

Spring Boot – Going Further

Topics

- **Working with Properties**
- Overriding Properties
- Using YAML for configuration

Inject Configuration with @Value

application.properties

```
quoteService.uri=http://localhost:8080
```

```
@Configuration
public class QuoteServiceConfiguration {

    @Value("${quoteService.uri}")
    String uri;

    (...)

}
```

The Problem with @Value

- Using property placeholders is sometimes cumbersome
 - Many properties, prefix has to be repeated

```
@Configuration
public class RewardsClientConfiguration {

    @Value("${rewards.client.host}") String host;
    @Value("${rewards.client.port}") int port;

    (...)

}
```

Use @ConfigurationProperties

- Use @ConfigurationProperties on a dedicated class
 - It contains the externalized properties
 - It avoids repeating the prefix
 - Java properties are mapped with properties names

```
@ConfigurationProperties(prefix="rewards.client")
public class ConnectionSettings {

    private String host;
    private int port;
    // getters / setters
}
```

```
rewards.client.host=192.168.1.23
rewards.client.port=8080
```

application.properties

Use @EnableConfigurationProperties

- @EnableConfigurationProperties on configuration class
 - Specify and inject a properties (settings) bean
 - Use it to configure and create the beans

```
@Configuration
@EnableConfigurationProperties (ConnectionSettings.class)
public class RewardsClientConfiguration {

    @Autowired ConnectionSettings connectionSettings;

    @Bean public RewardClient rewardClient() {
        return new RewardClient(
            connectionSettings.getHost(),
            connectionSettings.getPort()
        );
    }
}
```

Spring Profiles

- Segregate configuration based on environment
- application-{profile}.properties

Set Active Profiles:

```
SPRING_PROFILES_ACTIVE=qa
```

Topics

- Working with Properties
- **Overriding Properties**
- Using YAML for configuration

Overriding Properties

- Order of evaluation of the properties (non-exhaustive)
 - 1) Command line arguments
 - 2) Java system properties
 - 3) OS environment variables
 - 4) Property file(s)
- Property files provide defaults, override by external means
 - Access properties via usual syntax in the configuration

```
@Configuration
class AppConfig {

    @Value("${test.property}")
    String testProperty;

    ...
}
```

Relaxed Property Binding

- No need for an exact match between desired properties and names
- Intuitive mapping between system properties and environment variables
 - `test.property` **equivalent** to `TEST_PROPERTY`
 - `test.property` **isn't** a valid env variable name
 - **Ease** overriding property without changing the name!

Topics

- Working with Properties
- Overriding Properties
- **Using YAML for configuration**

YAML for Properties

- Spring Boot support YAML for Properties
 - An alternative to properties files

```
database.host=localhost  
database.user=admin
```

application.properties

```
database  
  host: localhost  
  user: admin
```

application.yml

- YAML is convenient for hierarchical configuration data
 - Spring Boot properties are organized in groups
 - e.g. server, database, etc
- Spring Boot automatically picks the YAML file
 - SnakeYAML must be on the classpath, provided by spring-boot-starter

Multi-profile File with YAML

- YAML file can contain for several documents
- Convenient to specify alternate configurations in the same file

