

# Resource Handling in Spring MVC



**박용권** SK planet

: 봄싹(SpringSprout)  
: 한국 스프링 사용자 모임(KSUG)  
: 라 스칼라 코딩단

: twitter / @arawnkr

Spring MVC에서  
정적 자원(css, js, etc)을  
처리해본 경험을 공유합니다.

# Spring MVC : 리소스 제공(Serving)

## ✓ ResourceHttpRequestHandler

- : URL 패턴에 따라 정적 자원 요청을 처리

- : `org.springframework.core.io.Resource` 인터페이스를 사용  
servletcontext, classpath, filesystem, etc

- : HTTP 캐시 설정 기능 제공

  - expires 또는 cache-control를 사용한 캐시 설정

  - last-modified 헤더 평가를 통해 304 상태코드 응답

## ✓ 설정 간소화 기능 제공

- : Java 기반 설정시 `WebMvcConfigurer.addResourceHandlers(...)` 사용

- : XML 기반 설정시 `mvc:resources` 태그 사용

# 리소스 제공 설정

## Java Config

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("리소스에 접근할 URL 패턴")  
        .addResourceLocations("리소스가 있는 위치");  
}
```

## XML Config

```
<mvc:resources mapping="리소스에 접근할 URL 패턴"  
    location="리소스가 있는 위치"/>
```

# 리소스 제공 설정

## 리소스에 접근할 URL 패턴 정의

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/");  
}
```

<http://spring.io/resources/css/default.css>

<http://spring.io/resources/js/spring-by-pivotal.png>

<http://spring.io/resources/img/spring-by-pivotal.png>

# 리소스 제공 설정

## 리소스가 있는 위치 설정

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/");  
}
```

<http://spring.io/resources/css/default.css>

```
.  
├── WEB-INF  
│   ├── classes  
│   └── lib  
└── resources  
    └── css  
        └── default.css
```

# DEMO

(리소스 제공 설정)



# Resource 인터페이스에 대한 다양한 구현체

## ✓ ServletContextResource

```
registry.addHandler("/resources/**")  
    .addResourceLocations("/resources/");
```

## ✓ ClassPathResource

```
registry.addHandler("/resources/**")  
    .addResourceLocations("classpath:/resources/");
```

## ✓ FileSystemResource

```
registry.addHandler("/resources/**")  
    .addResourceLocations("file:/resources/");
```

## ✓ UrlResource

```
registry.addHandler("/resources/**")  
    .addResourceLocations("http://spring.io");
```

## ✓ etc

# 리소스를 효율적으로 다루는 캐시 설정

## Java Config

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/")  
        .setCachePeriod(31556926);  
}
```

## XML Config

```
<mvc:resources mapping="/resources/**"  
    location="/resources/"  
    cache-period="31556926"/>
```

# 리소스를 효율적으로 다루는 캐시 설정

## Java Config

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/")  
        .setCachePeriod(31556926);  
}
```

Response headers :

**Cache-Control:** "max-age=31556926, must-revalidate"

**Expires:** "Sun, 16 Nov 2014 07:39:20 GMT"

**Last-Modified:** "Thu, 20 Nov 2014 10:49:18 GMT"

# 리소스를 효율적으로 다루는 캐시 설정

## Java Config

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry.addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/")  
        .setCachePeriod(0);  
}
```

Response headers :

**Pragma:** "no-cache"

**Cache-Control:** "no-cache, no-store"

**Expires:** "Thu, 01 Jan 1970 00:00:00 GMT"

# DEMO

(캐시 설정에 따른 동작)

# Multi Module Web Application

(Frontend / Backend)  
(Client-side / Server-side)

# 프로젝트 구조

```
├─ build.gradle
├─ gradle.properties
├─ settings.gradle
├─ backend
│   └─ src
│       ├── main
│       │   ├── java
│       │   └─ resources
│       └─ test
└─ frontend
    ├── package.json
    ├── bower.json
    ├── Gruntfile.js
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
```

- Gradle : 빌드 자동화

- Spring IO Platform 사용해 개발된 Server-side 모듈

- 정적 자원(html, css, javascript)을 제공하는 Client-side 모듈

# backend에 적용된 기술 훑어보기

```
├─ build.gradle
├─ gradle.properties
├─ settings.gradle
├─ backend
│   └─ src
│       ├── main
│       │   ├── java
│       │   └─ resources
│       └─ test
└─ frontend
    ├── package.json
    ├── bower.json
    ├── Gruntfile.js
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
```

## ✓ Spring Boot

- : Spring IO Platform 기반 개발을 빠르고 다양한 방법으로 시작
- : 애플리케이션 기능외 필요한 공통 컴포넌트를 제공

## ✓ Thymeleaf

- : HTML 태그/속성 기반의 템플릿 엔진
- : Spring MVC & Security 와 통합을 위한 라이브러리 제공



# frontend에 적용된 도구 훑어보기

```
├─ build.gradle
├─ gradle.properties
├─ settings.gradle
├─ backend
│   └─ src
│       ├── main
│       │   ├── java
│       │   └─ resources
│       └─ test
└─ frontend
    ├── package.json
    ├── bower.json
    ├── Gruntfile.js
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
```

## ✓ NPM(Node Package Manager)

: 개발환경 및 의존성 관리

## ✓ Bower

: JavaScript Lib 의존성 관리(jquery, bootstrap, etc)


## ✓ Grunt

: Client-side 빌드 자동화

: 다양한 Plugin 지원( jshint, usemin, filerev, assemble, etc )

# frontend에 적용된 기술 훑어보기 (1/3)

```
├─ backend
├─ frontend
│  ├─ package.json
│  ├─ Gruntfile.js
│  ├─ bower.json
│  ├─ src
│  │  ├─ assets
│  │  │  ├─ css
│  │  │  │  ├─ cover.css
│  │  │  │  └─ default.css
│  │  │  └─ js
│  │  │      └─ default.js
│  │  └─ libs
│  │      ├─ bootstrap
│  │      └─ jquery
│  └─ includes
│      ├─ common-css.hbs
│      └─ common-scripts.hbs
├─ layouts
│  └─ default.hbs
├─ pages
│  └─ about.hbs
├─ dist
│  ├─ assets
│  │  ├─ css
│  │  │  └─ style.min.99501602.css
│  │  └─ js
│  │      └─ app.min.264ed108.js
│  └─ pages
│      └─ about.html
```



## ✓ CSS/JS 최적화(병합 및 압축)

- : grunt-usemin
- : grunt-contrib-concat
- : grunt-contrib-cssmin
- : grunt-contrib-uglify

## ✓ Fingerprinting

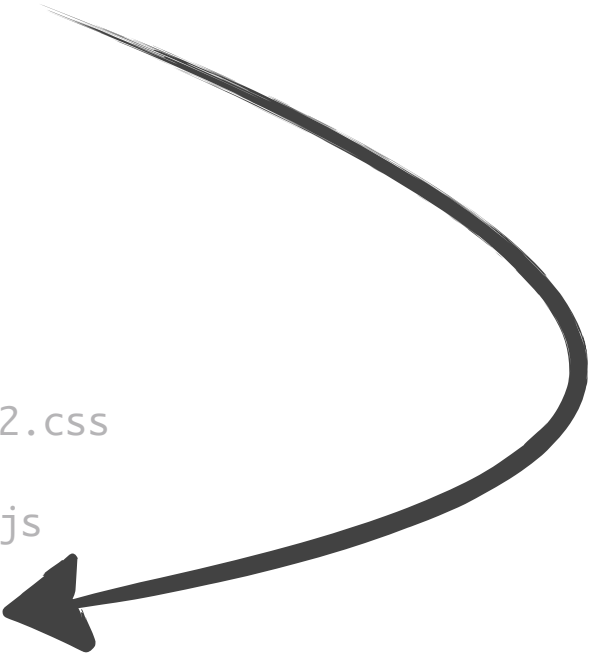
- : grunt-filerev



더 알아보려면 여기로!  
<http://goo.gl/oGVCYT>

# frontend에 적용된 기술 훑어보기 (2/3)

```
├─ backend
├─ frontend
│  ├─ package.json
│  ├─ Gruntfile.js
│  ├─ bower.json
│  ├─ src
│  │  ├─ assets
│  │  │  ├─ css
│  │  │  │  ├─ cover.css
│  │  │  │  └─ default.css
│  │  │  └─ js
│  │  │      └─ default.js
│  │  └─ libs
│  │      ├─ bootstrap
│  │      └─ jquery
│  └─ includes
│      ├─ common-css.hbs
│      └─ common-scripts.hbs
├─ layouts
│   └─ default.hbs
├─ pages
│   └─ about.hbs
├─ dist
│  ├─ assets
│  │  ├─ css
│  │  │  └─ style.min.99501602.css
│  │  └─ js
│  │      └─ app.min.264ed108.js
│  └─ pages
│      └─ about.html
```



✓ 템플릿(HTML) 생성

: assemble

# frontend에 적용된 기술 훑어보기 (3/3)

```
├─ backend
├─ frontend
│   ├── package.json
│   ├── Gruntfile.js
│   ├── bower.json
│   └── src
│       ├── assets
│       │   ├── css
│       │   │   ├── cover.css
│       │   │   └── default.css
│       │   └── js
│       │       └── default.js
│       ├── libs
│       │   ├── bootstrap
│       │   └── jquery
│       ├── includes
│       │   ├── common-css.hbs
│       │   └── common-scripts.hbs
│       ├── layouts
│       │   └── default.hbs
│       └── pages
│           └── about.hbs
└─ dist
    ├── assets
    │   ├── css
    │   │   └── style.min.99501602.css
    │   └── js
    │       └── app.min.264ed108.js
    └── pages
        └── about.html
```



이제  
코드로  
만나보시죠!

# frontend가 가출한 이유

```
├─ backend
├─ frontend
│   ├── package.json
│   ├── Gruntfile.js
│   ├── bower.json
│   └── src
│       ├── assets
│       │   ├── css
│       │   │   ├── cover.css
│       │   │   └── default.css
│       │   └── js
│       │       └── default.js
│       ├── libs
│       │   ├── bootstrap
│       │   └── jquery
│       ├── includes
│       │   ├── common-css.hbs
│       │   └── common-scripts.hbs
│       ├── layouts
│       │   └── default.hbs
│       └── pages
│           └── about.hbs
└─ dist
    ├── assets
    │   ├── css
    │   │   └── style.min.99501602.css
    │   └── js
    │       └── app.min.264ed108.js
    └── pages
        └── about.html
```

✓ dependency management

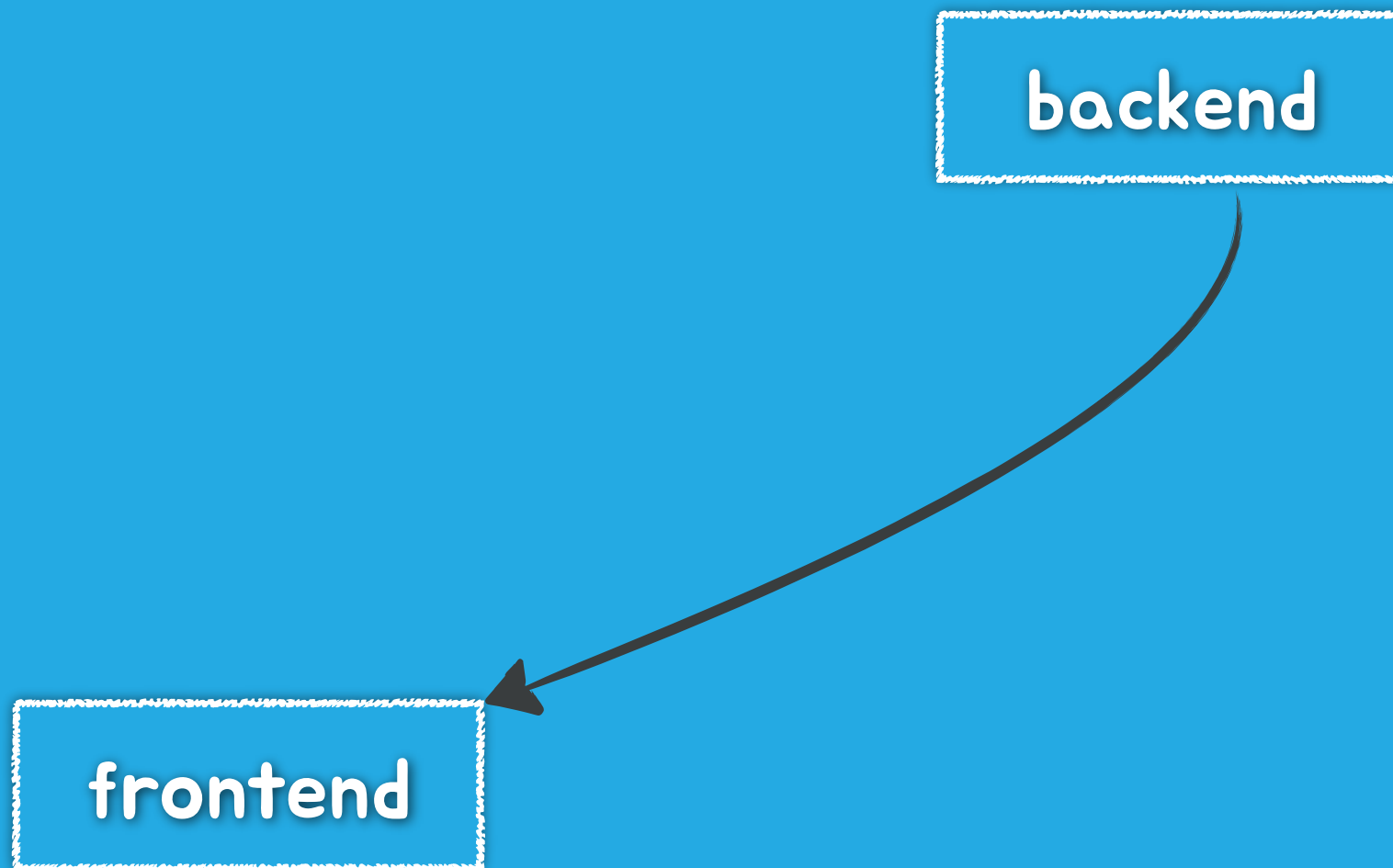
✓ modularity

✓ tests

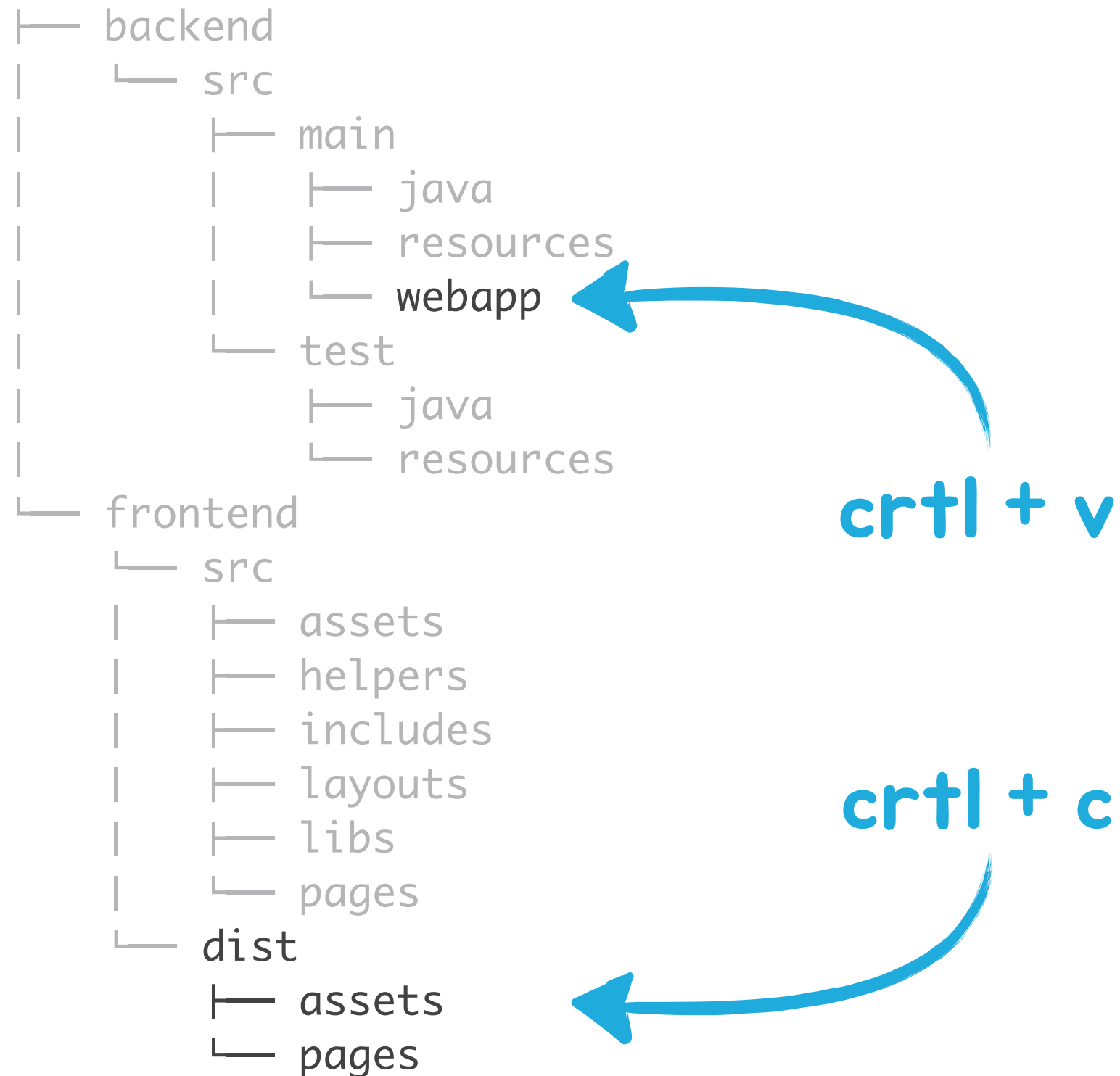
✓ build automation ( vs artifacts)

지금부터 Frontend의  
자원(html, css, javascript, image)을  
사용하는 방법을 살펴봅니다.

# Frontend 의존성 다루기



# 개발자의 친구 복붙-





# Web Libraries in Jars

## WebJars

- ✓ Client-side 웹 라이브러리를 JAR로 묶어서 제공하는 서비스
- ✓ JVM 기반 빌드 도구(gradle, maven, sbt, etc)를 지원 (maven 저장소)

```
<dependencies>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>3.3.1</version>
  </dependency>
</dependencies>
```



```
bootstrap-3.3.1.jar
├── META-INF
│   └── resources
│       └── webjars
│           └── bootstrap
│               └── 3.3.1
│                   ├── css
│                   │   ├── bootstrap.css
│                   │   └── bootstrap.min.css
│                   ├── js
│                   │   ├── bootstrap.js
│                   │   └── bootstrap.min.js
│                   ├── fonts
│                   ├── less
│                   └── webjars-requirejs.js
```

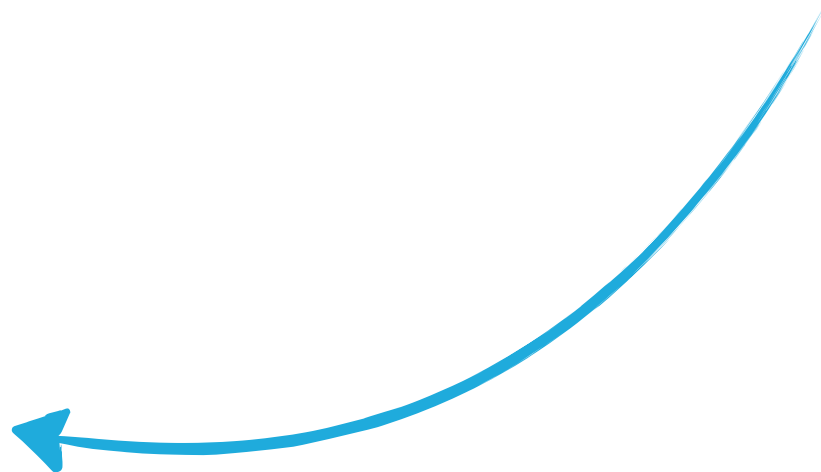
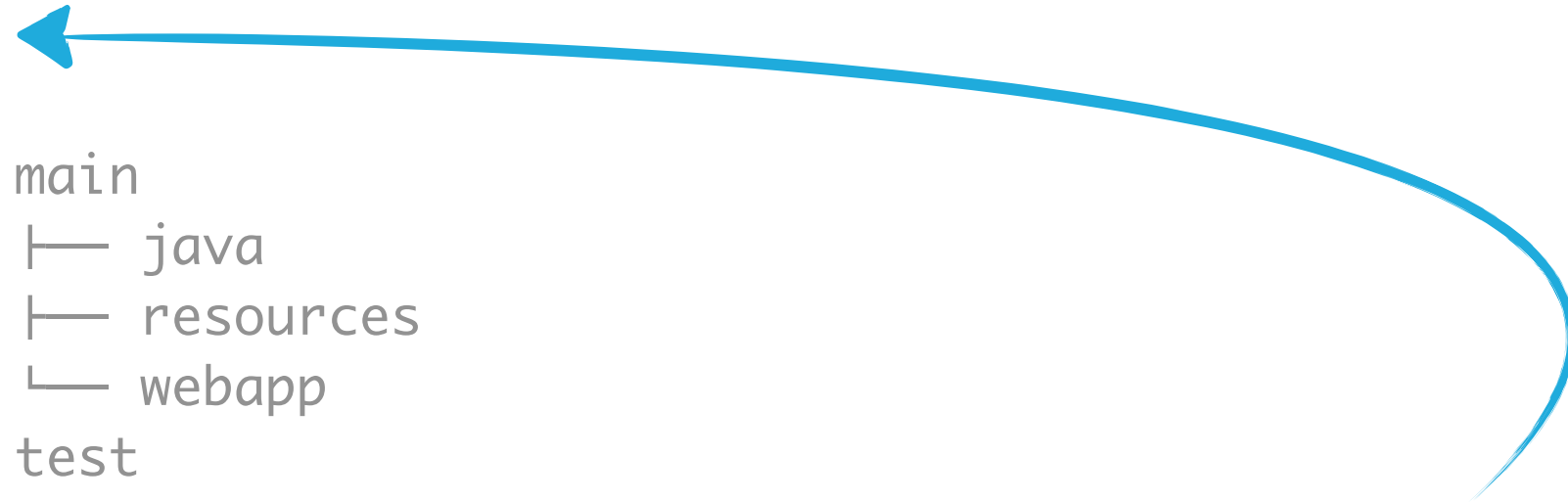
# DEMO

(Servlet 3에서 webjars 사용)

# Gradle로 통합하기

```
├─ build.gradle
├─ backend
│   └─ src
│       ├── main
│       │   ├── java
│       │   ├── resources
│       │   └─ webapp
│       └─ test
└─ frontend
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
    └─ dist
        ├── assets
        └─ pages
```

**frontend.jar** 로 만든 후  
backend 모듈에 의존성을 추가!



# frontend를 빌드 후 **jar**로 만들기

```
└─ build.gradle
└─ backend
    └─ src
        └─ main
            ├── java
            ├── resources
            └─ webapp
        └─ test
└─ frontend
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
    └─ dist
        ├── assets
        └─ pages
```

```
project(':frontend') {
    apply plugin: 'java'

    task npmInstall(type:Exec) {
        // do something
    }

    task gruntBuild(type:Exec, dependsOn: [npmInstall]) {
        // do something
    }

    jar {
        from 'dist'
        includeEmptyDirs = false
    }
    jar.dependsOn gruntBuild
}
```

# backend에 frontend 의존성 추가

```
├─ build.gradle
├─ backend
│   └─ src
│       ├── main
│       │   ├── java
│       │   ├── resources
│       │   └─ webapp
│       └─ test
└─ frontend
    └─ src
        ├── assets
        ├── helpers
        ├── includes
        ├── layouts
        ├── libs
        └─ pages
    └─ dist
        ├── assets
        └─ pages
```

```
project(':frontend') {
    apply plugin: 'java'

    task npmInstall(type: Exec) {
        // do something
    }

    task gruntBuild(type: Exec, dependsOn: [npmInstall]) {
        // do something
    }

    jar {
        from 'dist'
        includeEmptyDirs = false
    }
    jar.dependsOn gruntBuild
}

project(':backend') {
    apply plugin: 'war'

    dependencies {
        runtime project(':frontend')
    }
}
```

# DEMO

(Gradle 통합과 자원 사용)

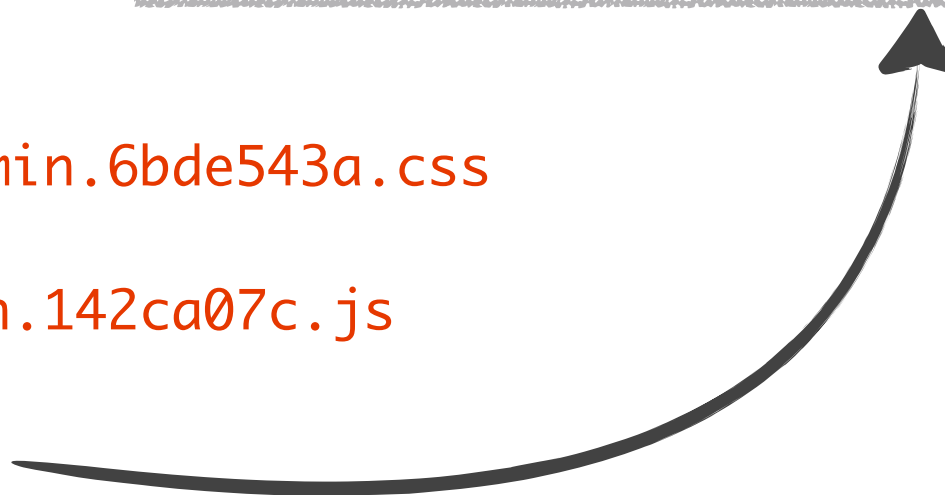
**개발과 배포는 다르다**

# frontend: 배포시에는 최적화된 자원을 사용

> grunt build:release

```
├─ backend
├─ frontend
│   └─ src
│       ├── assets
│       │   ├── css
│       │   ├── img
│       │   └─ js
│       ├── libs
│       │   ├── jquery
│       │   └─ bootstrap
│       └─ pages
└─ dist
    ├── assets
    │   ├── css
    │   │   └─ style.min.6bde543a.css
    │   └─ js
    │       └─ app.min.142ca07c.js
    └─ pages
        └─ about.html
```

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <link href='/assets/css/style.min.6bde543a.css'
        rel='stylesheet' type='text/css'/>
</head>
<body>
  <div class="site-wrapper">
    // 생략
  </div>
  <script src='/assets/js/app.min.264ed108.js'></script>
</body>
</html>
```





# frontend: 배포시에는 최적화된 자원을 사용

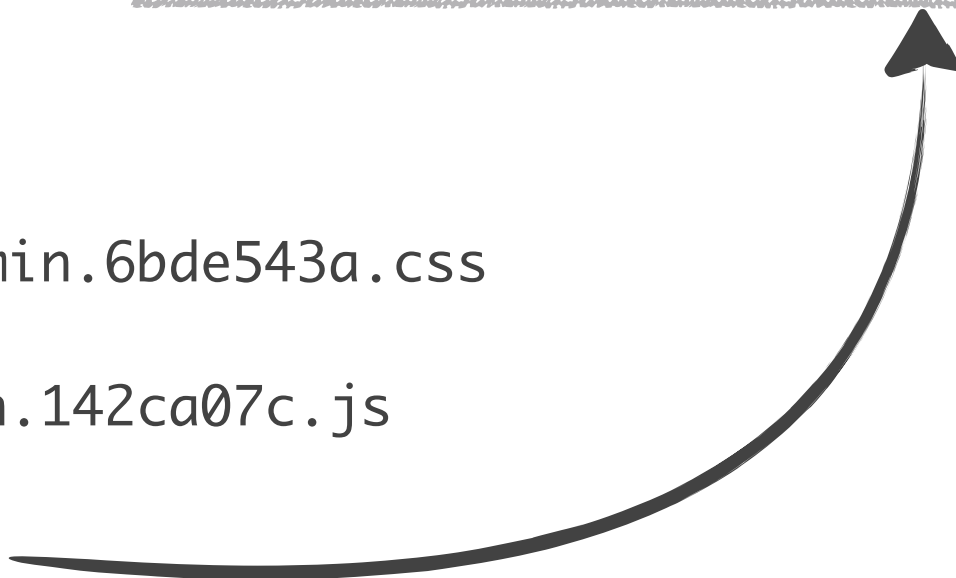
> grunt build:release

```
├─ backend
├─ frontend
│   ├─ src
│   │   ├─ assets
│   │   │   ├─ css
│   │   │   └─ img
│   │   └─ js
│   │   └─ libs
│   │       ├─ jquery
│   │       └─ bootstrap
│   └─ pages
├─ dist
│   ├─ assets
│   │   ├─ css
│   │   │   └─ style.min.6bde543a.css
│   │   └─ js
│   │       └─ app.min.142ca07c.js
│   └─ pages
│       └─ about.html
```

## Classpath 위치한 자원을 사용

```
// build.gradle
project(':frontend') {
    sourceSets.main.resources { srcDir 'dist' }
}

// Java Config
registry.addHandler("/assets/**")
    .addResourceLocations("classpath:assets/");
```



# frontend: 개발시에는 **작성중인** css, js 사용

> grunt build:develop

```
├─ backend
├─ frontend
│   └─ src
│       ├── assets
│       │   ├── css
│       │   │   └─ default.css
│       │   ├── img
│       │   └─ js
│       │       └─ default.js
│       ├── libs
│       │   ├── jquery
│       │   └─ bootstrap
│       └─ pages
└─ dist
    └─ pages
        └─ about.html
```



```
<!DOCTYPE html>
<html lang="ko">
<head>
  <link href="/libs/bootstrap/dist/css/bootstrap.css"/>
  <link href="/assets/css/default.css"/>
</head>
<body>
  <div class="site-wrapper">
    // 생략
  </div>
  <script src="/libs/jquery/dist/jquery.js"/>
  <script src="/libs/bootstrap/dist/js/bootstrap.js"/>
  <script src="/assets/js/default.js"/>
</body>
</html>
```

# backend: 개발시에는 **어떻게** 자원에 접근하지?

> grunt build:develop

```
├─ backend
├─ frontend
│  ├─ src
│  │  ├─ assets
│  │  │  ├─ css
│  │  │  │  └─ default.css
│  │  │  ├─ img
│  │  │  └─ js
│  │  │      └─ default.js
│  │  └─ libs
│  │      ├─ jquery
│  │      └─ bootstrap
│  └─ pages
└─ dist
    └─ pages
        └─ about.html
```


'어라!? **dist/assets**이 없네!'



# backend: 개발시에는 **어떻게** 자원에 접근하지?

> grunt build:develop

```
├─ backend
├─ frontend
│  ├─ src
│  │  ├─ assets
│  │  │  ├─ css
│  │  │  │  └─ default.css
│  │  │  └─ img
│  │  └─ js
│  │      └─ default.js
│  └─ libs
│      ├─ jquery
│      └─ bootstrap
├─ pages
└─ dist
    └─ pages
        └─ about.html
```



**src**를 Classpath에 추가해볼까?

```
// build.gradle
project(':frontend') {
    sourceSets.main.resources { srcDirs 'dist', 'src' }
}

// console
> gradle build
```

# backend: 개발시에는 **어떻게** 자원에 접근하지?

> grunt build:develop

```
├─ backend
├─ frontend
│  ├─ src
│  │  ├─ assets
│  │  │  ├─ css
│  │  │  │  └─ default.css
│  │  │  ├─ img
│  │  │  └─ js
│  │  │      └─ default.js
│  │  ├─ libs
│  │  │  ├─ jquery
│  │  │  └─ bootstrap
│  │  └─ pages
│  └─ dist
│      └─ pages
│          └─ about.html
```

없는데 없는 frontend.jar

```
frontend.jar
├─ assets
│  ├─ css
│  │  └─ default.css
│  ├─ img
│  └─ js
│      └─ default.js
├─ libs
│  ├─ jquery
│  └─ bootstrap
└─ pages
```

이건 아니야... :::

# backend: 환경에 따른 자원 접근 전략 변경

- └─ backend
- └─ frontend
  - └─ src
    - └─ assets
      - └─ css
      - └─ img
      - └─ js
    - └─ libs
      - └─ jquery
      - └─ bootstrap
    - └─ pages
  - └─ dist
    - └─ assets
      - └─ css
      - └─ img
      - └─ js
    - └─ templates
      - └─ about.html

String locations;

```
if(개발) {  
    locations = src/assets 사용  
} else {  
    locations = dist/assets 사용  
}
```

```
registry.addHandler("/assets/**")  
    .addResourceLocations(locations);
```

# DEMO

(환경에 따른 자원 접근 전략 변경)

회고



더 하고 싶었던

이야기가 있었지만...

# 묻고 답하는 시간

<https://github.com/arawn/resource-handling-in-springmvc>

