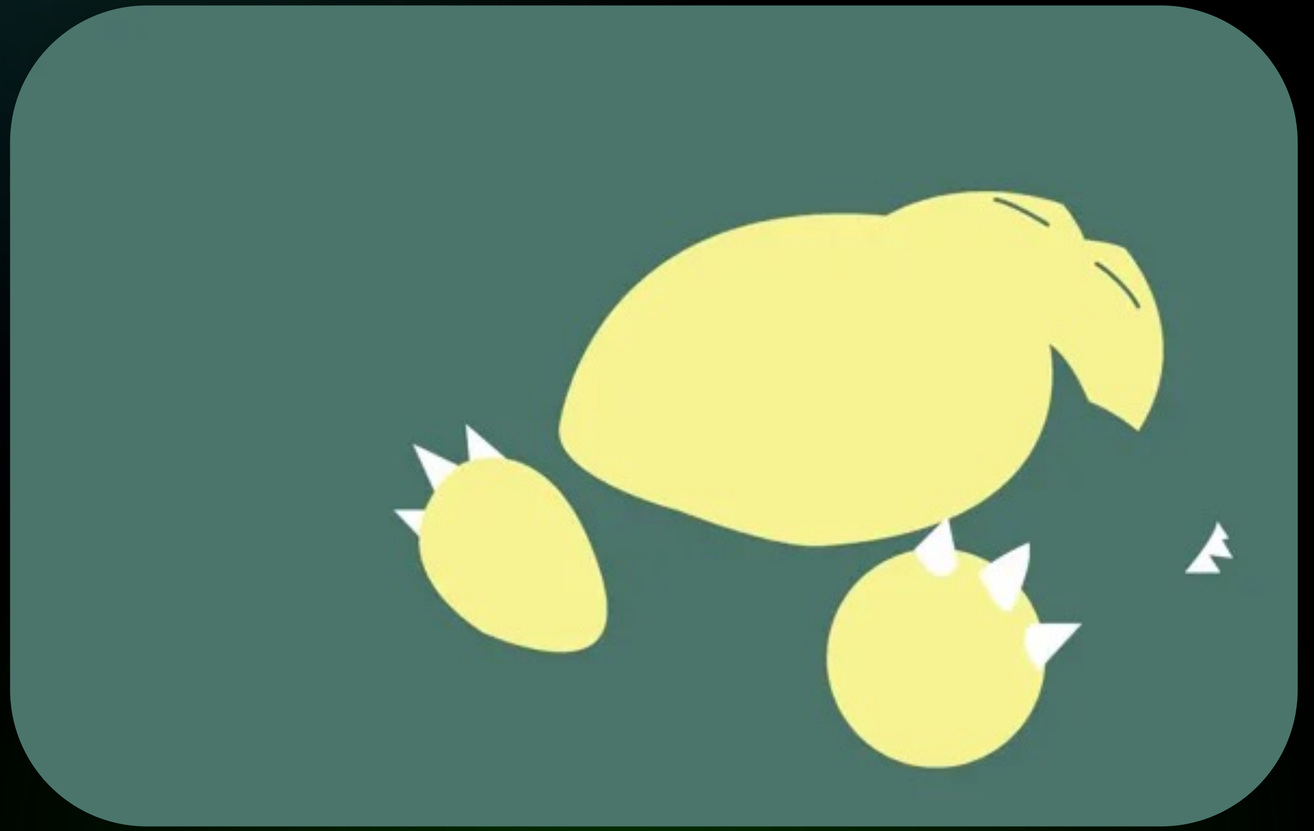
  от DSL-бинов в пользу утилитных DSL-методов

- **Используем**

  абстрактный ApplicationContextInitializer для заготовки DSL-методов

- **Инжектим**

  подготавливаемый Spring-контекст



Делаем проще!
Сменим парадигму!

- **Отказываемся**

  от DSL-бинов в пользу утилитных DSL-методов

- **Используем**

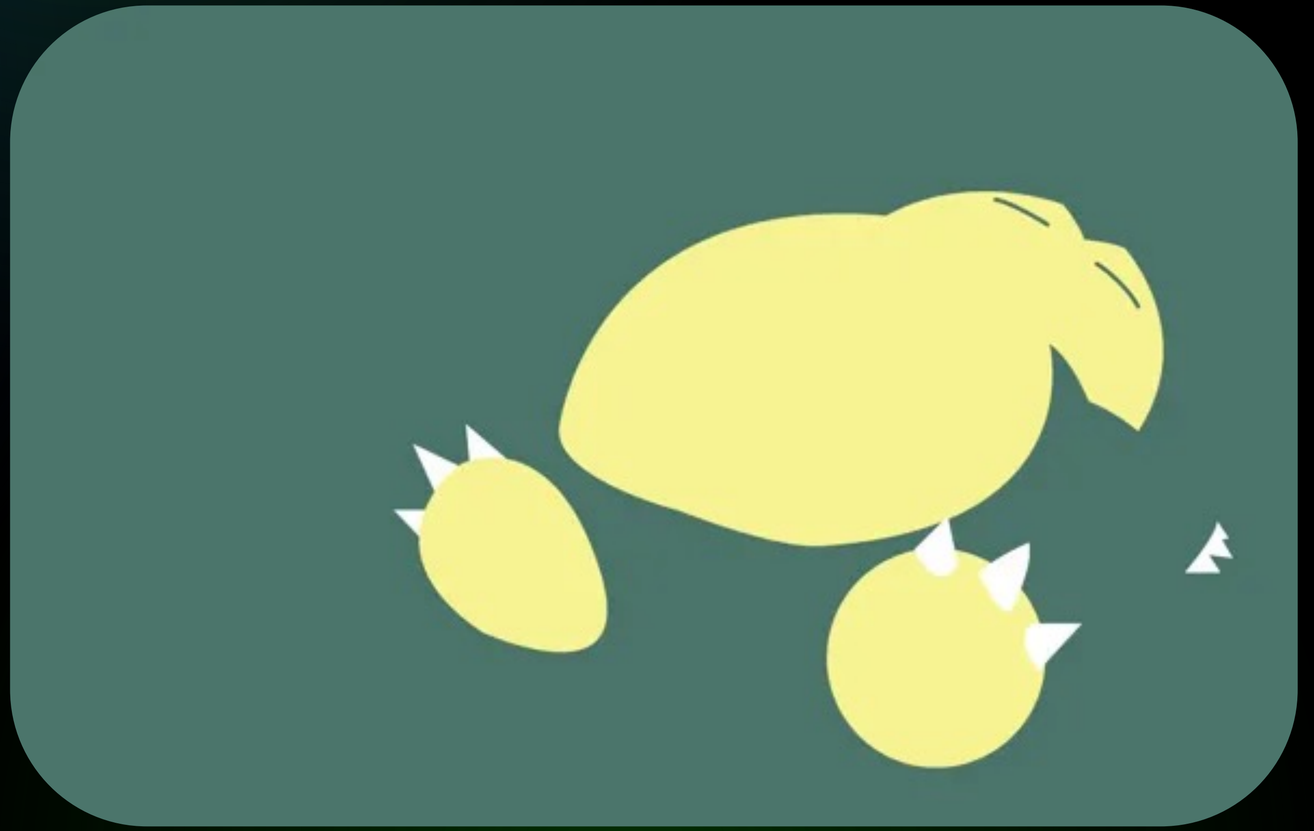
  абстрактный ApplicationContextInitializer для заготовки DSL-методов

- **Инжектим**

  подготавливаемый Spring-контекст

- **Напрямую**

  регистрируем бины в нем, используя функциональный стиль через Instance Supplier’ы




EnterpriseZoo

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

- AnnotatedBeanDefinitionReader
- BeanDefinitionReader
- ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

- BeanFactoryPostProcessor
- BeanDefinitionRegistryPostProcessor
- ImportBeanDefinitionRegistrar

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

- FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

- BeanFactory

Этап 5

Настройка созданных бинов

- BeanPostProcessor

Статья

Этап 0

Подготовка контекста

ApplicationContextInitializer

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

AnnotatedBeanDefinitionReader | BeanDefinitionReader | ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

BeanFactoryPostProcessor | BeanDefinitionRegistryPostProcessor | ImportBeanDefinitionRegistrar

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

BeanFactory

Этап 5

Настройка созданных бинов

BeanPostProcessor

Статья

Этап 0

Подготовка контекста

ApplicationContextInitializer

Этап 1

Парсинг конфигурации (XML, Groovy, JavaConfig и др.) и создание всех BeanDefinition

AnnotatedBeanDefinitionReader | BeanDefinitionReader | ClassPathBeanDefinitionScanner

Этап 2

Настройка созданных BeanDefinition

BeanFactoryPostProcessor | BeanDefinitionRegistryPostProcessor | ImportBeanDefinitionRegistrar

Этап 2.5

Создание BeanPostProcessor’ов, в качестве bean’ов

Этап 3

Создание кастомных FactoryBean

FactoryBean<T>

Этап 4

BeanFactory создает экземпляры бинов, при необходимости делегируя создание бина FactoryBean

BeanFactory

Этап 5

Настройка созданных бинов

BeanPostProcessor

Статья

Пойдет?

Пойдет?

1

Пошли напрямую

и зарегистрировали бины в подготавливаемом контексте

Пойдет?

1

Пошли напрямую

и зарегистрировали бины в подготавливаемом контексте

2

Критичных проблем нет

которые мешали бы работать нашему DSL

Пойдет?

1 **Пошли напрямую**

и зарегистрировали бины в подготавливаемом контексте

2 **Критичных проблем нет**

которые мешали бы работать нашему DSL

3 **Сильно ограничены**

стилем объявления bean’ов через initializer’ы

Пойдет?

1 **Пошли напрямую**

и зарегистрировали бины в подготавливаемом контексте

2 **Критичных проблем нет**

которые мешали бы работать нашему DSL

3 **Сильно ограничены**

стилем объявления bean’ов через initializer’ы

4 **Для библиотеки**

вариант «не очень», поскольку нужно наследоваться от предподготовленного initializer’а

Можно, полаконичнее?

Можно, полаконичнее?

- **Переписываем**

  предыдущий вариант, используя Kotlin Bean DSL

Можно, полаконичнее?

- **Переписываем**

  предыдущий вариант, используя Kotlin Bean DSL

- **Ну**

  собственно, и все...


EnterpriseZoo

А это как?

А это как?

1

Намного лаконичнее

и красивее предыдущего варианта с функциональным стилем

А это как?

1 **Намного лаконичнее**

и красивее предыдущего варианта с функциональным стилем

2 **Дополнительное ограничение**

в виде использования Kotlin

А это как?

1

Намного лаконичнее

и красивее предыдущего варианта с функциональным стилем

2

Дополнительное ограничение

в виде использования Kotlin

3

Те же минусы

что и в случае с функциональным стилем

Что сделали

Что сделали

1

Показали как сделать DSL

на Spring для регистрации бинов 4 разными способами

Что сделали

1

Показали как сделать DSL

на Spring для регистрации бинов 4 разными способами

2

Сравнили

их преимущества и недостатки, а также точки выполнения их логики

Что сделали

1

Показали как сделать DSL

на Spring для регистрации бинов 4 разными способами

2

Сравнили

их преимущества и недостатки, а также точки выполнения их логики

3

Показали примеры проблем

которые можно встретить при использовании их, в том числе проблема ранней инициализации

Что сделали

1 **Показали как сделать DSL**

на Spring для регистрации бинов 4 разными способами

2 **Сравнили**

их преимущества и недостатки, а также точки выполнения их логики

3 **Показали примеры проблем**

которые можно встретить при использовании их, в том числе проблема ранней инициализации

4 **Прошлись по ключевым точкам**

в логике резолвинга типа бина в Spring

Итоги

Итоги

1

У каждого подхода к написанию DSL

есть свои плюсы и минусы – идеального варианта нет

Итоги

1

У каждого подхода к написанию DSL

есть свои плюсы и минусы – идеального варианта нет

2

Будьте аккуратны

при использовании метода getBeanNamesForType

Итоги

1

У каждого подхода к написанию DSL

есть свои плюсы и минусы – идеального варианта нет

2

Будьте аккуратны

при использовании метода getBeanNamesForType

3

Не используйте

FactoryBean’ы, если это возможно

Итоги

1. **У каждого подхода к написанию DSL**

   есть свои плюсы и минусы – идеального варианта нет

2. **Будьте аккуратны**

   при использовании метода getBeanNamesForType

3. **Не используйте**

   FactoryBean’ы, если это возможно

4. **Если создаешь BeanDefinition программно**

   обращай внимание на его атрибуты

Итоги

1. **У каждого подхода к написанию DSL**

   есть свои плюсы и минусы – идеального варианта нет

2. **Будьте аккуратны**

   при использовании метода getBeanNamesForType

3. **Не используйте**

   FactoryBean’ы, если это возможно

4. **Если создаешь BeanDefinition программно**

   обращай внимание на его атрибуты

5. **Не надо бояться**

   смотреть в source’ы и разбираться в том, как работает Spring Framework

Спасибо за внимание!

Зубенко Евгений

Шестаков Максим