SpEL

You can use Spring Expression Language (SpEL) within the `@Value` annotation to reference properties of a bean. SpEL allows you to access bean properties, method invocations, arithmetic operations, and more.

Here's an example of how to use SpEL within the `@Value` annotation to reference a property of another bean:

```java
@Component
public class MyComponent {

    private String myProperty;

    // Constructor, getters, and setters

}

@Component
public class AnotherComponent {

    @Value("#{myComponent.myProperty}")
    private String anotherProperty;

    // Constructor, getters, and setters

}
```

In this example, the `@Value("#{myComponent.myProperty}")` annotation is using SpEL to reference the `myProperty` property of the `MyComponent` bean.

Make sure that the `MyComponent` bean is defined in your Spring configuration so that it can be referenced by other components.

https://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html

To use the `string.array.value` property from your application properties file with `@Value` annotation, you can do the following:

1. Define the property in your `application.properties` file:
   ```properties
   string.array.value=value1,value2,value3
   ```

2. Inject the property into your Spring bean using the `@Value` annotation:
   ```java
   import org.springframework.beans.factory.annotation.Value;
   import org.springframework.stereotype.Component;

   @Component
   public class MyComponent {

       @Value("#{'${string.array.value}'.split(',')}")
       private String[] arrayValue;

       public void printArrayValue() {
           for (String value : arrayValue) {
               System.out.println(value);
           }
       }
   }
   ```

In this example, `${string.array.value}` is replaced with the property value from `application.properties`. The `split(',')` function splits the property value into an array using comma as the delimiter. This array is then injected into the `arrayValue` field of `MyComponent`.

In Spring's SpEL (Spring Expression Language), you use `#` to indicate that you're referencing a Spring Bean or evaluating a Spring-specific expression, and you use `$` to reference placeholders, environment variables, or system properties.

Here's a breakdown:

1. **#**:
   - Used for evaluating Spring-specific expressions.
   - Examples:
     - `@Value("#{'${my.property}')}"`: Evaluates a Spring Bean property named `my.property`.
     - `@Autowired`: Used to inject Spring Beans.

2. **$**:
   - Used for referencing placeholders, environment variables, or system properties.
   - Examples:
     - `@Value("${my.property}")`: Resolves the value of the property `my.property` from the property sources.
     - `${JAVA_HOME}`: References the `JAVA_HOME` environment variable.

- `${user.home}`: References the `user.home` system property.

In summary, use `#` for Spring Bean-related expressions and `@` for referencing placeholders, environment variables, or system properties.

The reason why `@Value("#{'${string.array.value}'.split("-")}")` does not work as expected is that the `@Value` annotation resolves the placeholder `${string.array.value}` before applying any SpEL expressions. Therefore, the `split("-")` method is applied to the resolved string, not to the placeholder.

To achieve splitting the string retrieved from the property placeholder, you can use SpEL directly within the `split()` method like this:

```java
@Value("#{'${string.array.value}'.split('-')}")
```

This expression retrieves the value of the `string.array.value` property placeholder and then applies the `split('-')` method to it. Make sure that `${string.array.value}` resolves to a string that can be split using the `-` delimiter.

@Value("#{${string.array.value}.split('-')}")

This will not work as by default spring understands it as integer.
So for string we have to wrap it with single quotes and then apply operation.

https://www.baeldung.com/spring-expression-language

To call the method:

@Value("#{<bean name>.<method name>}")

To call any state of bean:

@Value("#{<bean name>.<any property>}")

To do dependency inject (a bean)

@Value("#{<bean name>}")

For relation operators we can use (lt, gt, eq, neq, lte, gte) in combination with ==, !=, >, <, <= , >=

We can use ternary operator:

@Value("#{<bean name>.<state> == 'something' ? 'Take this' : "or take this' "})

If the type of variable is boolean

@Value("#{<bean name>.<state> == 'something' ? true : false "})

For logical operators we can use or, and, !=

Mathematical operators can also be applied directly.

To access the list under a bean:

@Value("#{<bean name>.<list-name>[<index value>]})
Similarly for map but it can be string, boolean, etc. as key

For checking string with regular expression:
@Value("#{address.state matches employee.stateRegex}"})
Here employee and address both are beans and we are using this inside Employee class

To call static method/variables of any bean or built in classes classes

#{T(<qualified name of class>).<any method>}