Spring Boot 2

Чего не пишут в release notes

Владимир Плизгá
ЦФТ

Обо мне

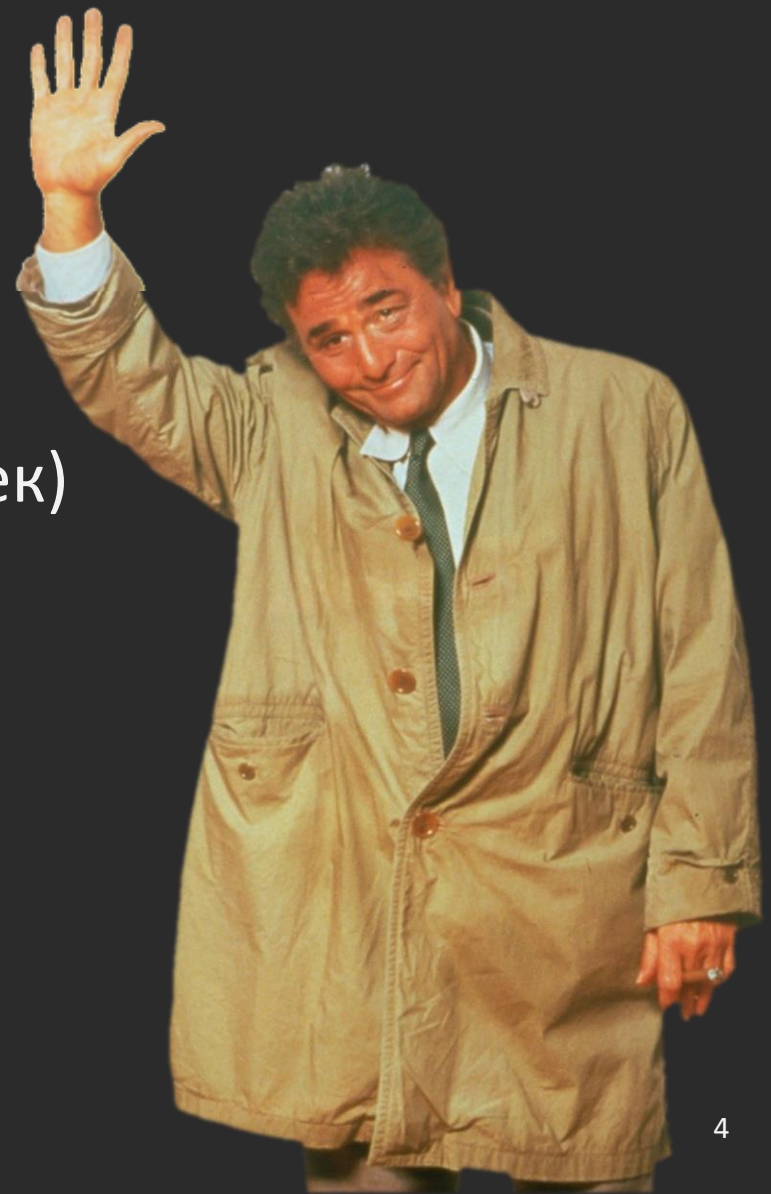
- Владимир Плизгá
  https://github.com/Toparvion
- ЦФТ (Центр Финансовых Технологий)
  В топ-3 крупнейших разработчиков ПО в России
- Backend-разработчик (Java)
  ≈7 лет в деле
- TechLead в команде Интернет-банка
- Инициатор перехода к микросервисам

О подопытном продукте

- Универсальный Интернет-банк для держателей предоплаченных карт
  - ≈ 20 партнеров по РФ
- 1-е место в **Markswebb Mobile Banking Rank 2017** в категории «Детали операций» http://markswebb.ru/e-finance/mobile-banking-rank-2017/
- 1 ядро и 20+ микросервисов на Spring Boot
  - + Spring Cloud (Netflix), Integration, Batch, …
- На Spring Boot 2 в production с июля 2018

О чём поговорим

- Compile time (API)
- Content-Type (web)
- Scheduling (запуск по расписанию)
- Spring Cloud & Co. (совместимость библиотек)
  + JMX (проксирование бинов)
- Relax Binding (внешние свойства)
- Unit Testing (Mockito 2)
- Gradle Plugin (сборка Spring Boot проектов)
- Прочее: Boot, Cloud, Integration



Вводная: Spring Boot / 2

- В статусе GA (General Availability) с 1 марта 2018 г.
- Минимум Java 8
- На основе Spring Framework 5
- ≈150 зависимостей в BOM (Bill Of Materials)

  ⇒не все изменения растут из Spring
- Есть отличные Release Notes и Migration Guide

Compile Time

Примеры изменений в API

Основные источники изменений

- Переход Spring на Java 8
- Разделение web-стека
- Исправление недочетов
- Новшества сторонних библиотек
- [Прочее]

Первое впечатление от Release Notes

На чем сломается компиляция

- **Почему:**
  Класса `WebMvcConfigurerAdapter` больше нет

- **Зачем:**
  Для поддержки фишек Java 8 (default-методы в интерфейсах)

- **Что делать:**
  Использовать интерфейс `WebMvcConfigurer`

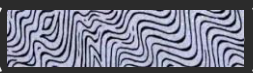
На чем сломается компиляция

- **Почему**:
  Метод `PropertySourceLoader#load` стал возвращать список источников вместо одного

- **Зачем**:
  Для поддержки мульти-документных ресурсов, например, YAML

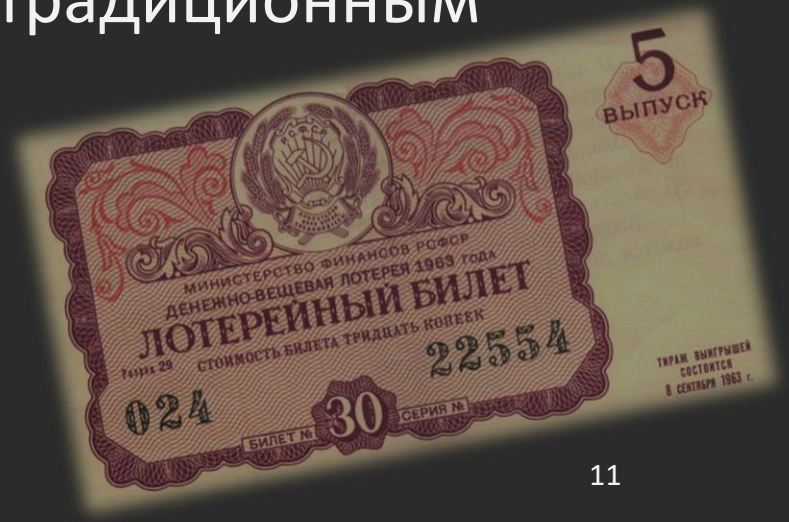
- **Что делать**:
  Оборачивать ответ в `singletonList()` (при переопределении)

На чем сломается компиляция

- **Почему**:
  Поменялась сигнатура методов класса `ErrorAttributes`:
  вместо `RequestAttributes` стали использоваться
  `WebRequest` () и `ServerRequest` ()

- **Зачем**:
  Чтобы поддержать реактивный стек наравне с традиционным

- **Что делать**:
  Заменить имена классов в сигнатурах



На чем сломается компиляция

- **Почему**:
  Некоторые классы из пакета `org.springframework.boot.autoconfigure.web` разъехались по пакетам `org.springframework.boot.autoconfigure.web.servlet` и `.reactive`
- **Зачем**:
  Чтобы поддержать реактивный стек наравне с традиционным
- **Что делать**:
  Обновить import’ы

И как быть?

- Заглядывать Spring Boot 2.0 Migration Guide

  Например, Developing Web Applications / Embedded containers package structure

- Учитывать, что в версии v2.x понятие «`web`» уточняется с помощью «`servlet`» и «`reactive`»

- Проверять названия зависимостей:

```
spring-cloud-starter-eureka-server →
spring-cloud-starter-netflix-eureka-server
```

Content-Type

Определение типа HTTP-ответа

Web-приложения на Spring/Boot

- Два стека: v1.x – servlet, v2.x – servlet & reactive
- Можно отдавать контент с разными типами (JSON, XML, HTML, …)
- Можно **не** указывать тип ответа при отдаче с сервера,
  т.к. есть автоопределение:
  - По заголовку запроса Accept (`Accept: “application/json”`)
  - По расширению запрошенного файла (`GET /document.json`)
  - По параметру запроса (`GET /document?format=json`)

Например (v1.x)

```
dependencies {
    ext {
        springBootVersion = '1.5.14.RELEASE'
    }
    compile("org.springframework.boot:spring-boot-starter-web:$springBootVersion")
}
```

Полный исходный код:
https://github.com/toparvion/joker-2018-samples/tree/master/content-type

Контроллер раздачи файлов

ContentTypeDemoApplication.java

```
@GetMapping(value = "/download/{fileName:.+}",
            produces = {TEXT_HTML_VALUE, APPLICATION_JSON_VALUE, TEXT_PLAIN_VALUE})
public ResponseEntity<Resource> download(@PathVariable String fileName) {
  // формируем только тело ответа, без Content-Type
}
```

Полный исходный код:
https://github.com/toparvion/joker-2018-samples/tree/master/content-type

И это работает! (v1.x)

Варианты запроса	**Content-Type** (определяется Spring'ом)	**OK?**
`GET /download/document.html`	`Content-Type → text/html`	
`GET /download/document.json`	`Content-Type → application/json`	
`GET /download/document.txt`	`Content-Type → text/plain`	

Обновляем до v2.x

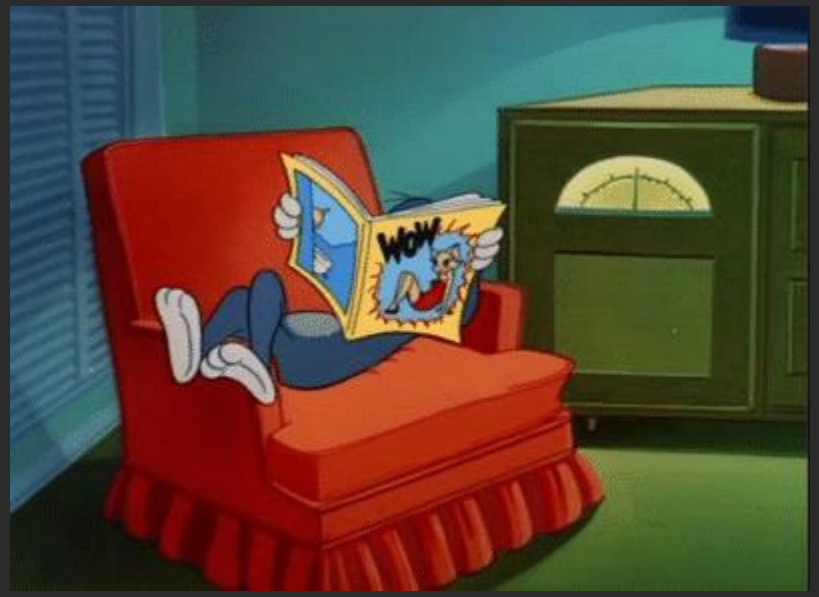
```
dependencies {
    ext {
        springBootVersion = '2.0.4.RELEASE'
    }
    compile("org.springframework.boot:spring-boot-starter-web:$springBootVersion")
}
```

И даже смотрим Migration Guide

Не наш случай
(к счастью)

«Spring MVC Path Matching Default Behavior Change

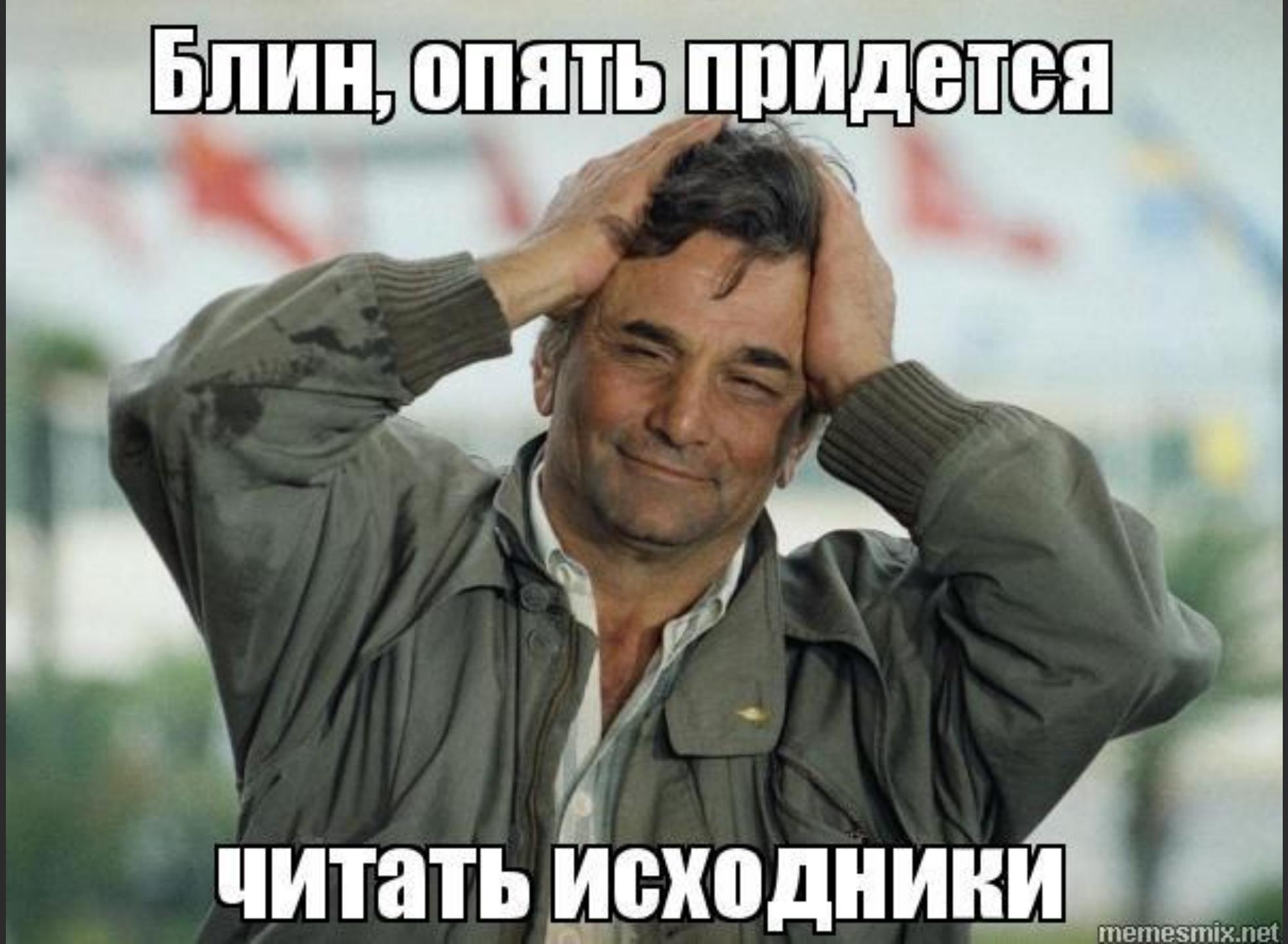
We’ve decided to change the default for suffix path matching in Spring MVC applications…»



https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide#spring-mvc-path-matching-default-behavior-change

Проверяем (v2.x)

Варианты запроса	**Content-Type** (определяется Spring'ом)	**OK?**
`GET /download/document.html`	`Content-Type → text/html`	
`GET /download/document.json`	`Content-Type → text/html`	
`GET /download/document.txt`	`Content-Type → text/html`	


Блин, опять придется
читать исходники
memesmix.net

ContentNegotiationManagerFactoryBean

```
public ContentNegotiationManager build() {
 List<ContentNegotiationStrategy> strategies = new ArrayList<>();

 if (this.strategies != null) {
  strategies.addAll(this.strategies);
 }
 else {
  if (this.favorPathExtension) {
   PathExtensionContentNegotiationStrategy strategy;
    // ...
```


Среднее имя бина в Spring
вот такой длины

Источник расхождения

```
public class ContentNegotiationManagerFactoryBean
  implements FactoryBean<ContentNegotiationManager>, ... {

 //...

 private boolean favorPathExtension = true
```

Spring Framework **4/5**

```
public static class Contentnegotiation {

 /**
  * Whether the path extension in the URL path should be used to determine the
  * requested media type. If enabled a request "/users.pdf" will be interpreted as
  * a request for "application/pdf" regardless of the 'Accept' header.
  */
 private boolean favorPathExtension = false
```

Spring Boot **2**

И как быть?

- В Spring Boot 2 флаг `favorPathExtension` вынесли в параметры, это хорошо.
- Но не с таким значением по умолчанию, какое было в Spring Framework, это плохо.
- Что делать?
  - Переключить параметр у себя в проекте

На всякий случай

- Смотрим, что пишу… в документации на Spring Boot

Причем тут определение Content-Type?!

«If you understand the caveats and would still like your application to use suffix pattern matching, the following configuration is required:

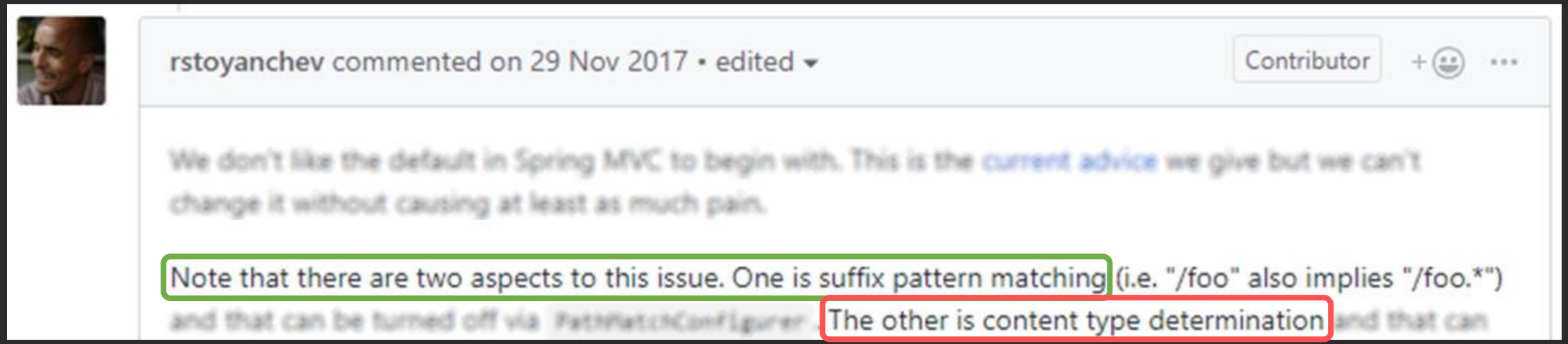
```
spring.mvc.contentnegotiation.favor-path-extension=true
```

…»

https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#boot-features-spring-mvc-pathmatch

Истина где-то рядом

В Migration Guide есть ссылка на pull request #11105:



https://github.com/spring-projects/spring-boot/issues/11105

Т.е. изменение `favorPathExtension` связано
с правками по path matching.


Я Ж ГОВОРИЛ,
НА ГИТХАБЕ НАДО ИСКАТЬ
PicsCom.com

Тем временем в Spring Framework Reference

- Using file extensions … was necessary when browsers used to send Accept headers that were hard to interpret consistently.
- At present that is no longer a necessity and using the "Accept" header should be the preferred choice.
- Over time the use of file name extensions has proven problematic in a variety of ways.

https://docs.spring.io/spring/docs/5.0.9.RELEASE/spring-framework-reference/web.html#mvc-ann-requestmapping-suffix-pattern-match

теперь точно Резюме

- Изменение `favorPathExtension` – не баг, а фича
- Неразрывно связана с изменениями в path matching
- Призвана:
  - Снизить риски по безопасности
  - Выровнять WebFlux и WebMvc
  - Выровнять заявления в документации с кодом фреймворка

И как быть?

- Не полагаться на определение `Content-Type` по расширению

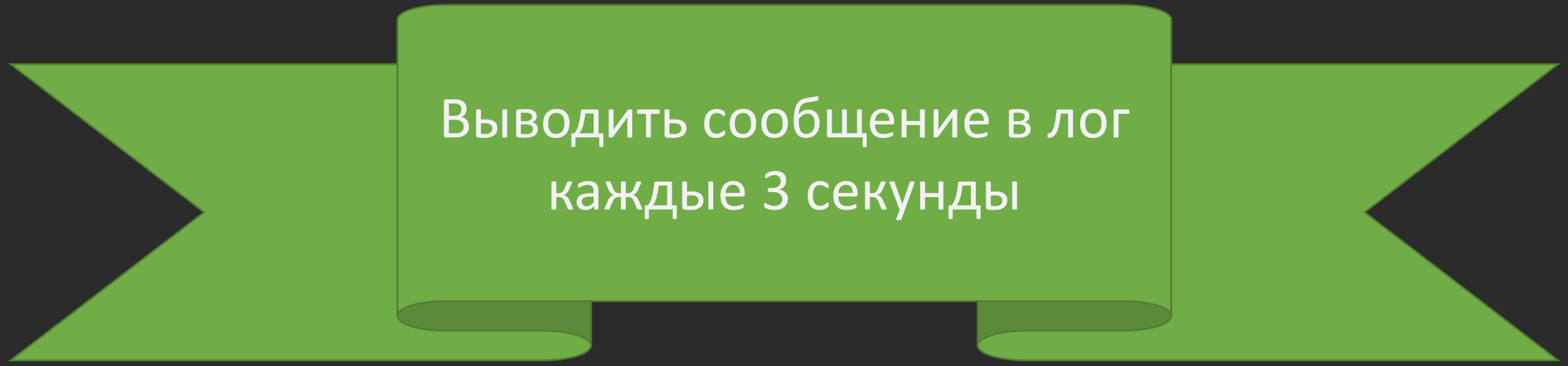
  И на то, что запрос `GET /resource.json` попадет на `GET /resource`

- Полагаться на заголовок `Accept` либо параметр (e.g. `format`)
- Если никак, то выставить `spring.mvc.contentnegotiation.favor-path-extension=true`
- Почитать:
  - Главу **Suffix match** в Spring Framework Reference
  - CVE-2015-5211 (RFD - Reflected File Download Attack)

Scheduling

Выполнение задач
по расписанию или периодически

Пример задачи

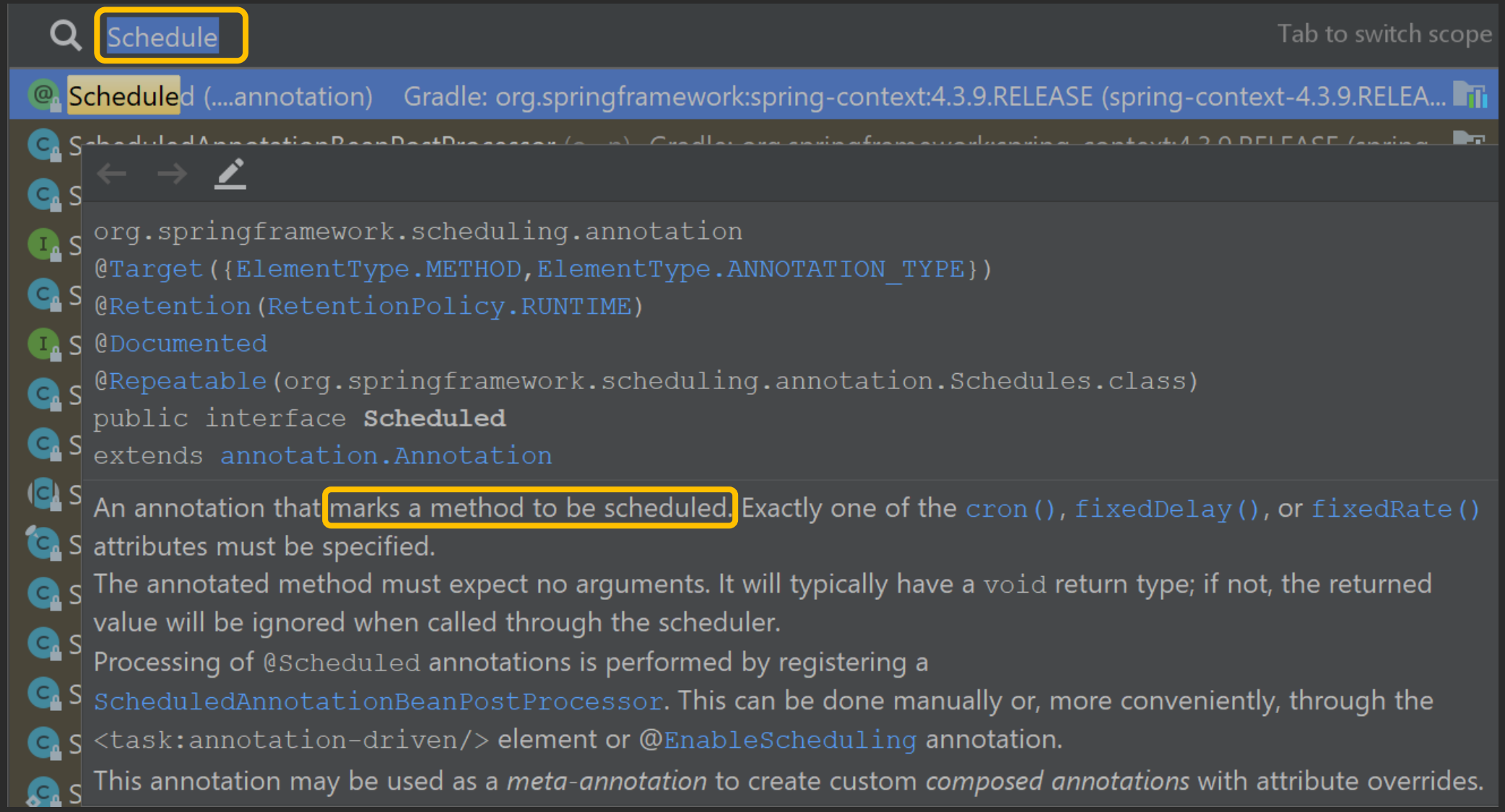



ОТВЕЧАЮ
В СПРИНГЕ ЭТО УЖЕ ЕСТЬ
PicsCom.com

Вариант 1: поиск примера в своём проекте

```java
/**
 * A very helpful service
 * @since v0.9
 */
@Service
public class ReallyBusinessService {

  // ... a bunch of methods ...

  @Scheduled(fixedDelay = 3000L)
  public void runRepeatedlyWithFixedDelay() {
    assert Runtime.getRuntime().availableProcessors() >= 4;
  }

  // ... another bunch of methods ...
}
```

Вариант 2: поиск нужной аннотации



Вариант 3: Googling

So, for example, you could have something like:

```java
@Component
public class EventCreator {

    private static final Logger LOG = LoggerFactory.getLogger(EventCreator.class);

    private final EventRepository eventRepository;

    public EventCreator(final EventRepository eventRepository) {
        this.eventRepository = eventRepository;
    }

    @Scheduled(fixedRate = 1000)
    public void create() {
        final LocalDateTime start = LocalDateTime.now();
        eventRepository.save(
            new Event(new EventKey("An event type", start, UUID.randomUUID()), Math.random() * 1000));
        LOG.debug("Event created!");
    }
}
```

https://dzone.com/articles/running-on-time-with-springs-scheduled-tasks

Show me the code!

Разбираемся

Ожидание

«…you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.»

https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#getting-started-introducing-spring-boot

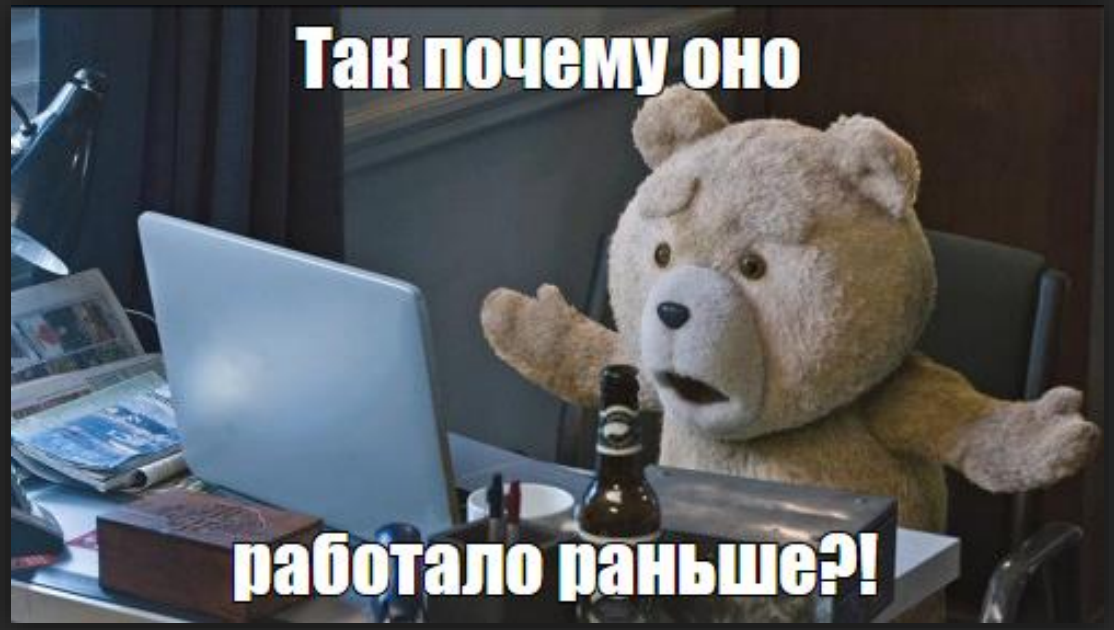
Реальность

К первоисточнику

«7.4.1. Enable scheduling annotations

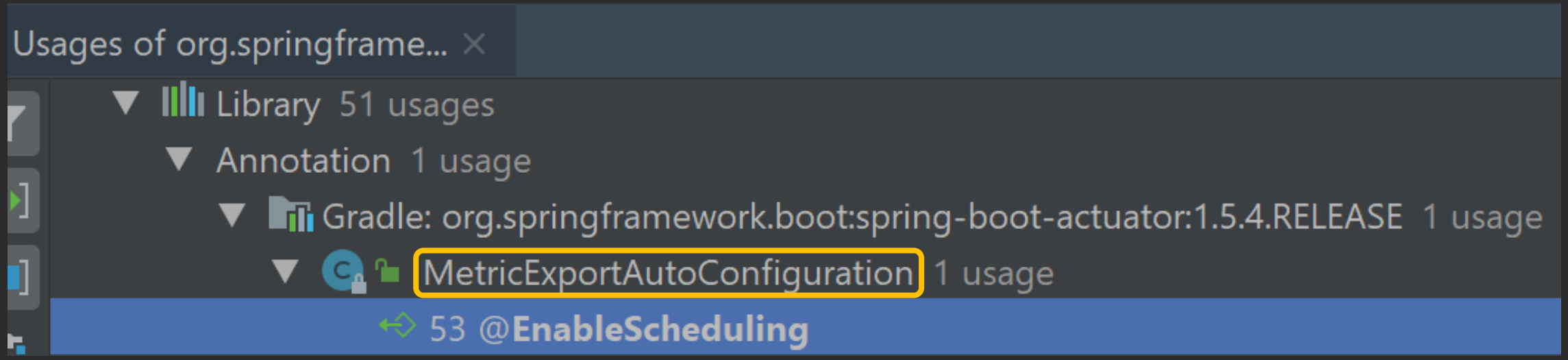
To enable support for `@Scheduled` … annotations add `@EnableScheduling` … to one of your `@Configuration` classes.»

https://docs.spring.io/spring/docs/current/spring-framework-reference/integration.html#scheduling-enable-annotation-support



На заборе тоже написано

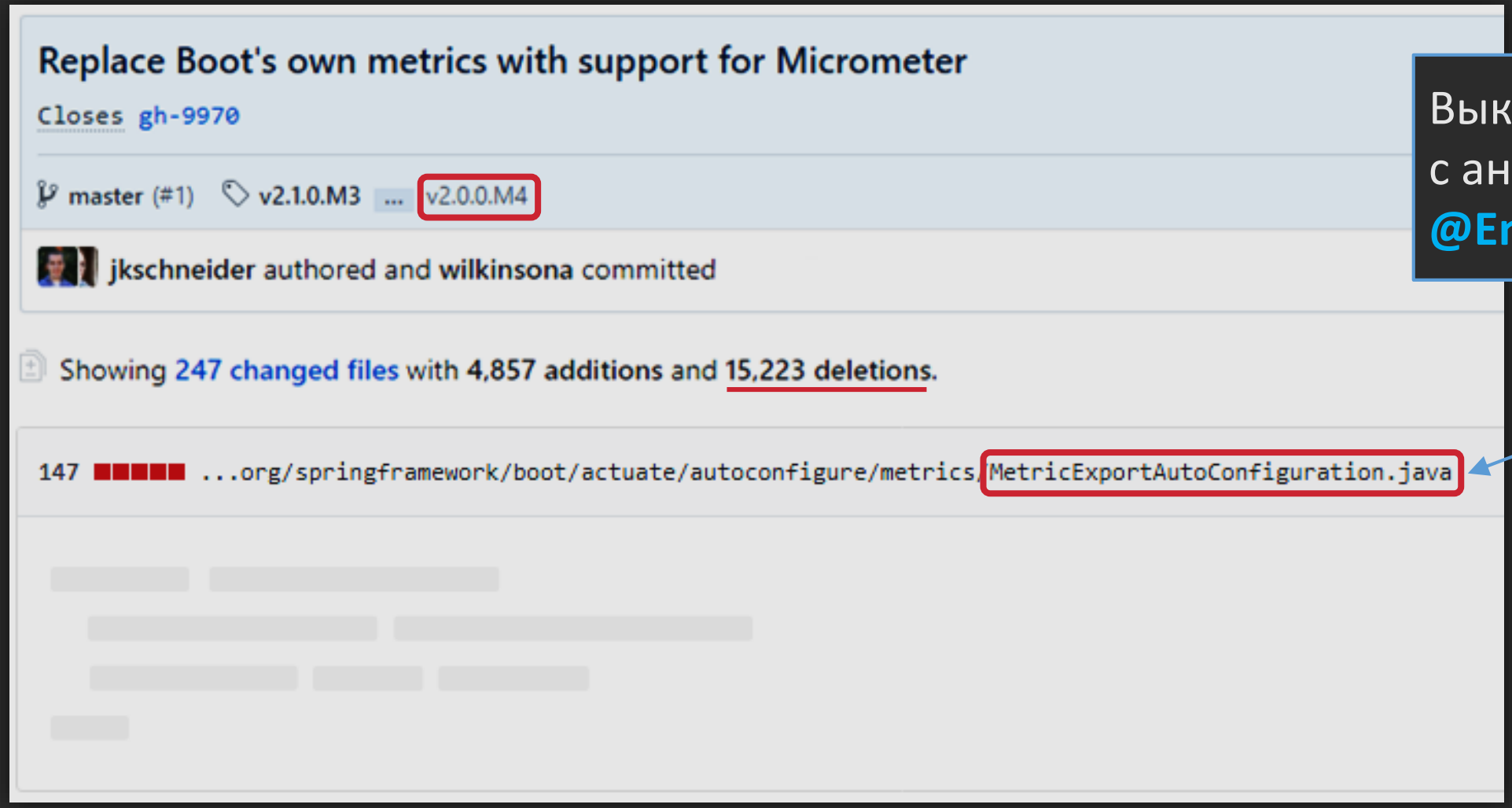
```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Import(SchedulingConfiguration.class)
@Documented
public @interface EnableScheduling {
```



Копаем глубже

```
/**
 * {@link EnableAutoConfiguration Auto-con
 *
 * …
 * …
 * @since 1.3.0
 */
@Configuration
@EnableScheduling
@ConditionalOnProperty(value = "spring.metr
@EnableConfigurationProperties
public class MetricExportAutoConfiguration
```

А почему сломалось?

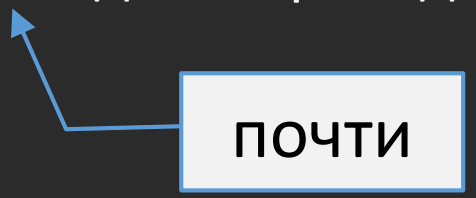


Выкошен ~~на хрен~~ вместе с аннотацией **@EnableScheduling**

Show me the code!

И как быть?

1. Читать не только копируемые фрагменты документации
2. Помнить, что некоторым фичам Spring Boot (scheduling, async, caching, …) нужен включатель
3. Перестраховываться:
   - добавлять аннотации `@Enable*` в свой код, не надеясь на фреймворк
   - (дублирование аннотаций никогда не приводит к ошибкам)*



*)

- Аннотации `@EnableAsync` и `@EnableCaching` имеют атрибуты
  => их можно [случайно] задать разными
- Из javadoc в классе `AutoProxyRegistrar`:

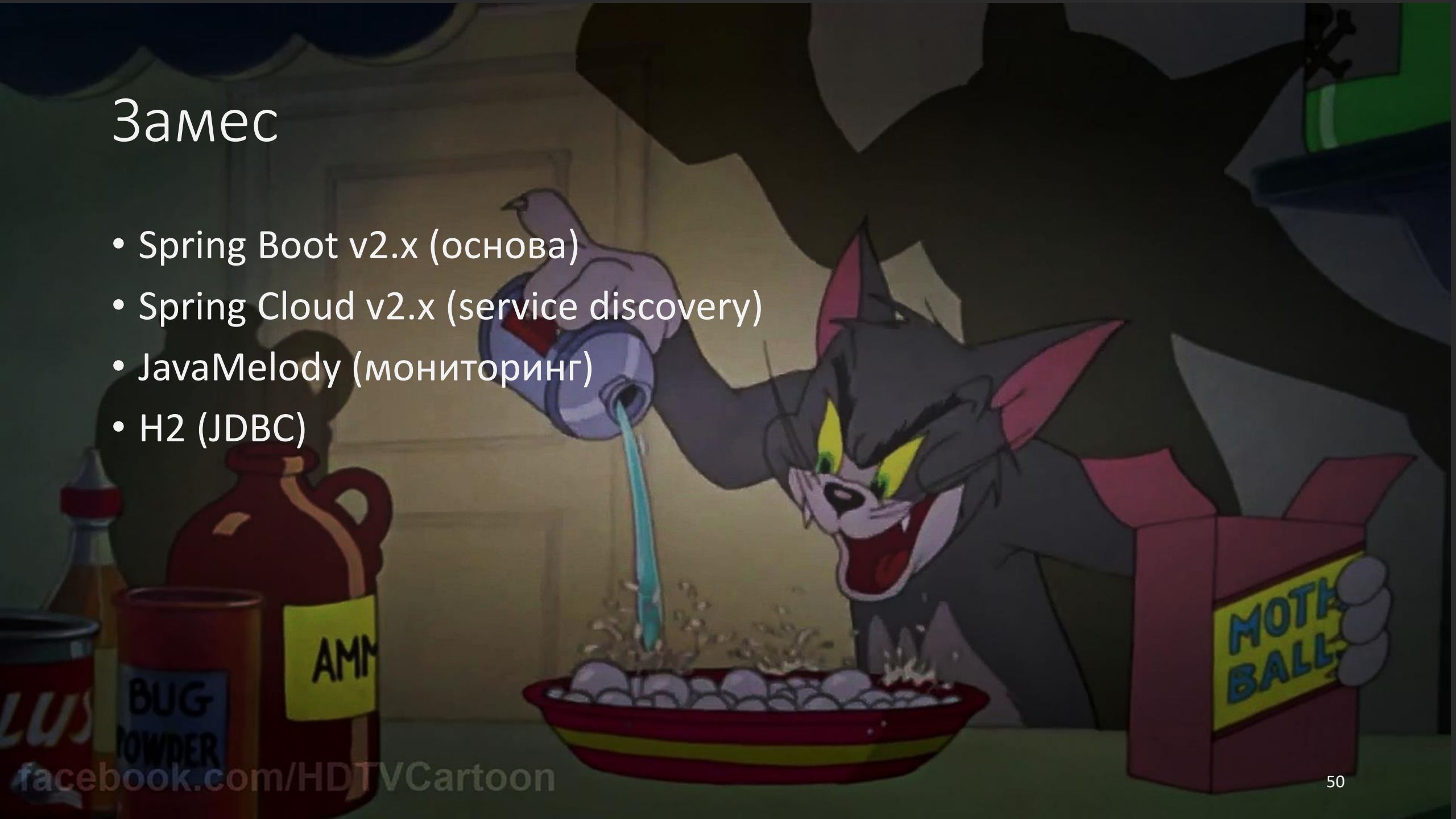
Works by finding the nearest annotation ...
Several `@Enable*` annotations expose both `mode` and `proxyTargetClass` attributes. It ... end up sharing a single APC (auto proxy creator).
... implementation doesn't "care" exactly which annotation it finds.

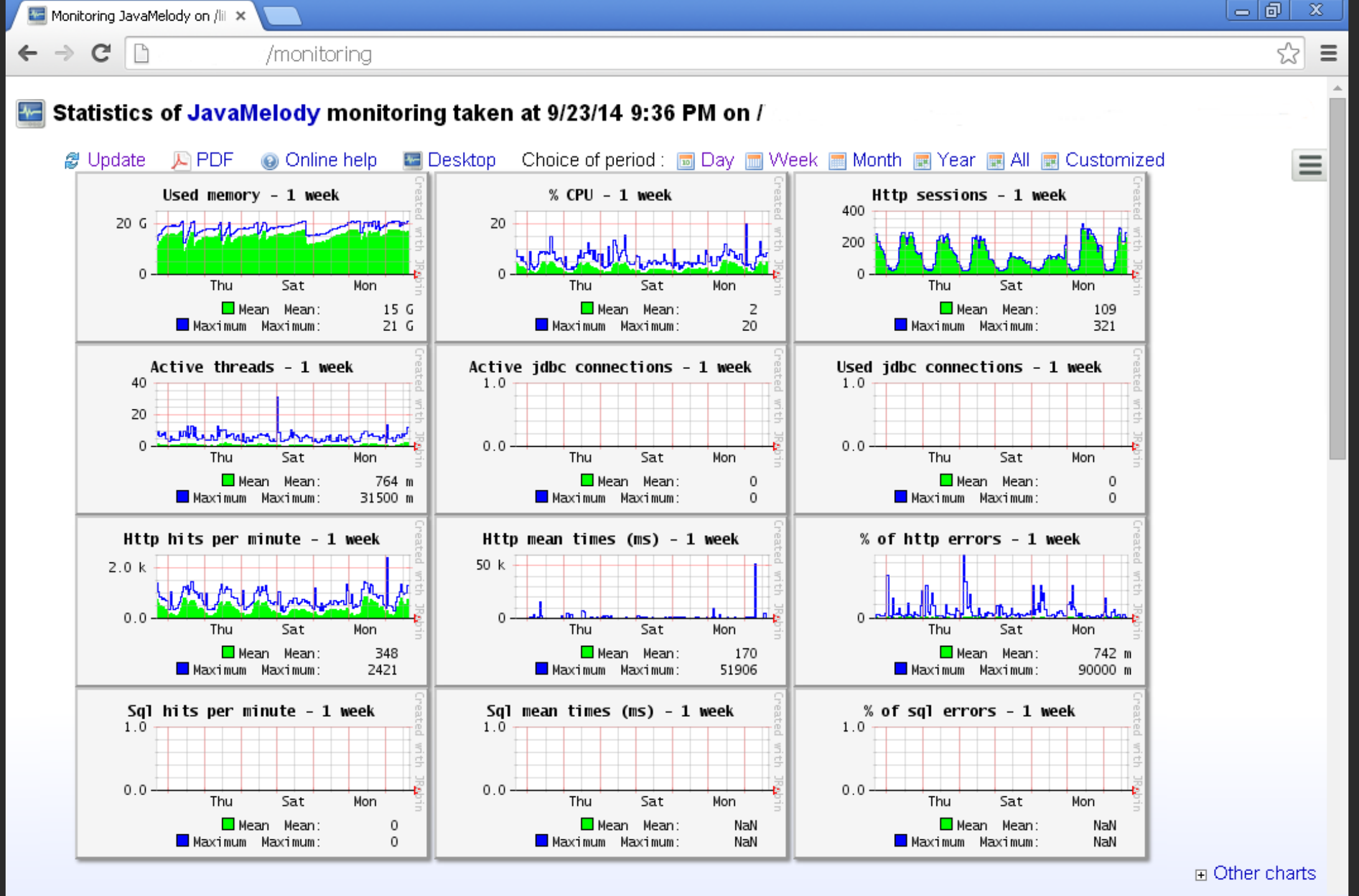
SpringCloud & Co.

Совместимость библиотек

Замес

- Spring Boot v2.x (основа)
- Spring Cloud v2.x (service discovery)
- JavaMelody (мониторинг)
- H2 (JDBC)




Monitoring JavaMelody on /
/monitoring
Statistics of JavaMelody monitoring taken at 9/23/14 9:36 PM on /
Update
PDF
Online help
Desktop
Choice of period :
Day
Week
Month
Year
All
Customized
Used memory - 1 week
20 G
0
Thu
Sat
Mon
Mean Mean: 15 G
Maximum Maximum: 21 G
% CPU - 1 week
20
0
Thu
Sat
Mon
Mean Mean: 2
Maximum Maximum: 20
Http sessions - 1 week
400
200
0
Thu
Sat
Mon
Mean Mean: 109
Maximum Maximum: 321
Active threads - 1 week
40
20
0
Thu
Sat
Mon
Mean Mean: 764 m
Maximum Maximum: 31500 m
Active jdbc connections - 1 week
1.0
0.0
Thu
Sat
Mon
Mean Mean: 0
Maximum Maximum: 0
Used jdbc connections - 1 week
1.0
0.0
Thu
Sat
Mon
Mean Mean: 0
Maximum Maximum: 0
Http hits per minute - 1 week
2.0 k
0.0
Thu
Sat
Mon
Mean Mean: 348
Maximum Maximum: 2421
Http mean times (ms) - 1 week
50 k
0
Thu
Sat
Mon
Mean Mean: 170
Maximum Maximum: 51906
% of http errors - 1 week
0
Thu
Sat
Mon
Mean Mean: 742 m
Maximum Maximum: 90000 m
Sql hits per minute - 1 week
1.0
0.0
Thu
Sat
Mon
Mean Mean: 0
Maximum Maximum: 0
Sql mean times (ms) - 1 week
1.0
0.0
Thu
Sat
Mon
Mean Mean: NaN
Maximum Maximum: NaN
% of sql errors - 1 week
1.0
0.0
Thu
Sat
Mon
Mean Mean: NaN
Maximum Maximum: NaN
Other charts

Подопытный кролик

```
dependencies {
    ext {
        springBootVersion  = '2.0.4.RELEASE'
        springCloudVersion = '2.0.1.RELEASE'
    }
    compile("org.springframework.boot:spring-boot-starter-web:$springBootVersion")
    runtime("org.springframework.boot:spring-boot-starter-jdbc:$springBootVersion")
    runtime group: "org.springframework.cloud",
            name: "spring-cloud-starter-netflix-eureka-client",
            version: springCloudVersion
    runtime("net.bull.javamelody:javamelody-spring-boot-starter:1.72.0")
    // ...
}
```

Полный исходный код:
https://github.com/toparvion/joker-2018-samples/tree/master/hikari-javamelody

Подопытный кролик

```
@SpringBootApplication
public class HikariJavamelodyDemoApplication {

  public static void main(String[] args) {
    SpringApplication.run(HikariJavamelodyDemoApplication.class, args);
  }
}
```

Полный исходный код:
https://github.com/toparvion/joker-2018-samples/tree/master/hikari-javamelody

Запускаем

```
ERROR o.s.boot.SpringApplication                                  :
Application run failed

Caused by: java.lang.ClassCastException:
com.sun.proxy.$Proxy76
cannot be cast to
com.zaxxer.hikari.HikariDataSource
```

Материалы дела

1. Spring Cloud оборачивает dataSource в прокси:
   - Нужно для обновления бинов на лету (RefreshScope)
   - Используется только CGLIB проксирование
   - Обёртывание производится раньше всех BeanPostProcessor’ов

Материалы дела

2. JavaMelody оборачивает dataSource в прокси:
   - Нужно для снятия данных для мониторинга
   - Используется только JDK проксирование
   - Обёртывание производится посредством BeanPostProcessor’а

Материалы дела

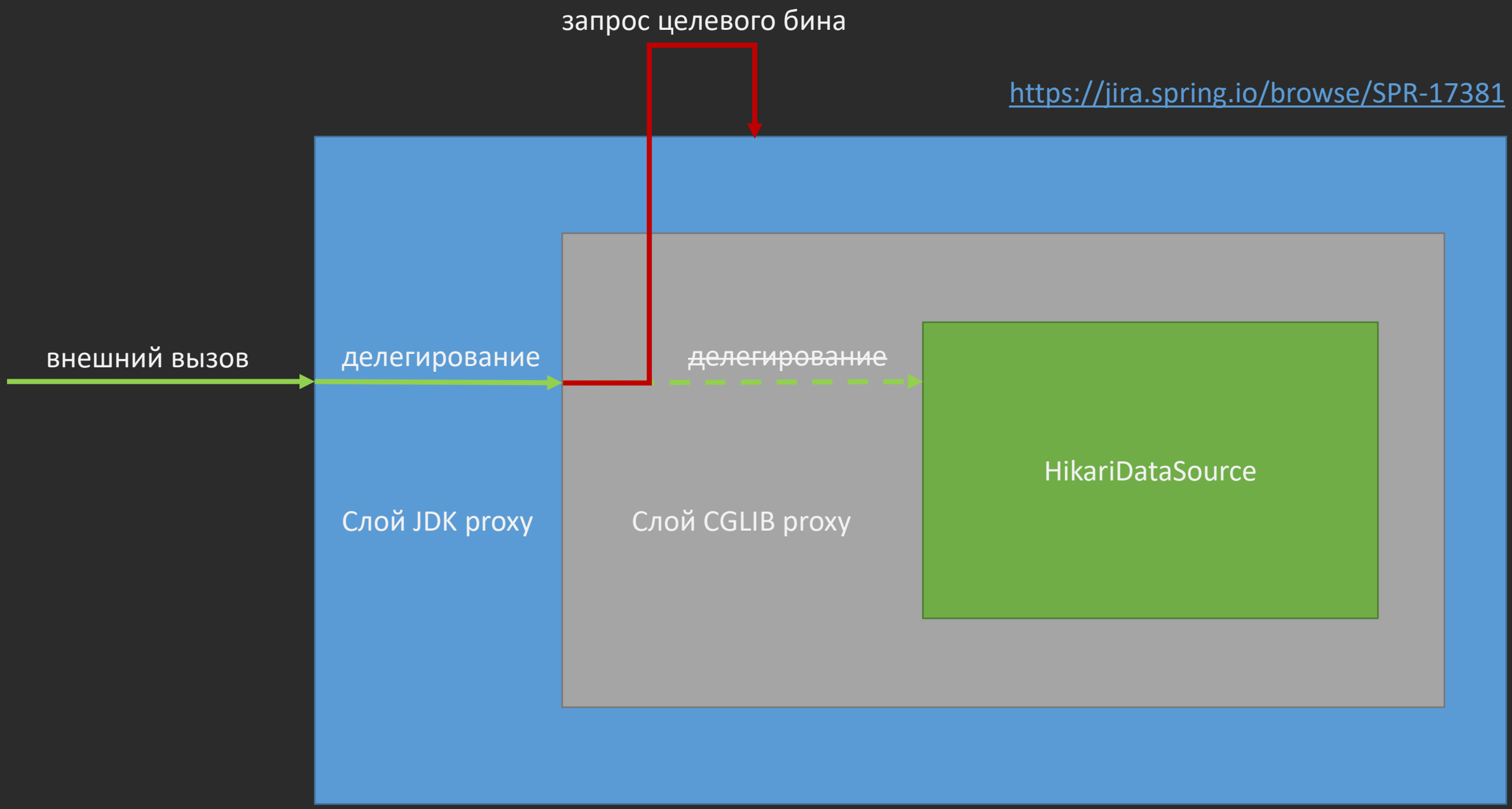
JDK proxy

CGLIB proxy

```
ataSource() {

riDataSource.class);   dataSource: com.sun.proxy.$Proxy77@6990

                                                this.dataSource

90} ... toString()
gatingInvocationHandler@7001}
pper$3@6983}
iDataSource$$EnhancerBySpringCGLIB$$54e860e6@6897} ... toString()
apper@7002}
kariDataSource$$EnhancerBySpringCGLIB$$54e860e6@6897} ... toString()
```

Материалы дела

3. Spring Boot вызывает dataSource.unwrap():
   - Нужно для выставления dataSource через JMX
   - JDK-прокси от JavaMelody пропускает вызов сквозь себя
   - CGLIB-прокси от Cloud снова запрашивает бин у контекста
   - Получает JDK-обертку, применяет к ней CGLIB методы и ломается


запрос целевого бина
https://jira.spring.io/browse/SPR-17381
внешний вызов
делегирование
делегирование
HikariDataSource
Слой JDK proxy
Слой CGLIB proxy


А ПРИЧЕМ ТУТ
HIKARI ?

Наблюдения & соображения

- Spring Cloud проксирует **только** Hikari data source
  (если сменить Hikari на другой пул, то проблемы нет)
- HikariCP стал умолчательным пулом в Spring Boot **2**
- Spring Cloud [по идее] не должен знать о пуле коннектов к БД

А на самом деле...

org.springframework.cloud.autoconfigure.RefreshAutoConfiguration
.RefreshScopeBeanDefinitionEnhancer:

```
/**
 * Class names for beans to post process into refresh scope. Useful when you
 * don't control the bean definition (e.g. it came from auto-configuration).
 */
private Set<String> refreshables = new HashSet<>(
  Arrays.asList("com.zaxxer.hikari.HikariDataSource"));
```

Выводы

- Все обновляемые бины создаются Spring Cloud’ом сразу в CGLIB-обёртках
- Не все прокси-обёртки одинаково ~~полезны~~ хорошо совместимы друг с другом (https://jira.spring.io/browse/SPR-17381)
- Оборачивать в прокси могут не только BeanPostProcessor’ы
- Не всегда можно переключиться между CGLIB и JDK прокси

Совместная работа компонентов приложения

И как быть? (в таком случае)

Выбрать обходной путь:

- Переключиться на другой пул (e.g. Tomcat JDBC Pool)
  `spring.datasource.type=org.apache.tomcat.jdbc.pool.DataSource`
  , не забыв добавить зависимость
  `runtime 'org.apache.tomcat:tomcat-jdbc:8.5.29'`
- Отключить JDBC мониторинг в JavaMelody
  `javamelody.excluded-datasources=scopedTarget.dataSource`
- Отключить обновление «на лету» в Spring Cloud:
  `spring.cloud.refresh.enabled=false`


ВСЕГО ОДИН СЛУЧАЙ?
НЕ СЕРЬЕЗНО.
PicsCom.com

Бонус (схожий случай*)

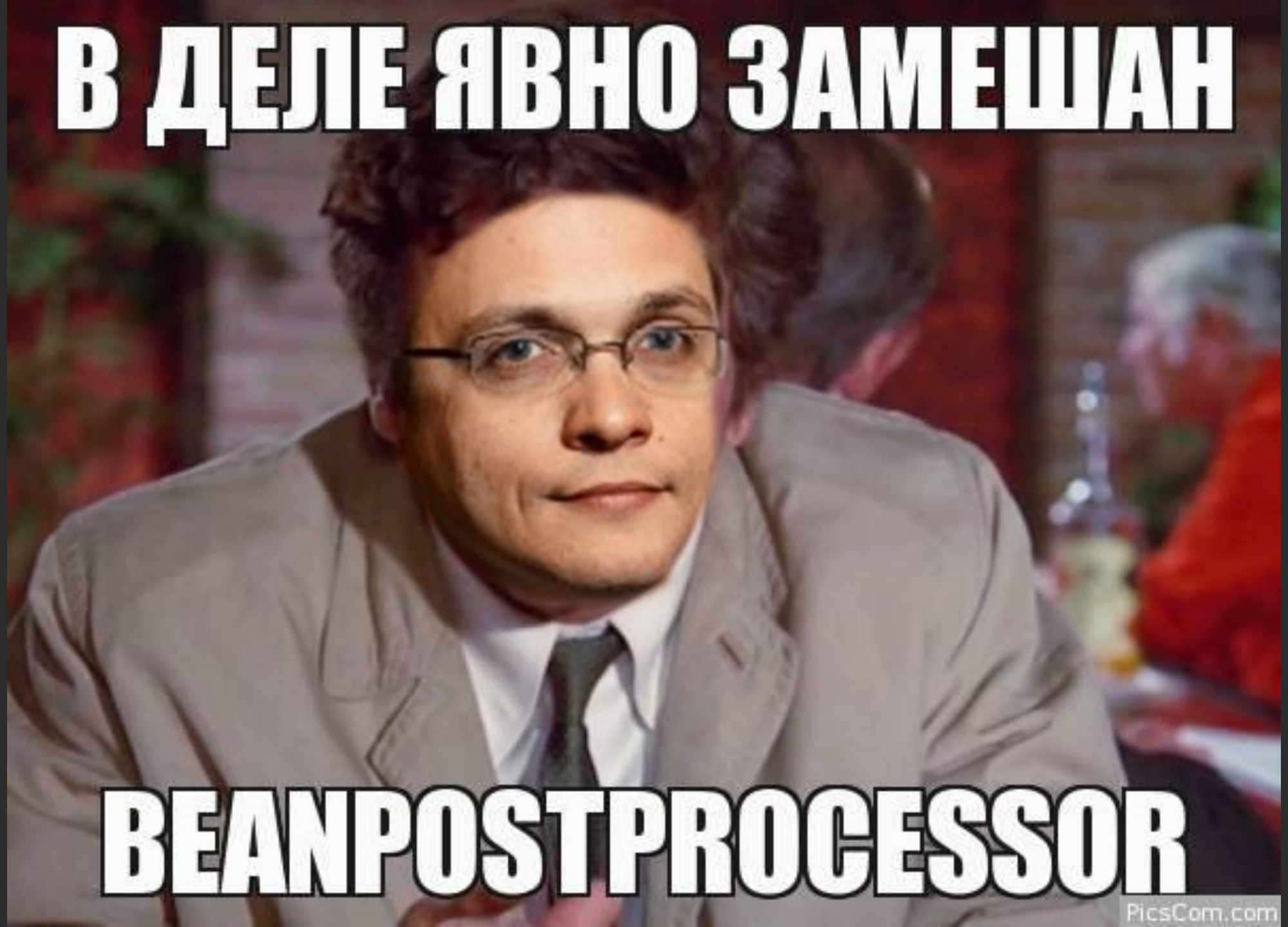
*но без Spring Cloud (и можно без JavaMelody)

```
@Component
@ManagedResource
@EnableAsync
public class MyJmxResource {

  @ManagedOperation
  @Async
  public void launchLongLastingJob() {
    // какой-то долгоиграющий код
  }
}
```

Полный исходный код:
https://github.com/toparvion/joker-2018-samples/tree/master/jmx-resource

Симптомы

- Приложение успешно стартует
- Ошибок в логах нет
- Бин `myJmxResource` не доступен по JMX ☹
- Бин `myJmxResource` обернут в 2 прокси: CGLIB и JDK


В ДЕЛЕ ЯВНО ЗАМЕШАН
BEANPOSTPROCESSOR
PicsCom.com

Причастные BeanPostProcessor’ы

1. `AsyncAnnotationBeanPostProcessor`

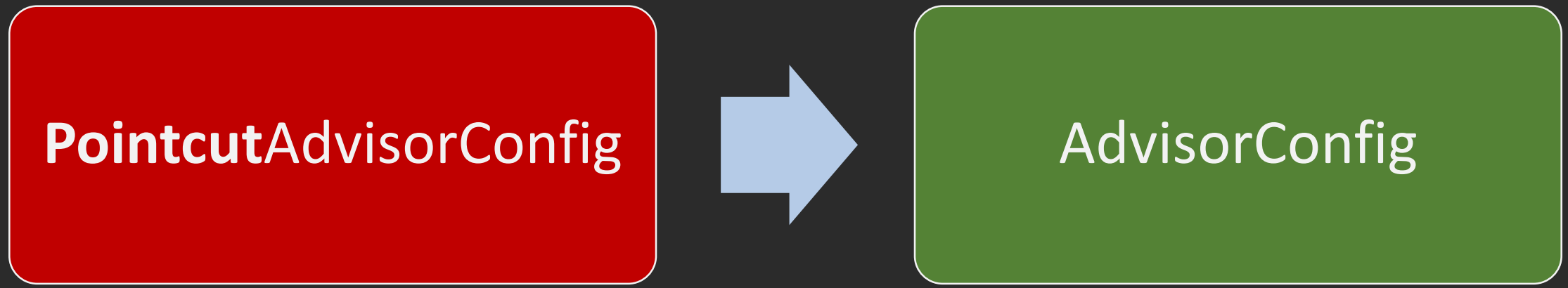
- **Должность**: директор по работе с аннотацией `@Async`
- **Прописка**: `org.springframework.scheduling`
- **Место рождения:** аннотация `@EnableAsync` (через `@Import`)

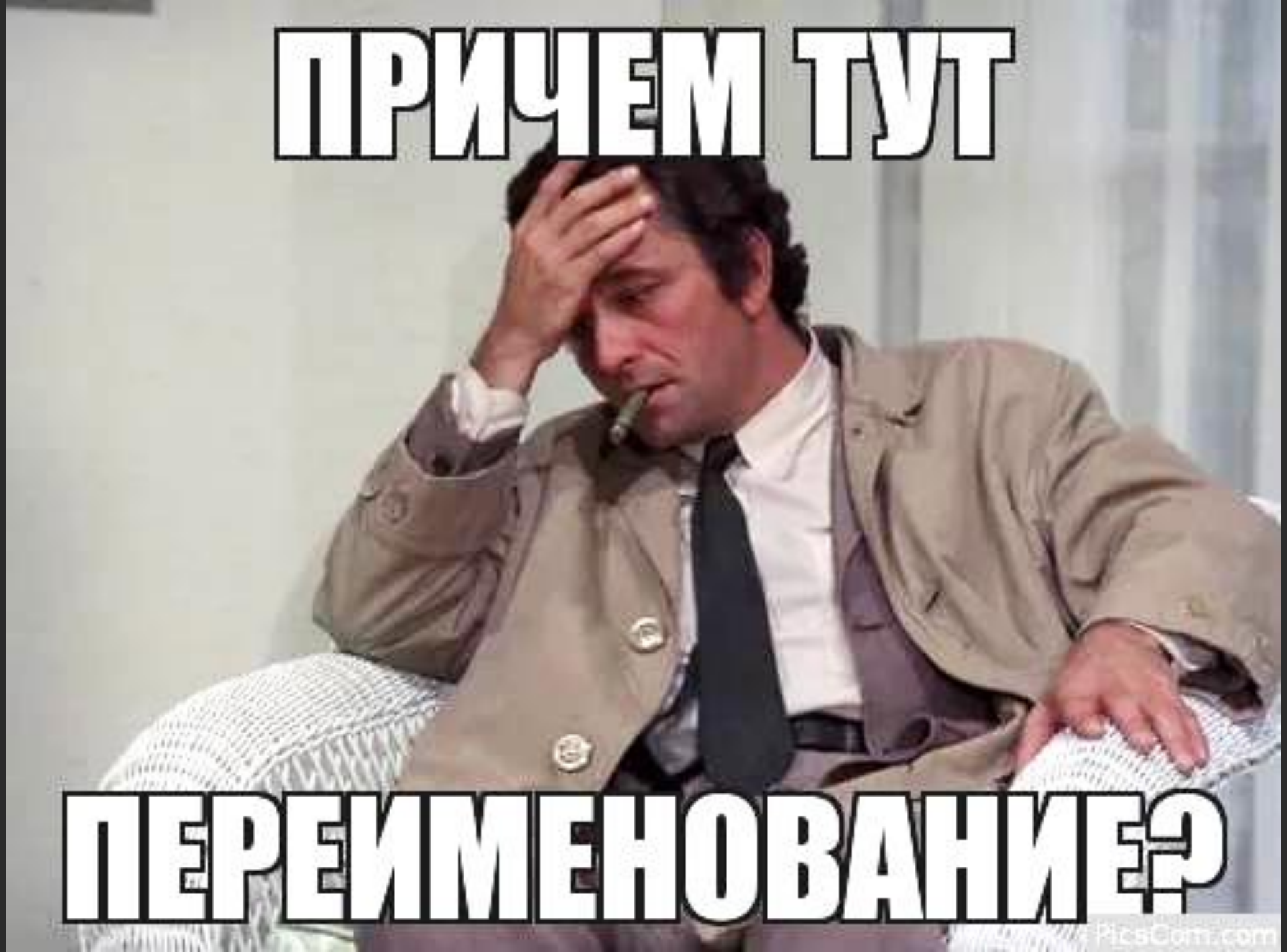
2. `DefaultAdvisorAutoProxyCreator`

- **Должность**: помощник по работе с AOP proxy
- **Прописка**: `org.springframework.aop.framework.autoproxy`
- **Место рождения:** `@Configuration`-класс `PointcutAdvisorConfig` (библиотечный либо самописный)

Наблюдение

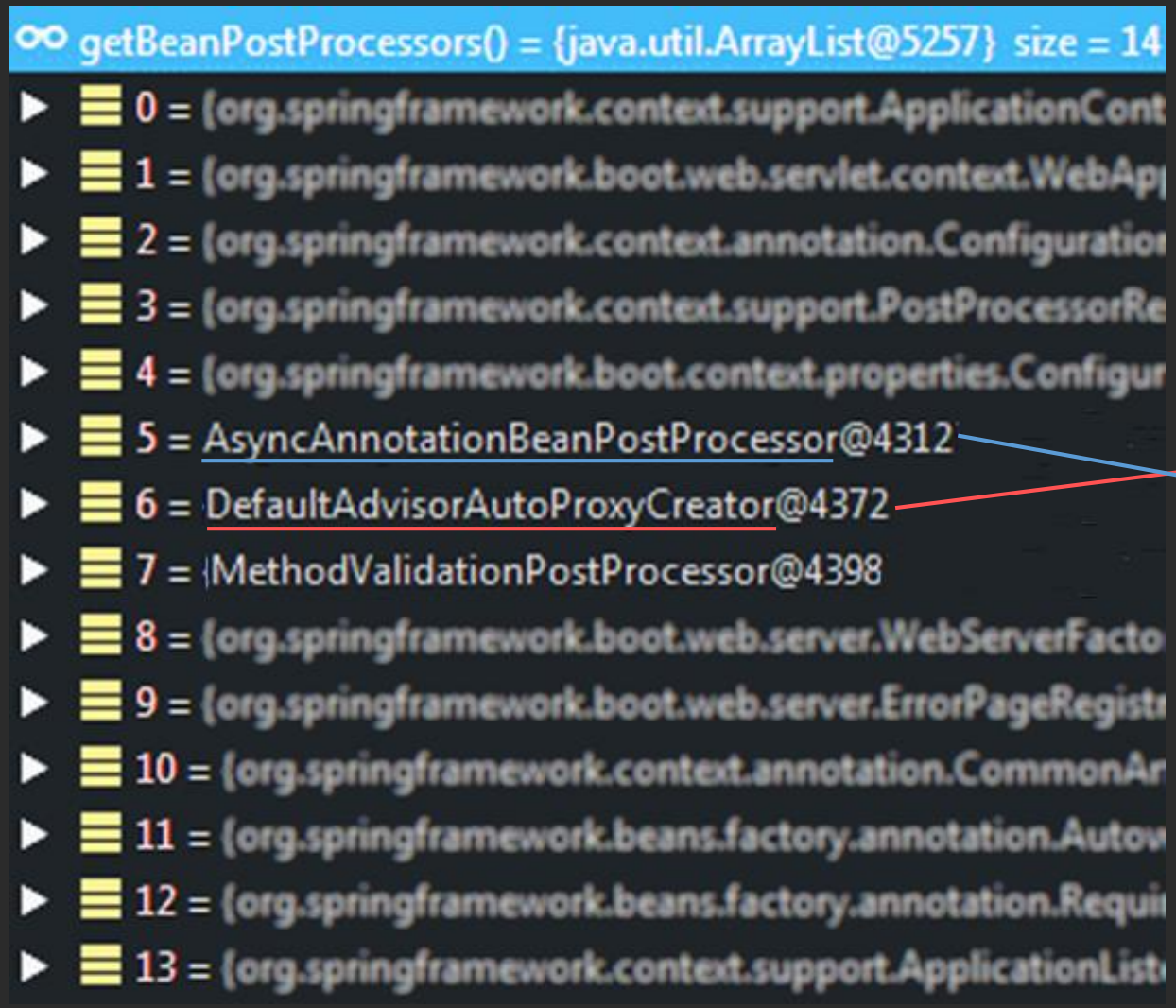
Если переименовать `@Configuration`-класс,
создающий бин `DefaultAdvisorAutoProxyCreator`,
то ошибки нет (JMX-бин доступен).



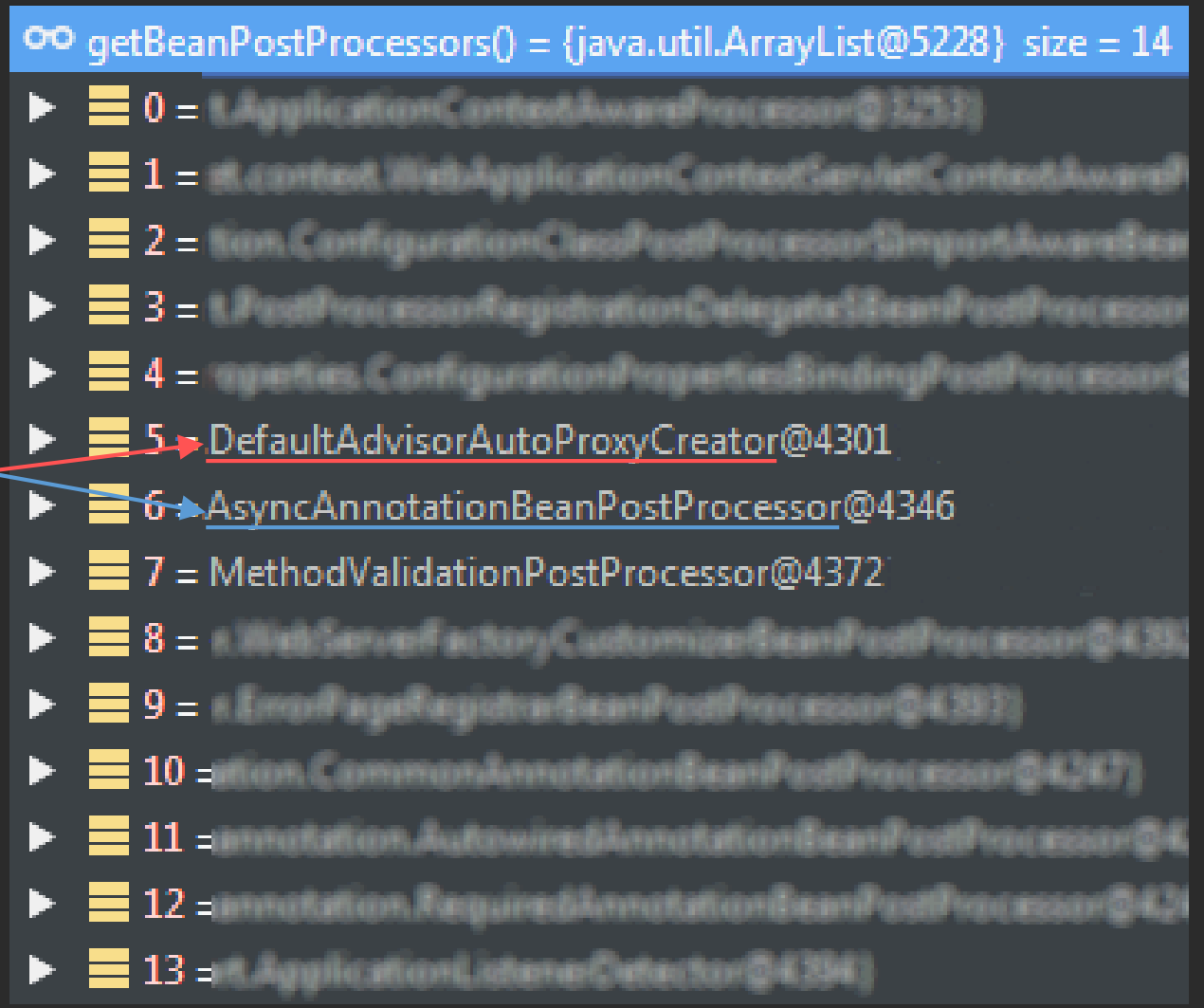

ПРИЧЕМ ТУТ
ПЕРЕИМЕНОВАНИЕ?

Наборы пост-процессоров

КОГДА **НЕ** РАБОТАЕТ



КОГДА РАБОТАЕТ




П
О
В
Л
И
Я
Л
О

Визуализация процесса

И как быть? (в таком случае)

- По возможности использовать полноценные аспекты (вместо «сырых» Advice’ов и Advisor’ов)
- Прятать прикладные бины под интерфейсами
- Если всё же сломалось:
  - Смотреть через отладчик на состав прокси
  - Пробовать autowire’ить проблемный бин в любой другой бин
  - Рулить порядком бинов через аннотации `@Order` (где применимо)
  - Рулить флажками `proxyTargetClass` на аннотациях (где можно)

И как быть? (в общем случае)

- Keep calm and YAGNI
- Интересоваться, как работают применяемые библиотеки
- Не включать все подряд фишки Spring Boot/Cloud на всякий случай

Relax Binding

Работа со свойствами (параметрами) приложения

Вводная

- Возможность чтения свойств приложения из внешних источников без строгого совпадения имён
- Например, свойство `String firstName` с префиксом `acme.my-project.person` можно задать **любым** из следующих способов:

Property	Note
`acme.my-project.person.first-name`	Kebab case, which is recommended for use in `.properties` and `.yml` files.
`acme.myProject.person.firstName`	Standard camel case syntax.
`acme.my_project.person.first_name`	Underscore notation, which is an alternative format for use in `.properties` and `.yml` files.
`ACME_MYPROJECT_PERSON_FIRSTNAME`	Upper case format, which is recommended when using system environment variables.

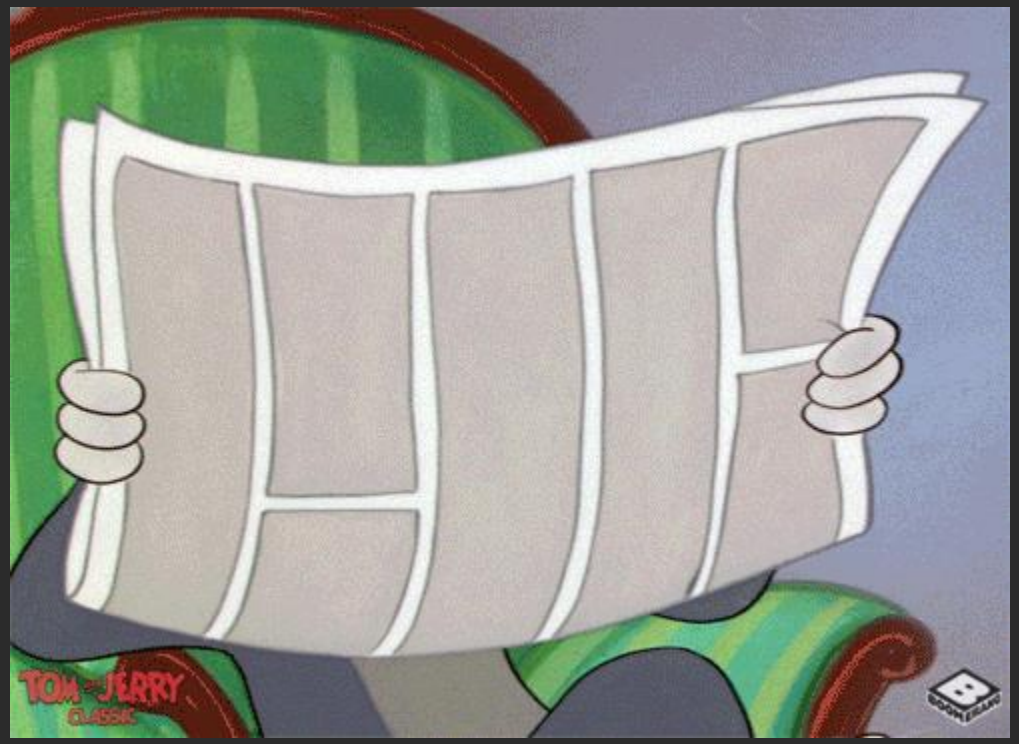
https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#boot-features-external-config-relaxed-binding

Изменения в v2.x

- В **Spring Boot 2.x** механизм был существенно переделан:
  - Ужесточены правила привязки
  - Унифицирован способ задания имен свойств в коде приложения

Документация

- https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide#relaxed-binding (≈1 страница)
- https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#boot-features-external-config-relaxed-binding (≈3 страницы)
- https://github.com/spring-projects/spring-boot/wiki/Relaxed-Binding-2.0 (≈4 страницы)



Но грабли-то остались☺

Пример

```
dependencies {
    ext {
        springBootVersion = '1.5.4.RELEASE'
    }
    compile("org.springframework.boot:spring-boot-starter:$springBootVersion")
}
```

Полный исходный код:

https://github.com/toparvion/joker-2018-samples/tree/master/relax-binding

Пример

```
@SpringBootApplication
public class RelaxBindingDemoApplication implements ApplicationRunner {
  private static final Logger log =
    LoggerFactory.getLogger(RelaxBindingDemoApplication.class);

  @Autowired
  private SecurityProperties securityProperties;

  public static void main(String[] args) {
   SpringApplication.run(RelaxBindingDemoApplication.class, args);
  }

  @Override
  public void run(ApplicationArguments args) {
    log.info("KEYSTORE TYPE IS: {}", securityProperties.getKeyStoreType());
  }
}
```

Пример

```java
@Component
@ConfigurationProperties(prefix = "security")
public class SecurityProperties {
  private String keystorePath;
  private String keystoreType;

  public String getKeystorePath() {
    return keystorePath;
  }

  public void setKeystorePath(String keystorePath) {
    this.keystorePath = keystorePath;
  }

  public String getKeyStoreType() {
    return keystoreType;
  }

  public void setKeystoreType(String keystoreType) {
    this.keystoreType = keystoreType;
  }
}
```

application.properties

```
security.keystorePath=...
security.keystoreType=jks
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.4.RELEASE)

INFO RelaxBindingDemoApplication    : Starting RelaxBindingDemoApplication on ...
INFO RelaxBindingDemoApplication    : No active profile set ...
INFO Annot nfigApplicationContext : Refreshing context ...
INFO Ap nExporter         : Registering beans for JMX exposure on startup
INFO R moApplication    : KEYSTORE TYPE IS: jks
INFO R emoApplication    : Started RelaxBindingDemoApplication ...
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.2.RELEASE)
...
***************************
APPLICATION FAILED TO START
***************************
Description:
Failed to bind properties under 'security' to
tech.toparvion.sample.joker18.relax.SecurityProperties:
    Property: security.keystoretype
    Value: jks
    Origin: class path resource [application.properties]:2:23
    Reason: No setter found for property: key-store-type
Action:
Update your application's configuration
```


Я ТУТ ПОЧИТАЛ
ДОКУ ПО SPRING


ТАМ ТАКОГО
ПОНАПИСАНО!..
PicsCom.com

Режим паранойи

Проверяем:

- наличие свойства `key-store-type` – **нет такого**
- имена поля и параметра в **.properties**:

```
private String keystoreType;
security.keystoreType=jks
```

совпадают

- имя setter’а: `void setKeystoreType(…)` – **ОК**
- имя getter’а: `String getKeyStoreType()` – **опаньки**


ugadajfilm.com
ЧТО ТЫ СКАЗАЛ?
GETTER?
risovach.com

Что происходит

- Первоисточником списка свойств бина стали его getter’ы (в т.ч. `getKeyStoreType()`)
- Под каждое свойство должен быть найден setter
- Но под свойство `keyStoreType` такого setter’а нет ☹

На самом деле

Пример реального класса

И как быть?

1. Сверять регистры букв в именах свойств
2. **Заранее** проверять все нужные источники свойств (properties/YAML, environment, JVM opts)
3. Почитывать **Spring Boot Relaxed Binding 2.0**
4. *Надеяться, что в v3.x такого не повторится*

Unit Testing

Выполнение тестов в Mockito 2

Причем тут Mockito?

```
$gradle -q dependencyInsight --configuration testCompile --dependency mockito

org.mockito:mockito-core:2.15.0
   variant "runtime"
\--- org.springframework.boot:spring-boot-starter-test:2.0.2.RELEASE
     \--- testCompile
```

Spring Boot:

- В версии 1.x по умолчанию использует Mockito 1
- Начиная с v1.5.2 допускает ручное включение Mockito 2
- В версии 2.x по умолчанию использует Mockito 2

Вводная

- Версии Mockito 2.0 и 2.1 вышли относительно давно
- Основные изменения:
  - Поддержка Java 8 (выведение типов)
  - Обход пересечений в Hamcrest
  - Учёт прошлых ошибок
- Обратно **не** совместимы с v1.x

Когда открыл Tests Report, а там…

Проходит успешно только в Mockito 1

```
@Test
 public void testAnyMatcher() {
   JButton buttonMock = mock(JButton.class);
   buttonMock.setName(null);
   verify(buttonMock).setName( isNull()    );
 }
```

Argument(s) are different! Wanted:
jButton.setName(<any string>);
-> at …
Actual invocation has different arguments:
jButton.setName(null);
-> at …

“We felt this change would make tests harness much safer that it was with Mockito 1.x”

Проходит успешно только в Mockito 1

```
public class MyService {
  public void setTarget(Object target) {
    // ...
  }
}
```

```
@T
pu                                          .class);

  verify(myServiceMock).setTarget(anyString());
}
```

Argument(s) are different! Wanted:
myService.setTarget(<any string>);
-> at ...
Actual invocation has different arguments:
myService.setTarget(javax.swing.JButton...);

Проходит успешно только в Mockito 1

```
public class MyService {
  public void callExternalSystem() {
    // ...
  }
}
```

```
@Test(expected = SocketTimeoutException.class)
public void testCheckedException() {
  MyService myServiceMock = mock(MyService.class);

  Class<? extends Throwable> exceptionClass = SocketTimeoutException.class;
  doThrow(exceptionClass).when(myServiceMock).callExternalSystem();
```

MockitoException: Checked exception is invalid for this method!

Осталось за кадром

- Несовместимость в compile-time
  `org.mockito.Matchers -> org.mockito.ArgumentMatchers`
- Несовместимость Mockito 1.x
  с `@MockBean` и `@SpyBean`
- Новый тестовый фреймворк
  в Spring Integration

https://docs.spring.io/spring-integration/docs/5.0.0.RELEASE/reference/htmlsingle/#testing

И как быть?

- Следовать старинным практикам Mockito: https://dzone.com/refcardz/mockito
- Переходить на Mockito 2 заранее (со Spring Boot 1.5.2+)
- Учитывать новшества версий 2.x:
  - https://asolntsev.github.io/en/2016/10/11/mockito-2.1
  - https://github.com/mockito/mockito/wiki/What%27s-new-in-Mockito-2

Gradle Plugin

Сборка Spring Boot проектов

Вводная

- Цитата из Migration Guide:

  «Spring Boot’s Gradle plugin has been largely rewritten…»

- Основное:
  - Нужен Gradle 4.0+

    Нужен хотя бы пустой файл `settings.gradle` в корне проекта
  - По умолчанию **не** подключает dependency management plugin
  - Задача `bootRepackage` заменена на две: `bootWar` и `bootJar`

Задача `bootJar`

- Активируется автоматически, если применены плагины
  - ✓ `org.springframework.boot`
  - ✓ `java`
- Отключает задачу `jar`
- Умеет находить `mainClassName` разными способами

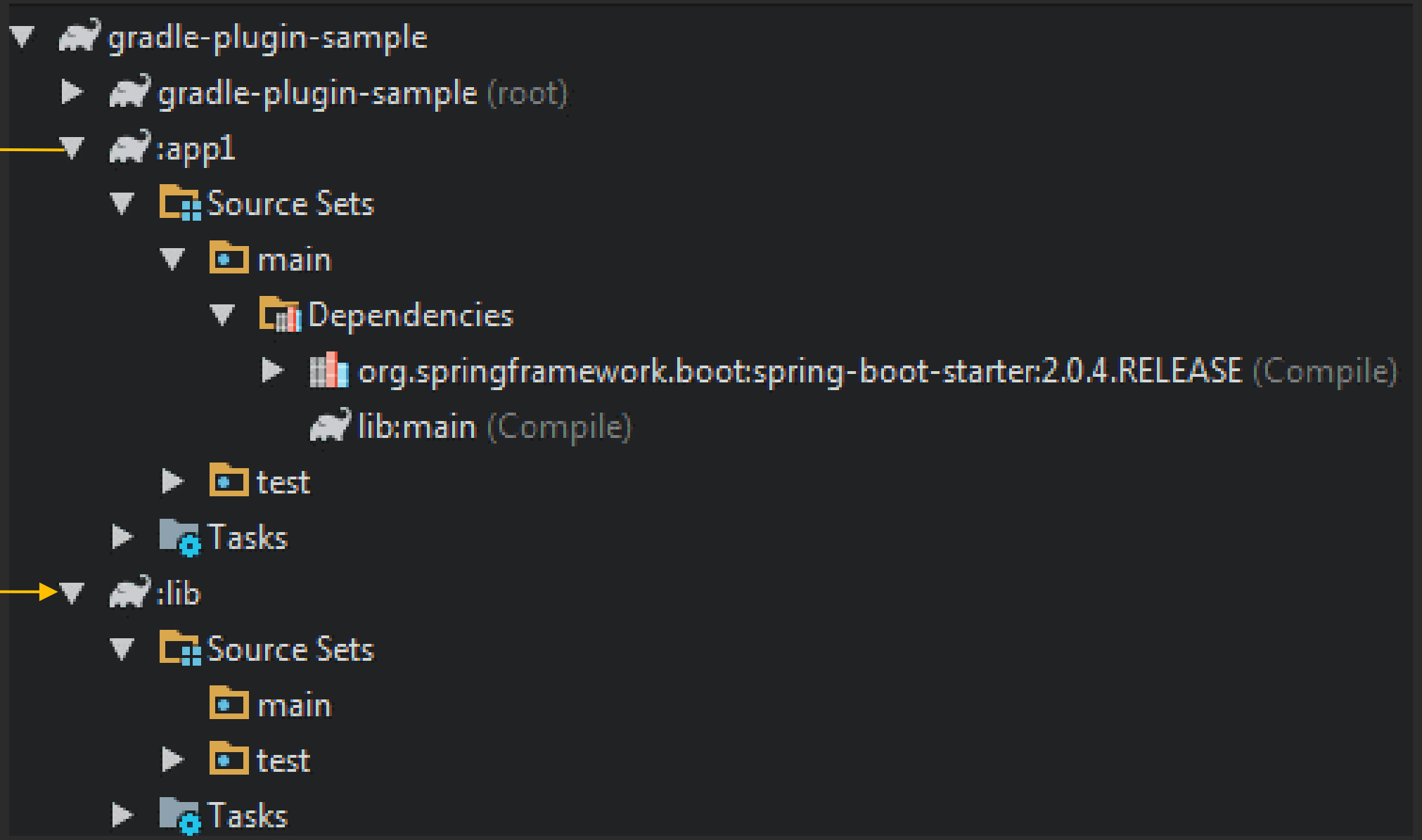
  И валит сборку, если всё-таки не нашла


НЕЛЬЗЯ ПРОСТО ТАК ВЗЯТЬ
И ЗАШАРИТЬ В GRADLE
PicsCom.com

Ну и что? Давай пример!

- Приложение Spring Boot 2.x
- Сборка на Gradle 4.x
- Использует Spring Boot Gradle Plugin 2.x
- Мультипроект: прикладной код + библиотека

app1
зависит от
lib



Show me the code!

Корневой проект

```
subprojects {
    repositories {
        mavenCentral()
    }
    apply plugin: 'java'
    apply plugin: 'org.springframework.boot'
}
```

Вариативность решений в Gradle

app1: скрипт сборки

```
dependencies {
    ext {
        springBootVersion = '2.0.4.RELEASE'
    }
    compile("org.springframework.boot:spring-boot-starter:$springBootVersion")
    compile project(':lib')
}
```

app1: исполняемый код

```
@SpringBootApplication
public class GradlePluginDemoApplication implements ApplicationRunner {

  // ...

  @Override
  public void run(ApplicationArguments args) {
    String appVersion = Util.getAppVersion(getClass());
    log.info("Current application version: {}", appVersion);
  }
}
```

lib: исполняемый код

```
public abstract class Util {

  public static String getAppVersion(Class<?> appClass) {
    return appClass.getPackage().getImplementationVersion();
  }
}
```

Скрипт сборки пуст.

./gradle build

```
> Task :app1:compileJava FAILED
app1\GradlePluginDemoApplication.java:9:
error: package tech.toparvion.sample.joker18.gradle.lib does not exist
import tech.toparvion.sample.joker18.gradle.lib.Util;
                                            ^

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app1:compileJava'.
```

Результаты расследования

Причины:

- `bootJar` глушит собою jar
- Gradle поставляет зависимости подпроектам на основе выхлопа от `jar`

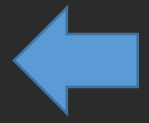
Следствия:

- Компилятор не может разрешить зависимость от библиотеки
- Все атрибуты манифеста, выставленные на задаче `jar`, игнорируются

Как быть ? (вариант 1)

Как было

```
subprojects {
  repositories {
      mavenCentral()
  }
  apply plugin: 'java‘


  apply plugin:
    'org.springframework.boot'
}
```



Как надо

```
subprojects {
  repositories {
      mavenCentral()
  }
  apply plugin: 'java‘
}
configure(subprojects.findAll { it.name != 'lib' }) {
  apply plugin:
    'org.springframework.boot'
}
```

Как быть? (вариант 2)

Применять SB Gradle Plugin только к Spring Boot подпроектам:

```
bootJar {
  enabled = false
}
```


МНЕ КАЖЕТСЯ
ОН НЕ ДОГОВАРИВАЕТ

Прочее: Spring Boot

- Изменились параметры конфигурации Spring Boot

  `server.display-name` → **`server.servlet.application-display-name`**

  ```
  i runtime group: 'org.springframework.boot',
    name: 'spring-boot-properties-migrator'
  ```

- Изменилась модель защиты доступа к методам Actuator’а

  `management.security.enabled=false` → **`management.endpoints.web.exposure.include=*`**

Прочее: Spring Cloud

- Переименовался артефакт клиента Feign

  `spring-cloud-starter-feign` →
  `spring-cloud-starter-openfeign`

- Переименовались артефакты Netflix (Eureka, Zuul, Hystrix, …)

  `spring-cloud-starter-eureka` →
  `spring-cloud-starter-netflix-eureka-client`

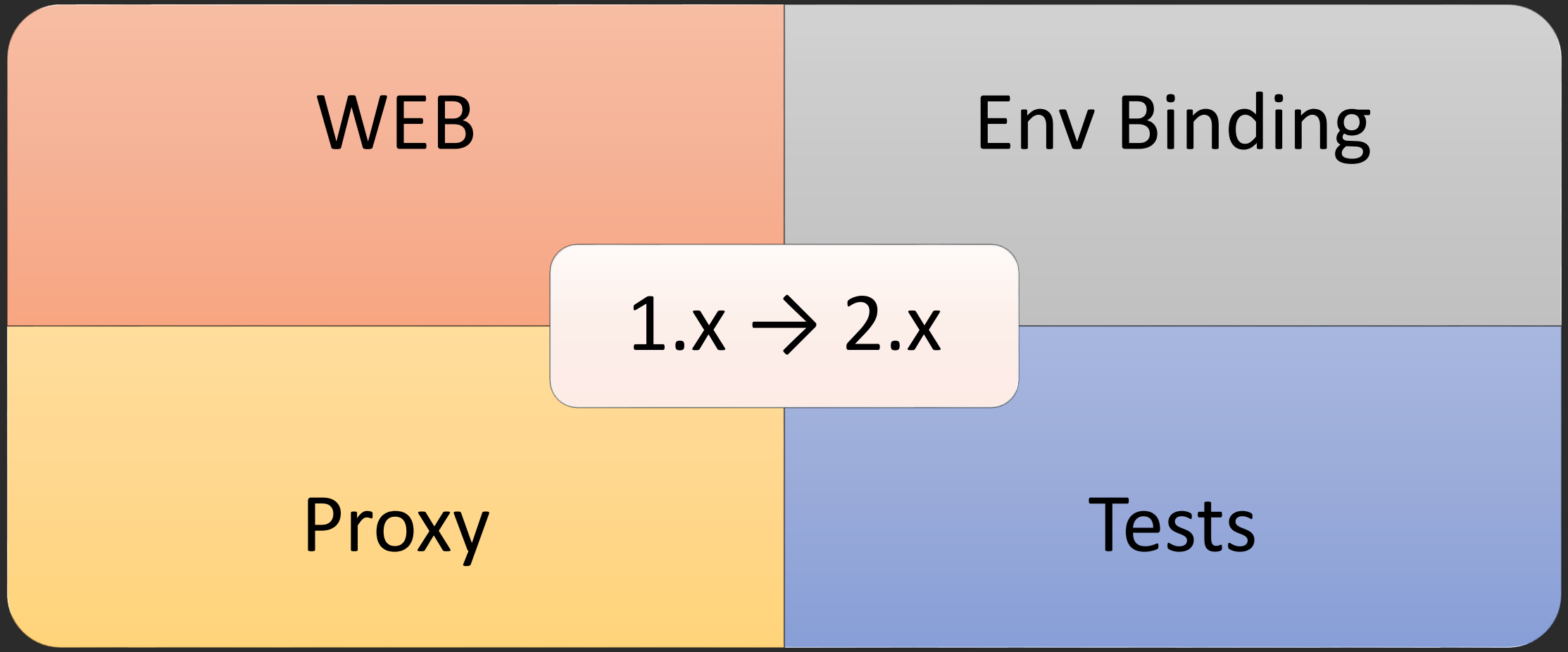
Прочее: Spring Integration (v5)

- Переехал Java DSL в **`spring-integration-core`**

- Перенесены inbound- и outboundAdapter’ы

```
.handleWithAdapter(f -> f.file(new File("work"))) →
.handle(Files.outboundAdapter(new File("work")))
```

Откуда ждать подвохов (резюме)



И как быть? (вообще)

- Пробовать «как есть» (YAGNI)
- Сверяться с образцами:
  https://github.com/Toparvion/joker-2018-samples
- Проверять обновления в Migration Guide
  https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide
- Смотреть другие грабледайджесты
  https://www.baeldung.com/new-spring-boot-2
  https://medium.com/@nucatus/spring-boot-2-x-migration-analysis-95f42bde402a

Spring Boot 2

Чего не пишут в release notes

Владимир Плизга́
ЦФТ
https://twitter.com/toparvion
vladimir.plizga@gmail.com

https://github.com/Toparvion/joker-2018-samples