InjectionPoint

`InjectionPoint` is an interface in Spring Framework that provides metadata about the injection point at which a bean is being injected. It allows access to information such as the method or field into which the bean is being injected, the target class that declares the injection point, and any annotations present on the injection point.

Here's a brief overview of the methods provided by the `InjectionPoint` interface:

- `Member getMember()`: Returns the member (field, method, or constructor) into which the bean is being injected.
- `Class<?> getDeclaringType()`: Returns the class that declares the injection point.
- `Type getType()`: Returns the type of the injection point.
- `AnnotatedElement getAnnotatedElement()`: Returns the annotated element at the injection point, if available.

`InjectionPoint` is often used in conjunction with `@Autowired` or `@Inject` annotations to gain insight into the context of dependency injection and to perform conditional injection based on the metadata obtained from the injection point.

https://moelholm.com/blog/2016/10/09/spring-43-introducing-the-injectionpoint-class
https://stackoverflow.com/questions/70008693/accessing-injectionpoint-inside-spring-functional-bean-definition-dsl

Sure, here's an example demonstrating the usage of `InjectionPoint` with Spring's `@Autowired` annotation:

```java
import org.springframework.beans.factory.InjectionPoint;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;

@Configuration
public class FactoryMethodComponent {

    @Bean
    @Scope("prototype")
    public TestBean prototypeInstance(InjectionPoint injectionPoint) {
        // Accessing metadata from InjectionPoint
        String memberName = injectionPoint.getMember().getName();
        Class<?> declaringType = injectionPoint.getMember().getDeclaringClass();

        // Creating a prototype instance of TestBean
        return new TestBean("Prototype instance for " +
declaringType.getSimpleName() + "." + memberName);
    }
}
```

In this example:

- We have a `FactoryMethodComponent` class annotated with `@Configuration`, indicating that it contains bean definitions.
- The `prototypeInstance` method is annotated with `@Bean` to indicate that it produces a bean managed by the Spring container. It is also annotated with `@Scope("prototype")`, specifying that it creates prototype-scoped instances.
- The method takes an `InjectionPoint` parameter, which provides metadata about the injection point.
- Inside the method, we access metadata from the `InjectionPoint` instance, such as the name of the member (field, method, or constructor) into which the bean will be injected, and the declaring type (class) of that member.
- We use this metadata to create a custom message for the `TestBean` instance, indicating the declaring class and member name where the bean is being injected.

This way, we can dynamically customize the creation of beans based on the injection point metadata.

If we were to inject the `TestBean` prototype instance into multiple places within our application, each injection point would receive a different instance of `TestBean` with a customized message based on the injection point metadata.

Here's an example to illustrate the output:

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

@Component
public class BeanUser {

    private final TestBean testBean1;
    private final TestBean testBean2;

    @Autowired
    public BeanUser(TestBean testBean1, TestBean testBean2) {
        this.testBean1 = testBean1;
        this.testBean2 = testBean2;
    }

    public void useTestBeans() {
        System.out.println("TestBean 1: " + testBean1.getMessage());
        System.out.println("TestBean 2: " + testBean2.getMessage());
    }
}
```

When `useTestBeans` method of `BeanUser` is invoked, it will print the messages associated with the injected `TestBean` instances:

```java
TestBean 1: Prototype instance for BeanUser.testBean1
TestBean 2: Prototype instance for BeanUser.testBean2
```

Each `TestBean` instance will have a different message based on the member name and declaring type where it is injected. This demonstrates how we can customize the behavior of prototype-scoped beans using injection point metadata.

No, the `InjectionPoint` interface is primarily used in conjunction with `@Bean` methods within `@Configuration` classes to provide contextual information about the point of injection. It allows you to access metadata about the injection point, such as the field or method where the injection occurs, the containing class, and annotations present at the injection point.

When a bean is created using a `@Bean` method within a `@Configuration` class, Spring injects the `InjectionPoint` instance into the method's parameter list if it's declared as a parameter. This allows the method to access information about the point of injection, which can be useful for conditional bean creation or customizing bean instantiation based on the injection context.

Outside of `@Configuration` classes, the use of `InjectionPoint` is limited, as it's primarily intended for use within the context of `@Bean` method definitions. In other parts of the application, such as regular Spring components (`@Component`, `@Service`, etc.), dependency injection is typically performed using `@Autowired` annotations on fields, constructor parameters, or setter methods. These injection points do not expose the same level of contextual information as `InjectionPoint`.