

09. Событийно-ориентированная модель + Сборка проекта

Лекции по информатике
для студентов второго курса Высшей школы ИТИС КФУ
2020

Ференец Александр Андреевич

старший преподаватель кафедры программной инженерии

С использованием материалов
к. т. н., доцента кафедры программной инженерии Абрамского М.М.

aferenets@it.kfu.ru

СБОРКА ПРОЕКТА. Зачем?

Что может быть в проекте?

1. Исходники-классы на Java
2. Библиотеки-зависимости
3. Файлы с переменными среды (properties и прочая конфигурация для разных машин)
4. Статические ресурсы
 - а. Веб: html, изображения, стили, медиа, *простой js*
 - б. Системное: изображения, стили, медиа, ~~простой js~~ *простые скрипты (lua?)*
5. Конфигурация для разворачивания/работы проекта (web.xml)

СБОРКА ПРОЕКТА. Зачем?

Что может быть в проекте?

1. Исходники-классы на Java
2. Библиотеки-зависимости
3. Файлы с переменными среды (properties и прочая конфигурация для разных машин)
4. Статические ресурсы
 - а. Веб: html, изображения, стили, медиа, *простой js*
 - б. Системное: изображения, стили, медиа, *простой js* простые скрипты (lua?)
5. Конфигурация для разворачивания/работы проекта (web.xml)
6. Прекомпилируемые стили (SASS, LESS)
7. JS: код, библиотеки-зависимости
8. Код на Python (PHP, Kotlin, Go)
9. ...

Почитайте.
Интересно

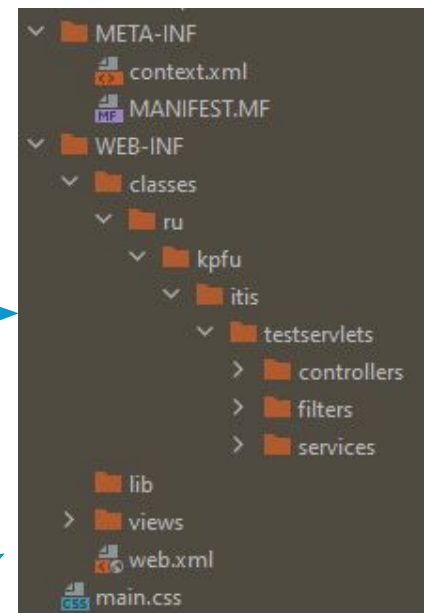
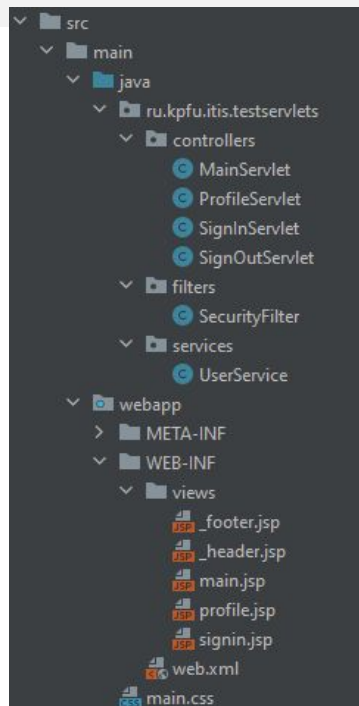
Клиентская часть веб-приложения может быть сложной

Потому что удобно разбивать программу на части и у разных ЯП удобно решать разные задачи

А ещё...

1. Файлы для создания комфортной среды разработки
2. Файлы для организации тестирования (автоматические и проч)

Не путайте **артефакт/релиз** и **проект разработки**! У них может не совпадать вид, структура. Чтобы превратить проект в релиз часто нужно сделать ряд действий (компиляция, перераспределение файлов, подготовка ресурсов, БД).



← вот так удобнее разрабатывать
вот так требует стандарт сервера →

СБОРКА ПРОЕКТА. Что можно делать?

1. Компиляция
2. Запуск тестирование
3. Менеджмент зависимостей
4. Расположение файлов в нужной структуре
5. Развёртывание (deploy)
6. Вопросы ведения проекта
(менеджмент задач, менеджмент версий)

СБОРКА ПРОЕКТА. Что предлагают?



maven

+
Make
Composer
NPM
Webpack

“Another Neat Tool”

Аналог make

Императивный подход

Targets – цели (какой именно процесс сборки выполняется).

Примеры:

- build – компиляция и создание jar/war,
- clean – удаление временных файлов,
- deploy – развертывание,
- ...

Tasks – задания, выполняемые в рамках целей.

Примеры:

- javac – компиляция java-файлов,
- copy – копирование файлов,
- exec – выполнение внешней команды,
- ...

ANT. Файл описания targets

```
<target name="compile" depends="prepare"
        description="Compile the servlet">
    <echo message="Compiling the Java file "/>
    <echo message="${compiled.servlet}.java"/>
    <javac srcdir="${src}" destdir="${build}"
        <include name="${compiled.servlet}.java" />
        <classpath refid="servlet-classpath"/>
    </javac>
</target>

<target name="deploy-servlet" depends="compile">
    <echo message="Copying the servlet to Tomcat web app"/>
    <copy todir="${tomcat.webapps}/WEB-INF/classes">
        <fileset dir="${build}" />
    </copy>
</target>
```

Jason van Zyl 2004 - 1.0
2010 - 3.0

Ant → императивный стиль

Maven → декларативный стиль



Project Object Model

```
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

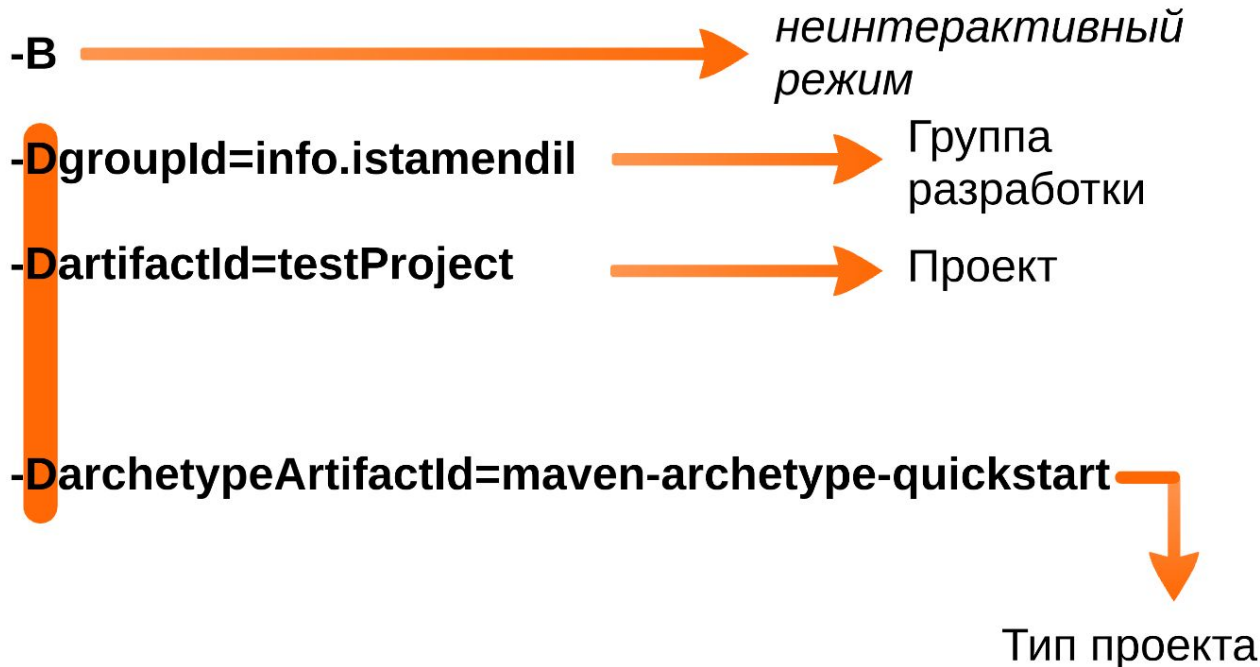
  <groupId>info.istamendil</groupId>
  <artifactId>testProject</artifactId>
  <version>1.0-SNAPSHOT</version>

</project>
```

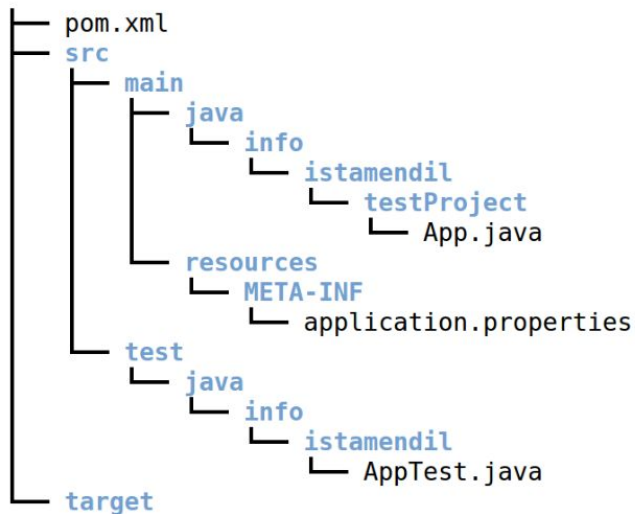
maven.apache.org

- *Автоматическая компиляция*
- *Слежение за зависимостями*
- *Работа с тестами*
- *Система плагинов*
- *Wagon (ftp, ssh)*
- *Source Code Management (svn, git, ...)*
- *Ant Tasks*

mvn archetype:generate

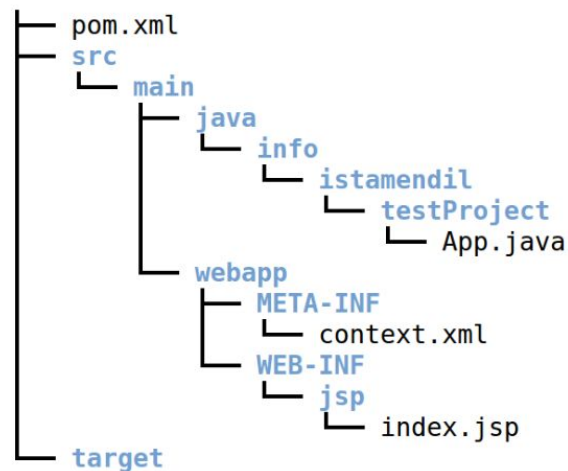


Простой проект *




* с ресурсами и тестами

Веб-проект **



** без доп. ресурсов и тестов

mvn ...

- 
- **validate** — проверяет корректность метаданных о проекте
 - **compile** — компилирует исходники
 - **test** — прогоняет тесты классов из предыдущего шага
 - **package** — упаковывает скомпилированные классы в удобноперебрасываемый формат (jar или war, к примеру)
 - **integration-test** — отправляет упакованные классы в среду интеграционного тестирования и прогоняет тесты
 - **verify** — проверяет корректность пакета и удовлетворение требованиям качества
 - **install** — загоняет пакет в локальный репозиторий, откуда он будет доступен для использования как зависимость в других проектах
 - **deploy** — отправляет пакет на удаленный production сервер, откуда другие разработчики его могут получить и использовать

Этапы связаны и знают о предыдущих:

test выполнит **validate** и **compile**



+ clean

MAVEN. Файл описания проекта

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project
3      xmlns="http://maven.apache.org/POM/4.0.0"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
6      <modelVersion>4.0.0</modelVersion>
7
8      <groupId>ru.kpfu</groupId>
9      <artifactId>servletExampleThird</artifactId>
10     <packaging>war</packaging>
11     <version>1.0</version>
12
13     <name>Servlet Example: Third</name>
14
15     <properties>
16         <!-- Generic properties -->
17         <maven.compiler.source>1.8</maven.compiler.source>
18         <maven.compiler.target>1.8</maven.compiler.target>
19         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
20         <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
21     </properties>
22
23     <dependencies>
24         <!-- Servlet + EJB Lite + JPA + JSP + EL + JSTL + JSF + JTA + JSR-45 + JSR-250 -->
25         <dependency>
26             <groupId>javax</groupId>
27             <artifactId>javaee-web-api</artifactId>
28             <version>8.0.1</version>
29             <scope>provided</scope>
30         </dependency>
31     </dependencies>
32 </project>
33
```

GRADLE. Структура проекта

```
├── buildSrc
│   ├── build.gradle
│   └── src
│       ├── main
│       │   └── java
│       │       └── com
│       │           └── enterprise
│       │               ├── Deploy.java
│       │               └── DeploymentPlugin.java
│       └── test
│           ├── java
│           │   └── com
│           │       └── enterprise
│           │           └── DeploymentPluginTest.java
├── settings.gradle
├── subproject-one
│   └── build.gradle.kts
└── subproject-two
    └── build.gradle.kts
```

```
├── build.gradle
└── src
    └── main
        ├── java
        │   └── HelloWorld.java
        └── kotlin
            └── Utils.kt
```



```
plugins {  
    id 'java'  
    id 'application'  
}  
repositories {  
    mavenCentral()  
}  
dependencies {  
    testImplementation('org.junit.jupiter:junit-jupiter:5.6.0')  
}  
test {  
    useJUnitPlatform()  
    testLogging {  
        events "passed", "skipped", "failed"  
        showExceptions true  
        exceptionFormat "full"  
        showStandardStreams = false  
    }  
}  
application {  
    mainClass = project.hasProperty("mc") ? getProperty("mc") : "NULL"  
}
```

Структурное описание на Groovy/Kotlin

Event Driven Development

Парадигма программирования, в которой ход выполнение программы определяется сообщениями (событиями) от программных модулей, пользователя и других ИСТОЧНИКОВ.

```
const http = require('http');

const server = http.createServer((request, response) => {
  // magic happens here!
});
```

Идеально для скриптовых языков. *А что такое скриптовые языки?*

Выглядит очень просто!

1. Когда запускать?
2. Что запускать?

```
RouterFunctions
    .route(GET("/person/{id}").and(accept(APPLICATION_JSON)), request -> {
        int personId = Integer.valueOf(request.pathVariable("id"));
        Mono<ServerResponse> notFound = ServerResponse.notFound().build();
        return repository.findOne(personId)
            .then(person -> ServerResponse.ok().body(Mono.just(person), Person.class))
            .otherwiseIfEmpty(notFound);
    })
    .andRoute(GET("/person").and(accept(APPLICATION_JSON)), request ->
        ServerResponse.ok().body(repository.findAll(), Person.class))
    .andRoute(POST("/person").and(contentType(APPLICATION_JSON)), request ->
        ServerResponse.ok().build(repository.save(request.bodyToMono(Person.class))));
```

Особенность

Обработчики должны быть простыми! Иначе код легко превратить в кашу.

Контроллеры в MVC так-то тоже должны быть простыми... Так что условность.

Проблемы

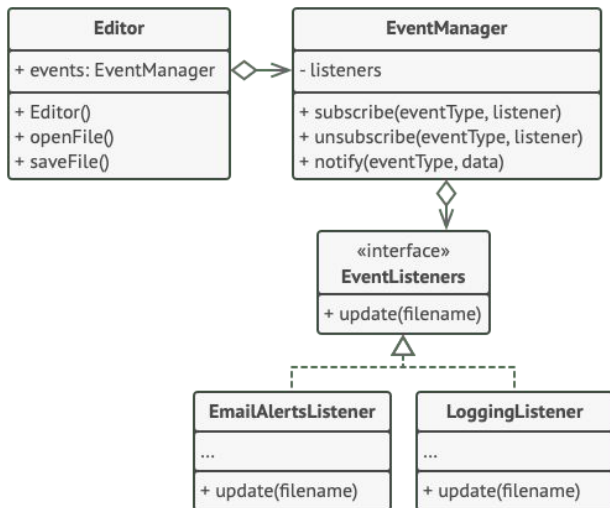
- Как регулируется выполнение обработчиков?
- Есть ли потоки? Есть ли приоритет реагирования?
- Как делиться данными, как организовывать модульность?

```
document.getElementById('signin-button').onclick =  
    function(e) {console.log(e.targetSource);} ;
```

Функцию удобно записывать в переменную. Выше свойство объекта. Какая разница?

```
action = function(e) {console.log(e.targetSource);} ;
```

EDD. Шаблон проектирования Наблюдатель



В Java уже есть интерфейсы Observer, EventListener, Event, но можно сделать свой интерфейс.

Всё должно быть аккуратно! Нужно создавать менеджеры и всяческие хранилища (пулы).
Это часто обеспечивается стандартными средствами подписки на события, отложенного старта и проч.

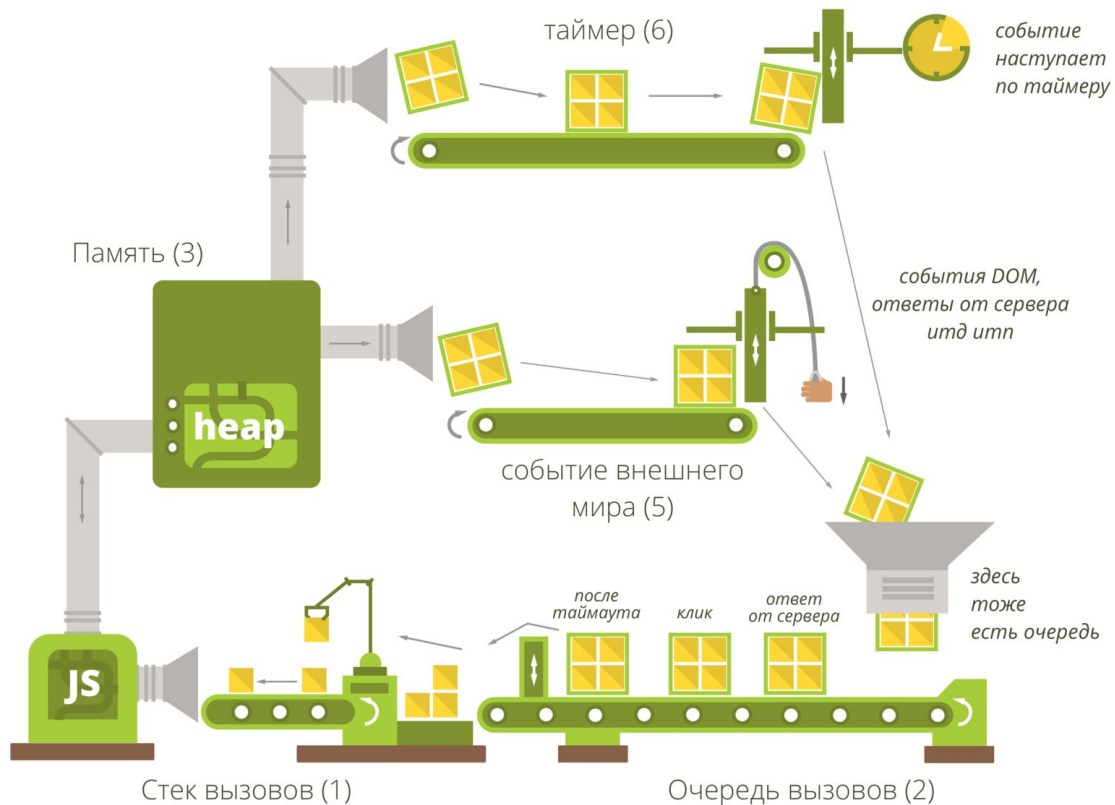
```
doNetworkRequestButton.addActionListener(e -> {
    SwingWorker worker = new SwingWorker<String, Integer>() {
        protected String doInBackground() throws Exception {
            HttpResponse response = doRequest();// Do long network request here
            return response.getBody();
        }

        protected void done() {
            try {
                respLabel.setText("Data has been updated:" + get());
            }
            catch (InterruptedException | ExecutionException ex){
                respLabel.setText("Error has been occured:" + ex.getMessage());
            }
        }
    };
    worker.execute();
});
```

```
Service<Void> service = new Service<Void>() {
    @Override
    protected Task<Void> createTask() {
        return new Task<Void>() {
            @Override
            protected Void call() throws Exception {
                //Background work
                final CountDownLatch latch = new CountDownLatch(1);
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        try{
                            //FX Stuff done here
                        }finally{
                            latch.countDown();
                        }
                    }
                });
                latch.await();
                //Keep with the background work
                return null;
            }
        };
    }
};

service.start();
```

ОТЛОЖЕННЫЙ/СОБЫТИЙНЫЙ ЗАПУСК. Реализация в JS



That's all

Лекции по информатике
для студентов второго курса Высшей школы ИТИС КФУ
2020

Ференец Александр Андреевич

старший преподаватель кафедры программной инженерии

С использованием материалов
к. т. н., доцента кафедры программной инженерии Абрамского М.М.

aferenets@it.kfu.ru