Spring Boot Project Structure

Spring Initializr

Spring Initializr

- Spring Initializr created a Maven project for us

http://start.spring.io

Spring Initializr

- Spring Initializr created a Maven project for us

- Let's explore the project structure

http://start.spring.io

Maven Standard Directory Structure

Maven Standard Directory Structure

Directory	Description
src/main/java	Your Java source code

Maven Standard Directory Structure

Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app

Maven Standard Directory Structure

Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/test/java	Unit testing source code

Maven Wrapper files

Maven Wrapper files

Maven Wrapper files

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct version and runs Maven

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct version and runs Maven
- Two files are provided

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct ve
      and runs Maven
- Two files are provided
  - **`mvnw.cmd`** for MS Windows

```
> mvnw clean compile test
```

Maven Wrapper files

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct ve
      and runs Maven
- Two files are provided
  - **`mvnw.cmd`** for MS Windows
  - **`mvnw.sh`** for Linux/Mac

```
> mvnw clean compile test
```

```
$ ./mvnw clean compile test
```

Maven Wrapper files

Maven Wrapper files

- If you already have Maven installed previously

Maven Wrapper files

- If you already have Maven installed previously
  - Then you can ignore / delete the **`mvnw`** files

Maven Wrapper files

- If you already have Maven installed previously
  - Then you can ignore/delete the **`mvnw`** files

- Just use Maven as you normally would

```
$ mvn clean compile test
```

Maven POM file

Maven POM file

- **`pom.xml`** includes info that you entered at Spring Initializr website

Maven POM file

- **`pom.xml`** includes info that you entered at Spring Initializr website

```
<groupId>com.luv2code.springboot.demo</groupId>
<artifactId>mycoolapp</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
```

Maven POM file

- **`pom.xml`** includes info that you entered at Spring Initializr website

```
<groupId>com.luv2code.springboot.demo</groupId>
<a
<v
<p
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  …
</dependencies>
```

Maven POM file

- **`pom.xml`** includes info that you entered

Spring Boot Starters

A collection of Maven dependencies (Compatible versions)

```
<groupId>com.luv2code.springboot.demo</groupId>
<a
<v
<p
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  …
</dependencies>
```

Maven POM file

- **`pom.xml`** includes info that you entered

```
<groupId>com.luv2code.springboot.demo</groupId>
<a
<v
<p
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.b
    <artifactId>spring-boot-starte
    <scope>test</scope>
  </dependency>
  …
</dependencies>
```

Spring Boot Starters

A collection of Maven dependencies (Compatible versions)

spring-web
spring-webmvc
hibernate-validator
tomcat
json
…

Maven POM file

- **`pom.xml`** includes info that you entered

```
<groupId>com.luv2code.springboot.demo</groupId>
<a
<v
<p
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  framework.b
  oot-starte

</dependencies>
```

Spring Boot Starters

A collection of Maven dependencies (Compatible versions)

Save's the developer from having to list all of the individual dependencies

Also, makes sure you have compatible versions

spring-web
spring-webmvc
hibernate-validator
tomcat
json
...

Maven POM file

Maven POM file

- Spring Boot Maven plugin

Maven POM file

- Spring Boot Maven plugin

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

Maven POM file

- Spring Boot Maven plugin

To package executable jar
or war archive

Can also easily run the app

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

Maven POM file

- Spring Boot Maven plugin

To package executable jar
or war archive

Can also easily run the app

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

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Maven POM file

- Spring Boot Maven plugin

To package executable jar
or war archive

Can also easily run the app

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

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Can also just use:

mvn package
mvn spring-boot:run

Java Source Code

Java Source Code

Main Spring Boot application class

Created by Spring Initializr

Java Source Code

Main Spring Boot application class

Created by Spring Initializr

RestController that we created in an earlier video

Java Source Code

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplicat

  public static void main(Stri
    SpringApplication.run(Mycoo
  }
}
```

Enables

Auto configuration
Component scanning
Additional configuration

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplicat

  public static void main(Stri
    SpringApplication.run(Mycoo
  }
}
```

Composed of following annotations

@EnableAutoConfiguration
@ComponentScan
@Configuration

Annotations

Annotations

- `@SpringBootApplication` is composed of the following annotations:

Annotation	Description

Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	**Enables Spring Boot's auto-configuration support**

Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	**Enables Spring Boot's auto-configuration support**
`@ComponentScan`	**Enables component scanning of current package Also recursively scans sub-packages**

Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	**Enables Spring Boot's auto-configuration support**
`@ComponentScan`	**Enables component scanning of current package** **Also recursively scans sub-packages**
`@Configuration`	**Able to register extra beans with `@Bean`** **or import other configuration classes**

Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	**Enables Spring Boot's auto-configuration support**
`@ComponentScan`	**En… kage Al…**
`@Configuration`	**Al… or import other configuration classes**

Same annotations that we've used before with traditional Spring apps

Annotations

- **`@SpringBootApplication`** is composed of the following annotations:

Annotation	Description
`@EnableAutoConfiguration`	**Enables Spring Boot's auto-configuration support**
`@ComponentScan`	**Enables component scanning of current package** **Also recursively scans sub-packages**
`@Configuration`	**Able to register extra beans with `@Bean`** **or import other configuration classes**

Java Source Code

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Bootstrap your Spring Boot application

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Bootstrap your Spring Boot application

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MycoolappApplication {

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

Bootstrap your Spring Boot application

Java Source Code

File: MycoolappApplication.java

```
package com.luv2code.springboot.demo.mycoolapp

import org.springframework.boot.SpringApplicat
import org.springframework.boot.autoconfigure.

@SpringBootApplication
public class MycoolappApplication {

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

Behind the scenes …

Creates application context and registers all beans

Starts the embedded server Tomcat etc…

Bootstrap your Spring Boot application

More on Component Scanning

Best Practice

More on Component Scanning

Best Practice

Place your main application class in
the root package above your other packages

More on Component Scanning

Best Practice

Place your main application class in the root package above your other packages

- This implicitly defines a base search package
  - Allows you to leverage default component scanning
  - No need to explicitly reference the base package name

More on Component Scanning

Best Practice

Place your main application class in the root package above your other packages

- This implicitly defines a base search package
  - Allows you to leverage default component scanning
  - No need to explicitly reference the base package name

Common pitfall in traditional Spring apps

Anyone use the wrong package name before in traditional Spring apps???

More on Component Scanning

More on Component Scanning

Main Spring Boot application class

Automatically component scans sub-packages

More on Component Scanning

Main Spring Boot application class

Automatically component scans sub-packages

Includes "rest" sub-package

More on Component Scanning

Main Spring Boot application class

Automatically component scans sub-packages

Includes "rest" sub-package

Any other sub-packages you create
Can give them any name

More on Component Scanning

More on Component Scanning

- Default scanning is fine if everything is under
  - `com.luv2code.springboot.demo.mycoolapp`

More on Component Scanning

- Default scanning is fine if everything is under
  - `com.luv2code.springboot.demo.mycoolapp`

- But what about my other packages?

More on Component Scanning

- Default scanning is fine if everything is under
  - `com.luv2code.springboot.demo.mycoolapp`

- But what about my other packages?
  - `org.acme.iot.utils`

More on Component Scanning

- Default scanning is fine if everything is under
  - `com.luv2code.springboot.demo.mycoolapp`

- But what about my other packages?
  - `org.acme.iot.utils`
  - `edu.cmu.wean`

More on Component Scanning

- Default scanning is fine if everything is under
  - **com.luv2code.springboot.demo.mycoolapp**

- But what about my other packages?
  - **org.acme.iot.utils**
  - **edu.cmu.wean**

```
package com.luv2code.springboot.demo.mycoolapp;
…
@SpringBootApplication(
    scanBasePackages={"com.luv2code.springboot.demo.mycoolapp",
                      "org.acme.iot.utils",
                      "edu.cmu.wean"})
public class MycoolappApplication {
  …
}
```

More on Component Scanning

- Default scanning is fine if everything is under
  - **`com.luv2code.springboot.demo.mycoolapp`**

- But what about my other packages?
  - **`org.acme.iot.utils`**
  - **`edu.cmu.wean`**

Explicitly list base packages to scan

```
package com.luv2code.springboot.demo.mycoolapp;
…
@SpringBootApplication(
     scanBasePackages={"com.luv2code.springboot.demo.mycoolapp",
                       "org.acme.iot.utils",
                       "edu.cmu.wean"})
public class MycoolappApplication {
   …
}
```

Application Properties

- By default, Spring Boot will load properties from: **`application.properties`**

Application Properties

- By default, Spring Boot will load properties from: **`application.properties`**

Created by Spring Initializr

Empty at the beginning

Application Properties

- By default, Spring Boot will load properties from: **`application.properties`**

Created by Spring Initializr

Empty at the beginning

Can add Spring Boot properties
`server.port=8585`

Application Properties

- By default, Spring Boot will load properties from: **`application.properties`**

Created by Spring Initializr

Empty at the beginning

Can add Spring Boot properties
`server.port=8585`

Also add your own custom properties
`coach.name=Mickey Mouse`

Application Properties

- Read data from: **`application.properties`**

Application Properties

- Read data from: **`application.properties`**

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {
```

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;
```

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;
```

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;
```

Application Properties

- Read data from: **`application.properties`**

```
# configure server port
server.port=8484

# configure my props
coach.name=Mickey Mouse
team.name=The Mouse Crew
```

```
@RestController
public class FunRestController {

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;

  ...
}
```

Static Content

Static Content

By default, Spring Boot will load static resources from "/static" directory

Static Content

By default, Spring Boot will load static resources from "/static" directory

Examples of static resources
HTML files, CSS, JavaScript, images, etc …

Static Content

Static Content

WARNING:

Do not use the `src/main/webapp` directory if your application is packaged as a JAR.

Although this is a standard Maven directory, it works only with WAR packaging.

It is silently ignored by most build tools if you generate a JAR.

Templates

Templates

- Spring Boot includes auto-configuration for following template engines

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker
  - Thymeleaf

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker
  - Thymeleaf
  - Mustache

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker
  - Thymeleaf
  - Mustache

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker
  - Thymeleaf
  - Mustache

Templates

- Spring Boot includes auto-configuration for following template engines
  - FreeMarker
  - Thymeleaf
  - Mustache

By default, Spring Boot will load templates from "/templates" directory

Thymeleaf is a popular template engine
We will use it later in the course

Unit Tests

Unit Tests

src
main
java
resources
test
java
com
luv2code
springboot
demo
mycoolapp
MycoolappApplicationTests.java

Unit Tests

Spring Boot unit test class

Created by Spring Initializr

Unit Tests

Spring Boot unit test class

Created by Spring Initializr

You can add unit tests to the file