Properties

To access properties defined in the `application.properties` file directly in your Spring Boot application, you can use the `@Value` annotation. Here's how you can do it:

1. Define your properties in the `application.properties` file:

```properties
# application.properties

app.name=MyApp
app.version=1.0
```

2. Inject the properties into your Spring beans using the `@Value` annotation:

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class MyComponent {

    @Value("${app.name}")
    private String appName;

    @Value("${app.version}")
    private String appVersion;

    public void displayProperties() {
        System.out.println("App Name: " + appName);
        System.out.println("App Version: " + appVersion);
    }
}
```

3. Now you can use the `appName` and `appVersion` variables in your component as needed. When the application starts, Spring Boot will automatically inject the values from the `application.properties` file into these variables.

This approach allows you to externalize configuration properties and easily inject them into your Spring beans. You can use the `@Value` annotation to inject properties into fields, methods, or constructor parameters of your beans.

Without using `@Value`, you can still access properties from the `application.properties` file by injecting the `Environment` object into your Spring bean and using its `getProperty` method. Here's how you can do it:

1. Define your properties in the `application.properties` file:

```properties
# application.properties

app.name=MyApp
app.version=1.0
```

2. Inject the `Environment` object into your Spring bean:

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;

@Configuration
public class MyConfiguration {

    @Autowired
    private Environment environment;

    public void displayProperties() {
        String appName = environment.getProperty("app.name");
        String appVersion = environment.getProperty("app.version");

        System.out.println("App Name: " + appName);
        System.out.println("App Version: " + appVersion);
    }
}
```

3. Now you can use the `getProperty` method of the `Environment` object to access the properties defined in the `application.properties` file. This approach provides a more flexible way to access properties and is useful when you need to access properties programmatically or when the properties are dynamic.