

Микросервисы

Что должен уметь/знать программист для их написания

Who are you?

Big Data & Java Technical Leader

Mentoring

Consulting

Lecturing

Writing courses

Writing code

bsevgeny@gmail.com

@jekaborisov



Что попробуем успеть

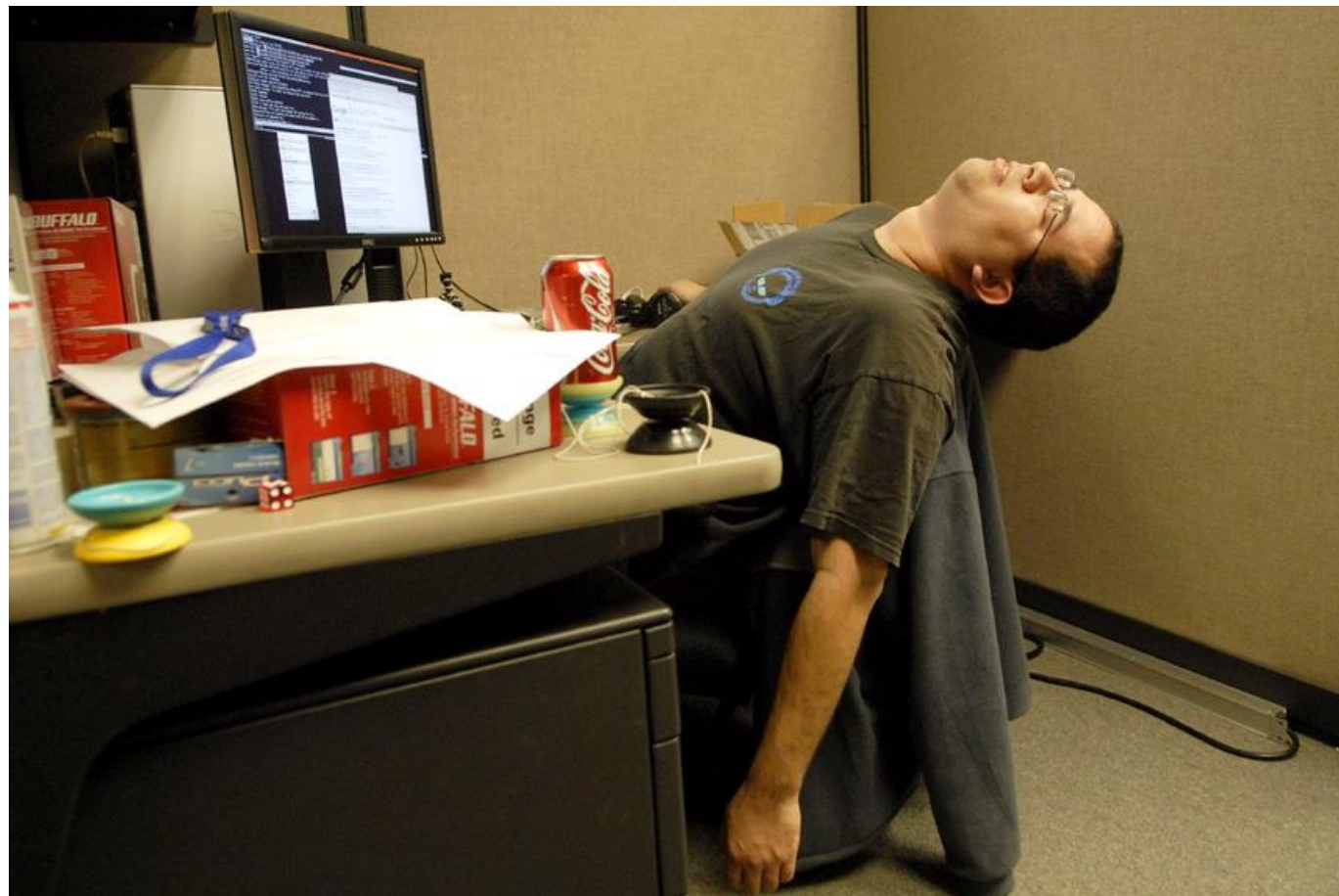
- Spring – надеюсь, что вы знаете
- История развития веба
- Spring MVC/WEB
 - web.xml и без него
 - Два контекста
- Spring Boot
- Spring Data – надеюсь вы в курсе
- Jackson / ObjectMapper
- Lombok – наследования зло, @Delegate rules
- Spring Data +
- Разбивка микросервиса на модули
- Заворачиваем rest в java api
- Тестируем контроллеры

Как будет проходить тренинг?

1. Часть –



Вторая часть



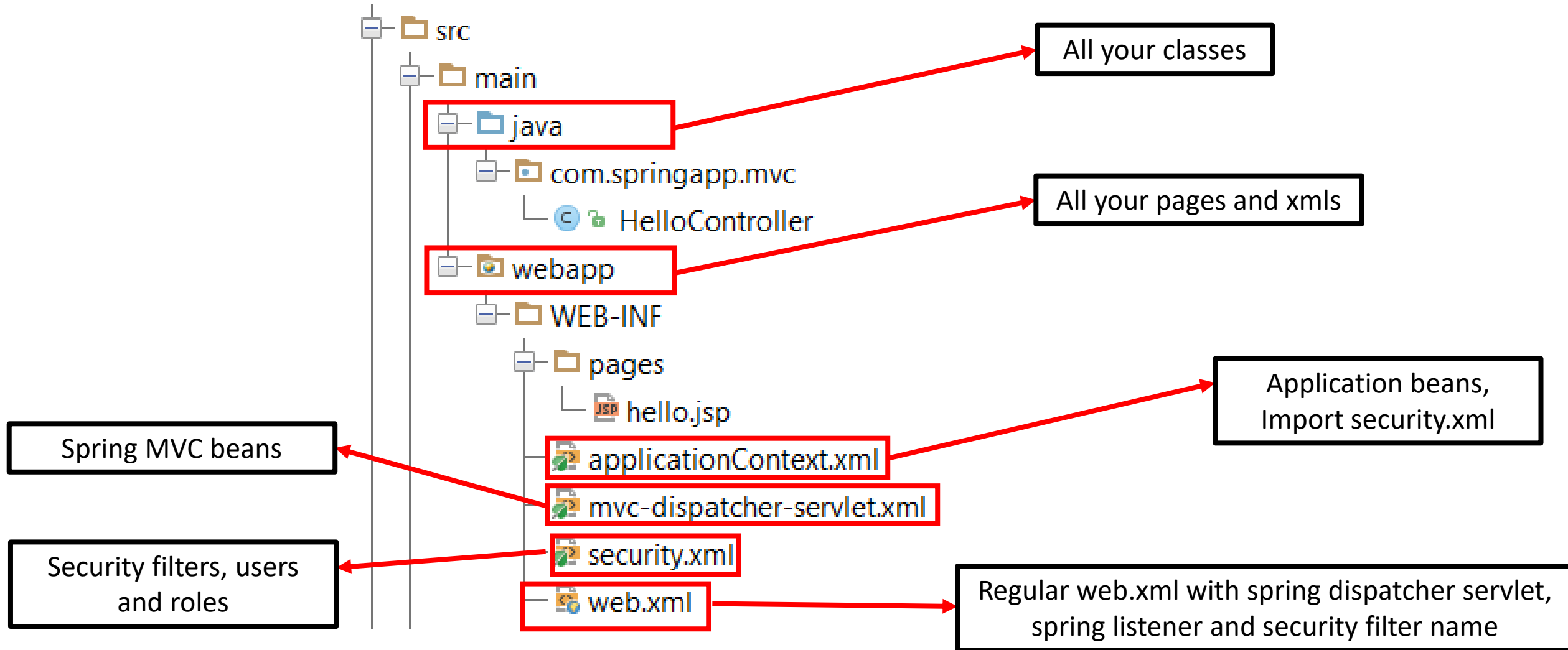
Поехали

Как писали раньше...



Смотрим...

The project structure



web.xml

```
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<servlet>
  <servlet-name>mvc-dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>mvc-dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

ONE DOES NOT SIMPLY

CHANGE THE CONVENTIONS

Avall from

```
<context-param>  
  <param-name>contextClass</param-name>  
  <param-value> ...AnnotationConfigWebApplicationContext</param-value>  
</context-param>
```

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>com.inwhite.conf.AppConfig</param-value>  
</context-param>
```

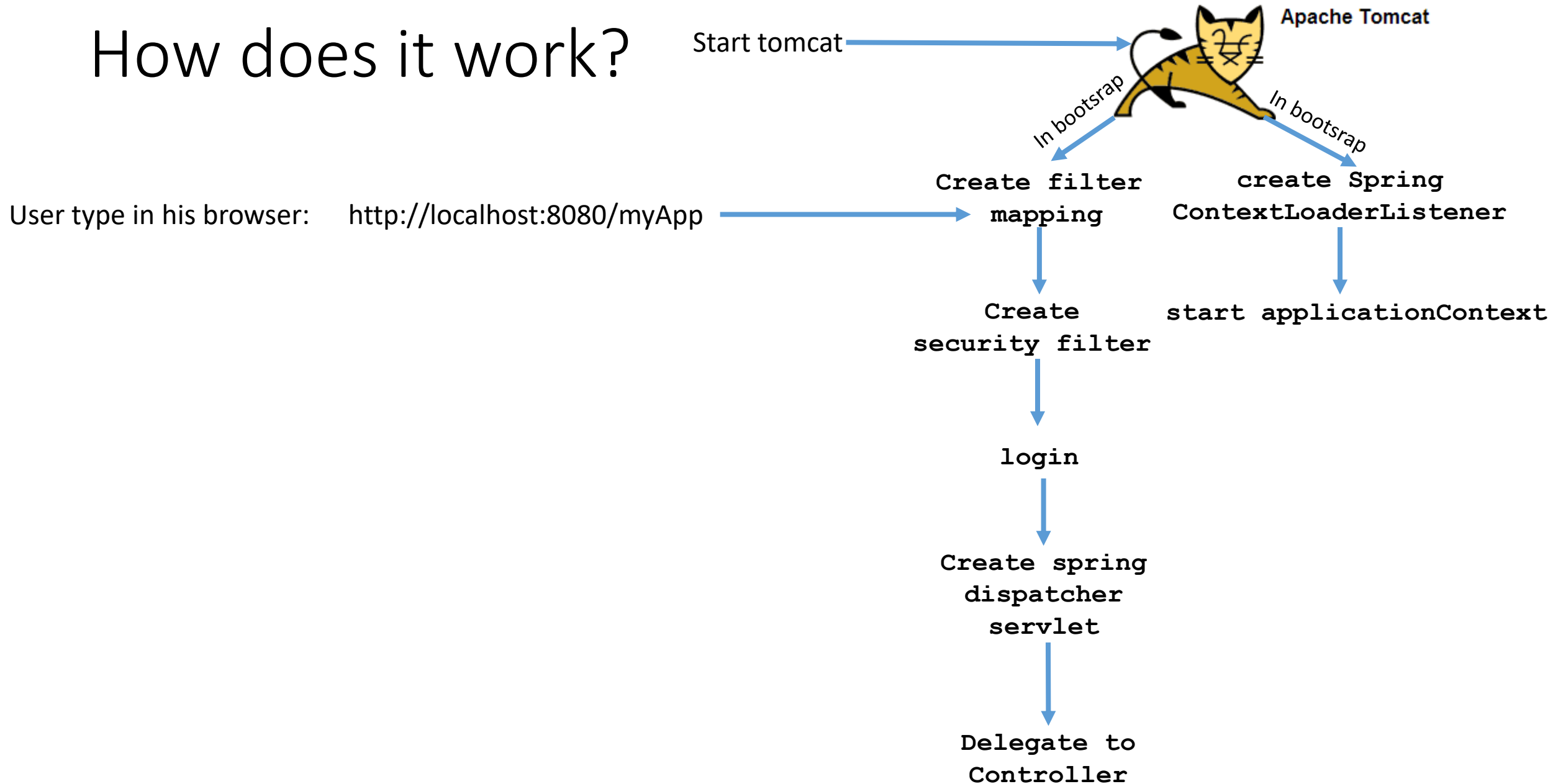
WE CAN



Deployment

- Use maven/gradle to build a war from that project
- Copy it to webapps directory of tomcat

How does it work?



Теперь давайте замочим XML



Servlet 3 API + Tomcat 7 – могут без web.xml

- Что больше нет сервлетов? А как тогда?
 - Вместо это делать это в специальном джава классе
- Что хорошего без XML-а?
 - Все более typesafe
 - Проще понимать ошибки конфигурации
 - Можно диктовать
 - Xml не модно

Как работаем без web.xml-а?

- Имплементируем `WebApplicationInitializer`, кстати чей это интерфейс?
- `ServletContainerInitializer` – это servlet 3.0, тот Спринга

** <p>Implementations of this interface must be declared by a JAR file
* resource located inside the <tt>META-INF/services</tt> directory and
* named for the fully qualified class name of this interface, and will be
* discovered using the runtime's service provider lookup mechanism
* or a container specific mechanism that is semantically equivalent to
* it. In either case, `ServletContainerInitializer` services from web
* fragment JAR files excluded from an absolute ordering must be ignored,
* and the order in which these services are discovered must follow the
* application's classloading delegation model.*

Так как это делать со спрингом?



Смотрим...

А теперь со спринг бутом и без томката



Смотрим...

Spring Boot – даёт нам 3 вещи

1. Dependency Management – решает конфликт зависимостей
Можно использовать как BOM
2. Стартеры (@EnableAutoConfiguration)
3. Плагин для мавена, который умеет собирать fat jar / war вместе с встроенным томкатом

Dependency Management

- Прописывает версии для зависимостей



Смотрим...

Немного у Спринг Буте

- Собирается в джар со всеми зависимостями
- В поме спринг буттового проекта есть их плагин
- Он ищет Мэйн и запускает его, если есть больше чем 1 ☹️

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Немного у Спринг Буте

- Собирается в джар со всеми зависимостями
- В поме спринг буттового проекта есть их плагин
- Он ищет Мэйн и запускает его, если есть больше чем 1 ☹️

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <mainClass>com.gameOfThrones.UsersApplication</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>
```


Немного у Спринг Буте

- Собирается в джар со всеми зависимостями
- В поме спринг буттового проекта есть их плагин
- Он ищет Мэйн и запускает его, если есть больше чем 1 ☹️

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <executable>true</executable>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Вот так должен выглядеть Main

```
@SpringBootApplication
public class UsersApplication {

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

@SpringBootApplication

- @Configuration
- @EnableAutoConfiguration - tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
- @ComponentScan
- Normally you would add @EnableWebMvc for a Spring MVC app, but Spring Boot adds it automatically when it sees spring-webmvc on the classpath

Application.properties

- Позволяет менять настройки инфраструктурных бинов.
- Можно иметь несколько под разными профилями
- Можно использовать yml



Вы можете прописать сами нужный бин

- Как это работает?
- `@Conditional`
- `@ConditionalOnMissingBean` (`OnBeanCondition`)
- `@ConditionalOnMissingClass` (`OnClassCondition`)

С чего начинают строить микросервис

- Контроллеры
- Сервисы
- Dao
- Model

С чего начинают строить микросервис

- Контроллеры
- Сервисы
- Dao
- **Model** – поскольку модель используется на всех слоях, пожалуй лучше начинать с неё

Data model - задание

- Придумайте архитектуру модели, которая позволит написать Сервис принимающий на вход объект покупки, и отдаёт на выход объект продажи

На вход – покупка

если скидка не пришла, default = -100

price = mandatory

```
{
  "customer": {
    "id": 1,
    "name": "Jeka",
    "age": 38
  },
  "price": 2000,
  "discount": 70,
  "when": [
    2017,
    4,
    4,
    2,
    26,
    36,
    774000000
  ],
  "product2Numbers": {
    "INTELIJ": 3,
    "ARTIFACTORY": 1
  }
}
```

На выход

```
{
  "salesman": {
    "id": 12,
    "name": "Dima",
    "department": "ACCESSORIES"
  },
  "price": 2000,
  "discount": 70,
  "when": [
    2017,
    4,
    4,
    2,
    40,
    8,
    774000000
  ],
  "product2Numbers": {
    "INTELIJ": 3,
    "ARTIFACTORY": 1
  }
}
```

Зло наследования



Композиция намного лучше



У композиции нет границ



Lombok – composition / delegate and friends

- Annotation Processor
- Работает на этапе компиляции
- Генерит код, чтобы мы не писали
- Делает джаву более похожим на нормальный язык

Что полезно знать про Lombok для написания модели и не только

- @Data – POJO (@Getter, @Setter, @ToString, @EqualsAndHashCode)
- @Value – immutable POJO
- @AllArgsConstructor

Что полезно знать про Lombok для написания модели и не только

- `@Data` – POJO (`@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`)
- `@Value` – immutable POJO
- `@AllArgumentConstruct(onConstructor = @_(@Autowired))`
- `@RequireArgumentConstructor` / `@NoArgumentConstructor`
- `@Builder` / `@Singular`
- `@Delegate` – примерно как в груви
- `@SneakyThrows`
- `@Slf4j` / `@Log4j` / ...

Теперь давайте разбираться Джэксоном



Как Джексон пишет в джейсон

- Чтобы вытащить данные Джексон пользуется геттерами.
- Точнее вызывает методы начинающиеся на `get` (case sensitive) и потом должен быть ещё хотя бы один символ
- Если мы не хотим, чтобы какой-то `getter` читался надо поставить `JsonIgnore` либо над геттером, либо над филдом, либо на сеттером
- Если нет ни одного геттера, всё упадёт

Как Джексон пишет в объект

- Если не конструктор помеченного `ConstructProperties`, будет сетить через сеттеры, там где их нет, то напрямую в филды, но при условии, что для них есть геттеры, но всё это есно при условии наличия пустого конструктора
- Если есть `ConstructProperties` то будет сеттить через конструктор.
- В любом варианте когда всё будет сделано зачем-то вызовет геттеры
- Но это дифолтное поведение

Меняем дифолтное поведение

- `JsonIgnoreProperties`
- `JsonIgnoreType`
- `JsonIgnore`
- `JsonIgnore`
- `JsonProperty`

А что не так с JSR 310?



С последних версий есть поддержка Java8

Если вы настраиваете руками:

```
ObjectMapper mapper = new ObjectMapper()
    .registerModule(new ParameterNamesModule())
    .registerModule(new Jdk8Module())
    .registerModule(new JavaTimeModule())
    ;
```

Или так

```
mapper.findAndRegisterModules();
```

Добавить зависимости

```
<dependency>
  <groupId>com.fasterxml.jackson.module</groupId>
  <artifactId>jackson-module-parameter-names</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jdk8</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jsr310</artifactId>
</dependency>
```

Если вы работаете с Spring Boot



Достаточно одних зависимостей

Как Spring MVC работает с Джексоном

- Напишите метод который принимает json и класс, а возвращает объект данного класса

@DateTimeFormat

- Ну вы в курсе...
- А ещё можно в application.properties

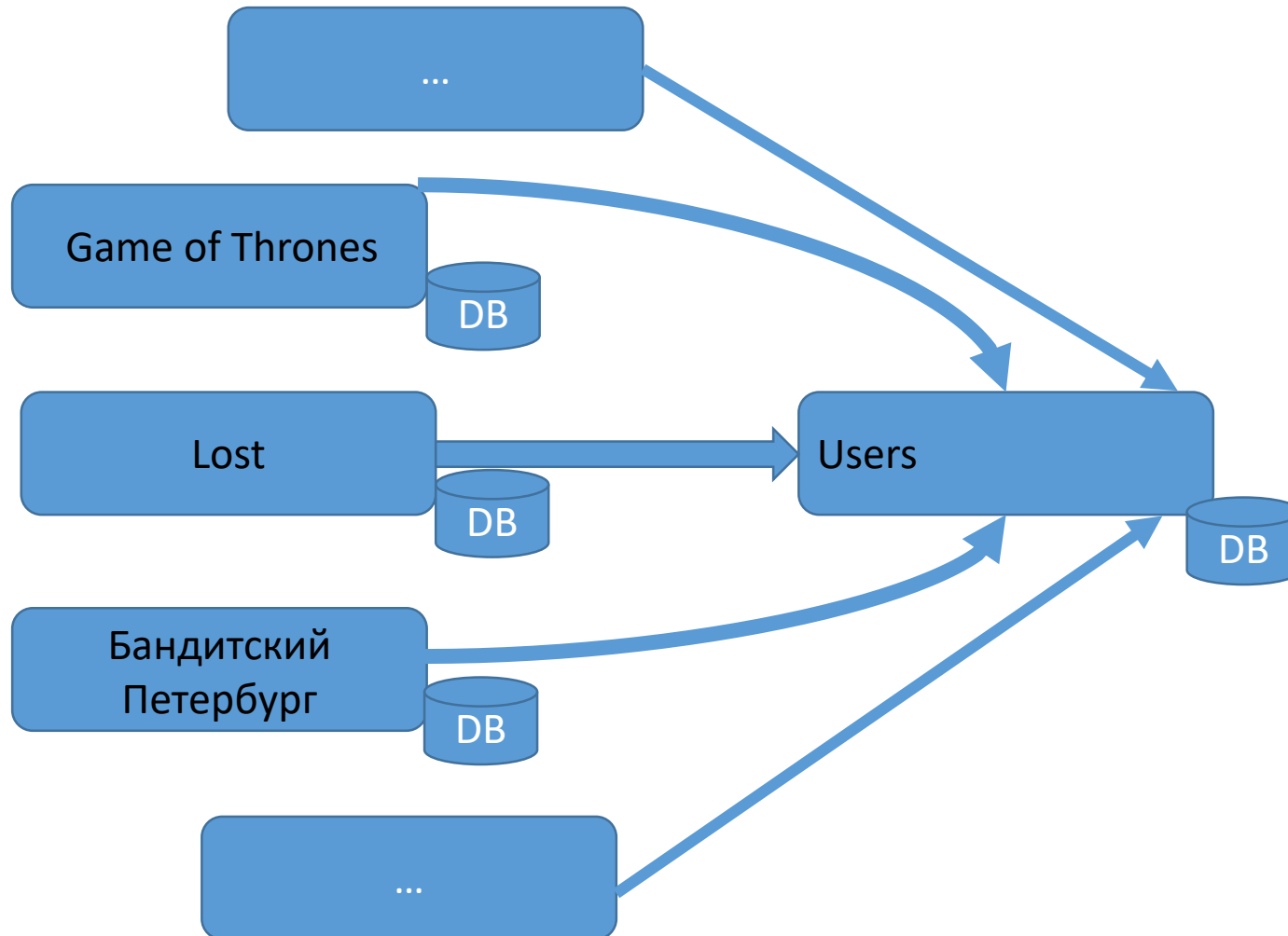
```
P spring.jackson.date-format (Date format string (yyyy-MM-d... String
P spring.jackson.default-property-inclusion (Controls the ... Include
.P spring.jackson.deserialization Map<DeserializationFeature, Boole...
.P spring.jackson.generator (Jackson on/off f... Map<Feature, Boolean>
P spring.jackson.joda-date-time-format (Joda date time form... String
P spring.jackson.locale (Locale used for formatting) Locale
.P spring.jackson.mapper (Jackson gener... Map<MapperFeature, Boolean>
.P spring.jackson.parser (Jackson on/off feat... Map<Feature, Boolean>
P spring.jackson.property-naming-strategy (One of the const... String
.P spring.jackson.serialization ... Map<SerializationFeature, Boolean>
P spring.jackson.time-zone (Time zone used when formatting TimeZone
```

Кстати о сериалайзерах



Давайте как то без них, по возможности

Пишем систему микросервисов для рекомендаций по сериалам



User model

```
{  
  "id": 12,  
  "name": "Vasya",  
  "age": 12,  
  "recommendations": [  
    "Lost",  
    "Prison break"  
  ],  
  "priorities": {  
    "drama": 70,  
    "action": 90  
  }  
}
```

Начинаем с Users

- Мы потом разобьём его на модули
- Написать контроль поверх CRUD операций на Юзера
 - Добавить Юзера
 - Стереть Юзера
 - Проапдэйтить Юзера
 - Пропатчить Юзера
 - Получить всех Юзеров
 - Получить Юзера по айдишнику
 - Получить всех Юзеров старше 18 лет
- База данных по вкусу, у меня Монго

```
{  
  "id": 12,  
  "name": "Vasya",  
  "age": 12,  
  "recommendations": [  
    "Lost",  
    "Prison break"  
  ],  
  "priorities": {  
    "drama": 70,  
    "action": 90  
  }  
}
```

Spring Data Rest

Можно конечно так...

```
<dependency>
```

```
  <groupId>org.springframework.data</groupId>
```

```
  <artifactId>spring-data-rest-webmvc</artifactId>
```

```
  <version>2.5.5.RELEASE</version>
```

```
</dependency>
```


Но мы наверное пойдём таким путём

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-rest</artifactId>  
</dependency>
```

Дальше пишем Entity + Repository

- Which repositories get exposed by defaults?

Name	Description
DEFAULT	Exposes all public repository interfaces but considers <code>@(Repository)RestResource</code> 's <code>exported</code> flag.
ALL	Exposes all repositories independently of type visibility and annotations.
ANNOTATION	Only repositories annotated with <code>@(Repository)RestResource</code> are exposed, unless their <code>exported</code> flag is set to false.
VISIBILITY	Only public repositories annotated are exposed.

Как поменять стратегию

```
@Component
public class MyWebConfiguration extends RepositoryRestConfigurerAdapter {

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {
        config.setRepositoryDetectionStrategy(ALL);
    }
}
```

- Можно конечно

Как задать URL?

```
@Configuration
class CustomRestMvcConfiguration {

    @Bean
    public RepositoryRestConfigurer repositoryRestConfigurer() {

        return new RepositoryRestConfigurerAdapter() {

            @Override
            public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {
                configuration.setBasePath("/api")
            }
        };
    }
}
```

- Или в том же RepositoryRestConfigurerAdapter прописать
- Но проще:
в application.properties - spring.data.rest.basePath=/api

Spring Data Rest official supports:

- [Spring Data JPA](#)
- [Spring Data MongoDB](#)
- [Spring Data Neo4j](#)
- [Spring Data GemFire](#)
- [Spring Data Cassandra](#)

@RepositoryRestResource is not required for a repository to be exported. It is only used to change the export details, such as using /people instead of the default value of /persons.

```
@RepositoryRestResource(collectionResourceRel = "people", path = "people")
public interface PersonRepository extends MongoRepository<Person, String> {
    @RestResource(path = "byname")
    List<Person> findByLastName(@Param("name") String name);
}
```

URL's и методы

- localhost:8080/people – список всех (GET)
- localhost:8080/people/1 – дай человека с айдишником 1 (GET)
- localhost:8080/people/1 – стереть с айдишником 1 (DELETE)
- localhost:8080/people/1 – заменить с айдишником 1 (PUT)
- localhost:8080/people/1 – проапдэйтить с айдишником 1 (PATCH)
- localhost:8080/people/search/findByName?name=Lanister

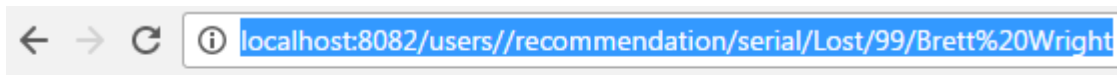
Добавляем бизнес логику

- По обращению к микросервису можно получить информацию про конкретный сериал, сколько всего Юзеров его видело и сколько из них его советуют

`http://localhost:8082/users/recommendation/count/serial/Lost`

```
{ "positiveUsers": 15, "totalUsers": 20 }
```

- Можно передать в микросервис своё имя, интересующий тебя сериал и **число** до 100. Микросервис найдет всех людей похожих на того, чьё имя пришло (**число** это на сколько похожих по вкусам) и даст рекомендацию от имени тех из них, кто смотрел этот сериал. (Для этого надо просто проверить есть ли имя этого сериала у них в рекомендациях



← → ↻ ⓘ localhost:8082/users//recommendation/serial/Lost/99/Brett%20Wright

true

Разбиваем микросервис на модули



Кому отсыпать
наносервисов?

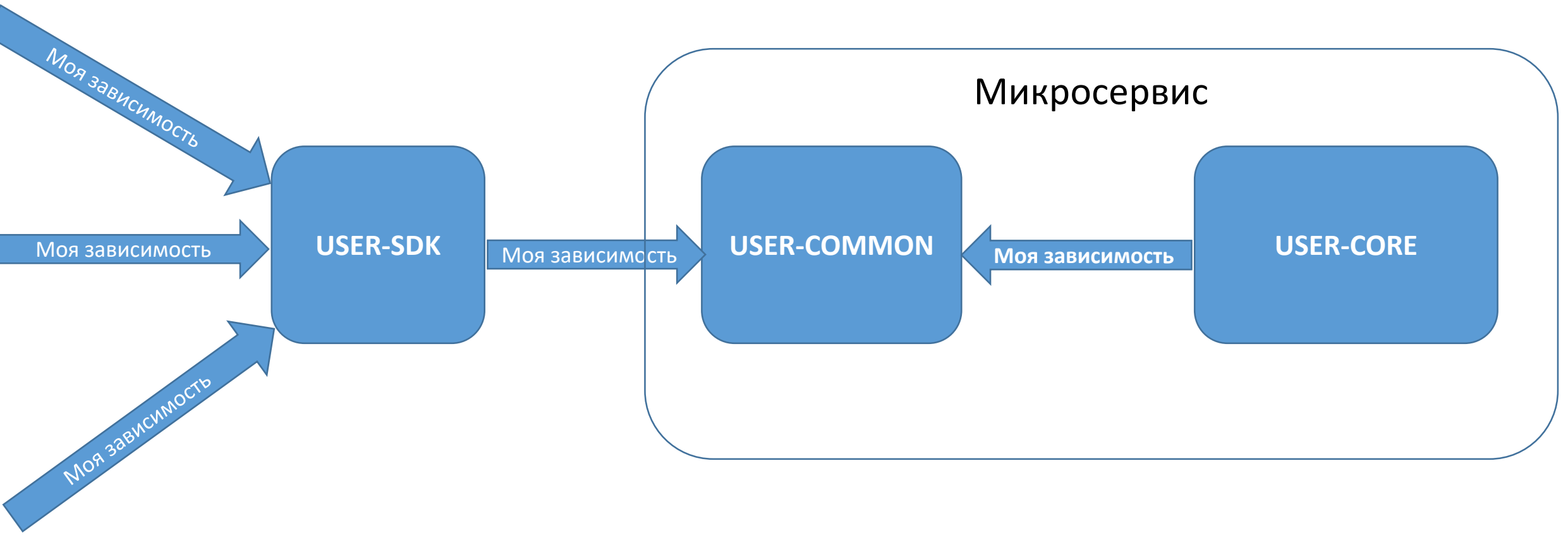
Зачем разбивать микросервис на модули?

- Это нужно делать когда мы хотим иметь джававский rest api которых могут использовать другие приложения или микросервисы добавив зависимость на этот API

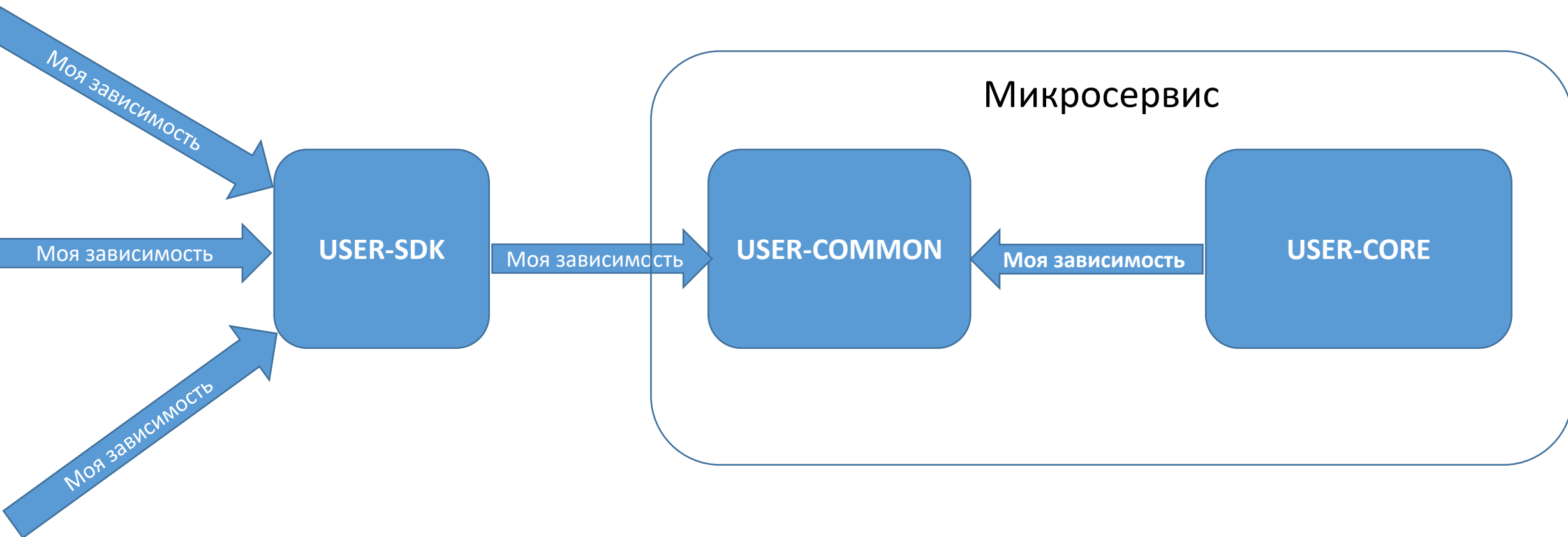
Модули микросервиса

- `microservice-core` - тут все контроллеры и сервисы
- `microservice-sdk` - сервисы, которые при помощи `RestTemplate` или другой альтернативы обращаются к контроллерам данного микросервиса (есно этот джар сделан для других, ему самому он нафиг не нужен)
- `microservice-common` – модель, которая используется в `core` и в `sdk`

Зависимости между модулями

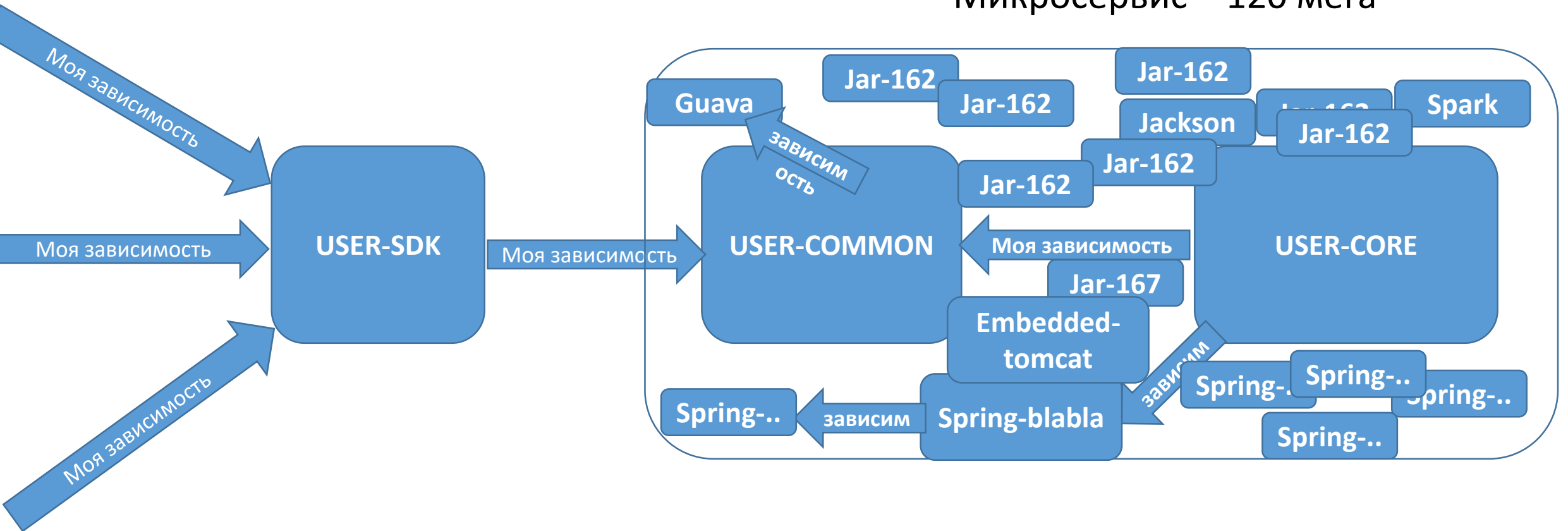


Это утопия



Это реальность

Микросервис – 120 мега



Давайте тестировать

- Unit тесты (в рамках одного класса)
Ничего не надо, только JUnit + mocks. Тестируемый объект строим через new.
- Компонент тесты (смесь юнит + интеграционный. Тестируем кусок системы, который работает вместе, как одно целое, остальное моки).
@RunWith(SpringRunner)+тестовая конфигурация. Тестируемый объект строится Спрингом
- Микросервис тест
Тестируем end to end микросервис, моки только на sdk других микросервисов + wire mock на все остальные внешние ресты

Какие аннотации нам помогут

- `@ContextConfiguration` – для указания конфигурации
- `@SpringBootTest` – в двух словах не объяснишь, рассказывать буду
- `@ContextHierarchy`
- `@WebMvcTest` – для тестирования контроллеров
- `@DataJpaTest` – догадайтесь. Как `SpringBootTest`, только поднимаются репозитории SpringData + тестовый entity manager
- `@TestConfiguration` – для тестовых конфигураций, которые не должны влиять на другие тесты
- `@ContextHierarchy` – для кэширования тестовых контекстов
- `@DirtyContext` – для пересоздания контекста после запуска теста
- `@MockBean` / `@SpyBean`

@SpringBootTest

- Применяем **@SpringBootTest**
- Долго...
- **@SpringBootTest(classes = ...class)**
- Стало быстрее
- С кэшированием конфигураций – еще быстрее

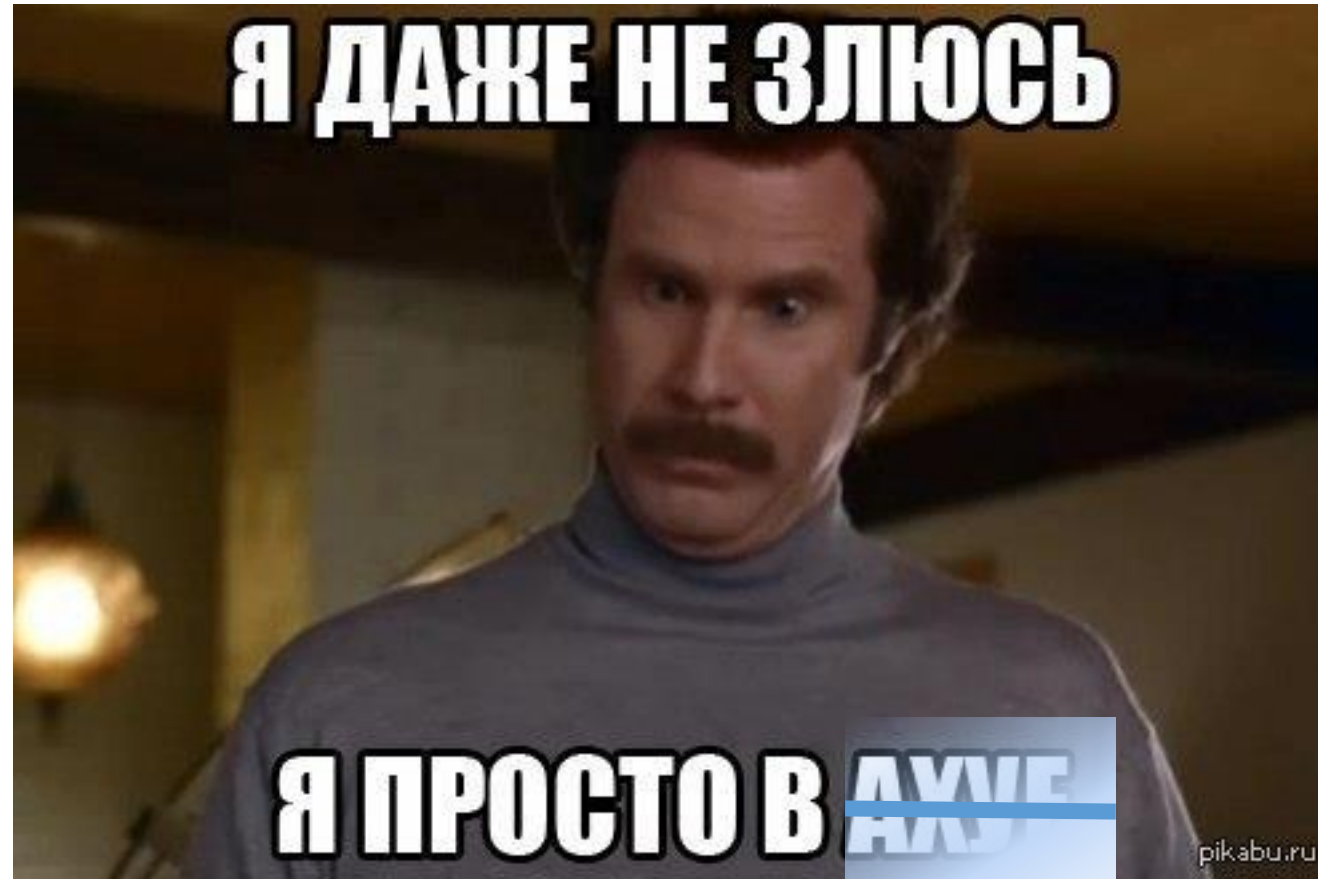
Cache configuration

```
@ContextHierarchy({  
@ContextConfiguration(classes=CommonConfiguration.c  
lass),  
@ContextConfiguration(classes= ...class)  
})
```

Порядок важен! Инициализация конфигураций происходит в том порядке, в котором мы это указано и если первая конфигурация использует бины из второй, то всё упадёт.

Их надо поменять местами

Ничего не кэшируется



Чтобы кэшировалось надо:

- **@SpringBootTest** – должен быть **езде**
- **@Import** – должен быть **нигде**
- **@ActiveProfiles** – **один** на всех
- **SpringBootTest.properties** – должны быть **одинаковые**

Вот так не закешируется

- **@SpringBootTest(properties={"a=b","b=a"})**
- **@SpringBootTest(properties={"b=a","a=b"})**

Кэш конфигураций штука хрупкая, может
посыпаться



logging.level.org.springframework.test.context.cache=debug

Вот наша суперсила



SpringBootTest

- SpringBootConfiguration / SpringBootApplication
- webEnvironment
- RANDOM_PORT

WebMvcTest и его друзья

- Не создаёт никаких бинов, кроме указанных или всех контроллеров
- Не поднимает томкат, чтобы общаться к контроллером удобнее всего пользоваться MockMvc, который можно заинжектировать в тест
- Для ассертов есть удобные методы у следующих классов
 - MockMvcRequestBuilders
 - MockMvcResultMatchers
 - ResultActions

Два сканирования

- **@SpringBootTest** сканирование
- **@SpringBootApplication (@ComponentScan)**



- Причём сканируются не только тесты, но и сорсы

ЧИНМ

@SpringBootConfiguration

public class **StopConfiguration** {}

Вот вам пример

```
mockMvc.perform(get("/users/search/byname?name=Bilbo")
                .accept(MediaType.APPLICATION_JSON))
    .andExpect(status().isOk())
    .andExpect(jsonPath("$.family", equalTo("Baggins")))
    .andExpect(jsonPath("$.age", equalTo(111)));
```

МеждуМикросервисный End to End test

- Или делать внешнюю тестирующую систему
- Или пользоваться SpringBootTest, а не WebMvcTest
- Пользоваться не MockMvc а TestRestTemplate
- Если в процессе теста будет использован не мок, а настоящий SDK то моковый web environment не катит

Выводы

- Spring для Unit тестирования может быть быстрым
- Кэш контекстов – хрупкая штука
- Для тестов – только **@TestConfiguration**
- Изолировать группы тестов с помощью
 - выделения в пакеты
 - **@SpringBootTestConfiguration**
- SpringBootTest надо в основном использовать для микросервис тестов
- Если есть **DirtyContext** – стоит задуматься :)

А что с Асинхронными реквестами?

DeferredResult