<http://java-interview-questions.com/cgi-java-interview-questions-cgi-interview-questions-for-java/?_lbGate=468087>

Spring :

1. What are different spring bean scopes?

<http://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html>

Ans: The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext).

|  |
| --- |
| singleton |
|  |
| Scopes a single bean definition to a single object instance per Spring IoC container. |
| he singleton scope is the default scope in Spring. |
| Prototype |
|  |
| Scopes a single bean definition to any number of object instances. |
|  |
| Request |
|  |
| Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.  Per each http:// |
|  |
| session |
|  |
| Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring **ApplicationContext**. |
| Per entire http session until discarded |
| **global session** |
|  |
| Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically, only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext. |

The global session scope is similar to the standard HTTP Session scope ([described immediately above](http://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-session)), and really only makes sense in the context of portlet-based web applications

1. Explain different annotations used in Spring?

Ans:

<http://www.techferry.com/articles/spring-annotations.html>

<http://refcardz.dzone.com/refcardz/spring-annotations>

<http://simplespringtutorial.com/annotations.html>

**Spring Annotations: Contents:**

 To summarize, @Autowired wires by type and @Resource wires by name.

|  |  |
| --- | --- |
| **Annotation** | **Package Detail/Import statement** |
| [@Service](http://www.techferry.com/articles/spring-annotations.html#Service) | import org.springframework.stereotype.Service; |
| [@Repository](http://www.techferry.com/articles/spring-annotations.html#Repository) | import org.springframework.stereotype.Repository; |
| [@Component](http://www.techferry.com/articles/spring-annotations.html#Component) | import org.springframework.stereotype.Component; |
| [@Autowired](http://www.techferry.com/articles/spring-annotations.html#Autowired) | import org.springframework.beans.factory.annotation.Autowired; |
| [@Transactional](http://www.techferry.com/articles/spring-annotations.html#Transactional) | import org.springframework.transaction.annotation.Transactional; |
| [@Scope](http://www.techferry.com/articles/spring-annotations.html#Scope) | import org.springframework.context.annotation.Scope; |
| [Spring MVC Annotations](http://www.techferry.com/articles/spring-annotations.html#MVC) | |
| [@Controller](http://www.techferry.com/articles/spring-annotations.html#Controller) | import org.springframework.stereotype.Controller; |
| [@RequestMapping](http://www.techferry.com/articles/spring-annotations.html#RequestMapping) | import org.springframework.web.bind.annotation.RequestMapping; |
| [@PathVariable](http://www.techferry.com/articles/spring-annotations.html#PathVariable) | import org.springframework.web.bind.annotation.PathVariable; |
| [@RequestParam](http://www.techferry.com/articles/spring-annotations.html#RequestParam) | import org.springframework.web.bind.annotation.RequestParam; |
| [@ModelAttribute](http://www.techferry.com/articles/spring-annotations.html#ModelAttribute) | import org.springframework.web.bind.annotation.ModelAttribute; |
| [@SessionAttributes](http://www.techferry.com/articles/spring-annotations.html#SessionAttributes) | import org.springframework.web.bind.annotation.SessionAttributes; |
| [Spring Security Annotations](http://www.techferry.com/articles/spring-annotations.html#SpringSecurity) | |
| [@PreAuthorize](http://www.techferry.com/articles/spring-annotations.html#PreAuthorize) | import org.springframework.security.access.prepost.PreAuthorize; |

For spring to process annotations, add the following lines in your application-context.xml file.

<context:annotation-config />

<context:component-scan base-package="...specify your package name..." />

|  |  |
| --- | --- |
| Spring Annotation Tip | Spring supports both Annotation based and XML based configurations. You can even mix them together. Annotation injection is performed before XML injection, thus the latter configuration will override the former for properties wired through both approaches. |

**@Service**

Annotate all your service classes with @Service. All your business logic should be in Service classes.

|  |  |
| --- | --- |
| 1  2  3  4 | @Service  public class CompanyServiceImpl implements CompanyService {  ...  } |

**@Repository**

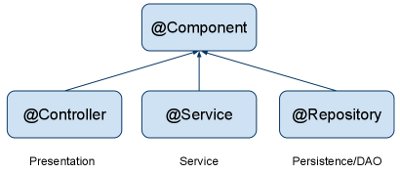
Annotate all your DAO classes with @Repository. All your database access logic should be in DAO classes.

|  |  |
| --- | --- |
| 1  2  3  4 | @Repository  public class CompanyDAOImpl implements CompanyDAO {  ...  } |

**@Component**

Annotate your other components (for example REST resource classes) with @Component.

|  |  |
| --- | --- |
| 1  2  3  4 | @Component  public class ContactResource {  ...  } |

@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.   
  


**@Autowired**

Let Spring auto-wire other beans into your classes using @Autowired annotation. 

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | @Service  public class CompanyServiceImpl implements CompanyService {      @Autowired    private CompanyDAO companyDAO;      ...  } | |
| Spring Annotation Tip | Spring beans can be wired by name or by type.   * @Autowire by default is a type driven injection. @Qualifier spring annotation can be used to further fine-tune autowiring. * @Resource (javax.annotation.Resource) annotation can be used for wiring by name.   Beans that are themselves defined as a collection or map type cannot be injected through @Autowired, because type matching is not properly applicable to them. Use @Resource for such beans, referring to the specific collection or map bean by unique name. |

**@Transactional**

Configure your transactions with @Transactional spring annotation.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @Service  public class CompanyServiceImpl implements CompanyService {      @Autowired    private CompanyDAO companyDAO;      @Transactional    public Company findByName(String name) {        Company company = companyDAO.findByName(name);      return company;    }    ...  } | |
| Spring Annotation Tip | To activate processing of Spring's @Transactional annotation, use the <tx:annotation-driven/> element in your spring's configuration file. |

The default @Transactional settings are as follows: (see transaction management .doc)

* Propagation setting is PROPAGATION\_REQUIRED.
* Isolation level is ISOLATION\_DEFAULT.
* Transaction is read/write.
* Transaction timeout defaults to the default timeout of the underlying transaction system, or to none if timeouts are not supported.
* Any RuntimeException triggers rollback, and any checked Exception does not.

These default settings can be changed using various properties of the @Transactional spring annotation. 

|  |  |
| --- | --- |
| Spring Annotation Tip | Specifying the @Transactional annotation on the bean class means that it applies to all applicable business methods of the class. Specifying the annotation on a method applies it to that method only. If the annotation is applied at both the class and the method level, the method value overrides if the two disagree. |

**@Scope**

As with Spring-managed components in general, the default and most common scope for autodetected components is singleton. To change this default behavior, use @Scope spring annotation.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Component  @Scope("request")  public class ContactResource {  ...  } |

Similarly, you can annotate your component with @Scope("prototype") for beans with prototype scopes.

|  |  |
| --- | --- |
| Spring Annotation Tip | Please note that the dependencies are resolved at instantiation time. For prototype scope, it does NOT create a new instance at runtime more than once. It is only during instantiation that each bean is injected with a separate instance of prototype bean. |

**Spring MVC Annotations**

**@Controller**

Annotate your controller classes with @Controller.

|  |  |
| --- | --- |
| 1  2  3  4 | @Controller  public class CompanyController {  ...  } |

**@RequestMapping**

You use the @RequestMapping spring annotation to map URLs onto an entire class or a particular handler method. Typically the class-level annotation maps a specific request path (or path pattern) onto a form controller, with additional method-level annotations narrowing the primary mapping.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Controller  @RequestMapping("/company")  public class CompanyController {      @Autowired    private CompanyService companyService;  ...  } |

**@PathVariable**

You can use the @PathVariable spring annotation on a method argument to bind it to the value of a URI template variable. In our example below, a request path of /company/techferry will bind companyName variable with 'techferry' value.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @Controller  @RequestMapping("/company")  public class CompanyController {      @Autowired    private CompanyService companyService;      @RequestMapping("{companyName}")    public String getCompany(Map<String, Object> map, @PathVariable String companyName) {      Company company = companyService.findByName(companyName);      map.put("company", company);      return "company";    }  ...  } |

**@RequestParam**

You can bind request parameters to method variables using spring annotation @RequestParam.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @Controller  @RequestMapping("/company")  public class CompanyController {      @Autowired    private CompanyService companyService;      @RequestMapping("/companyList")    public String listCompanies(Map<String, Object> map, @RequestParam int pageNum) {      map.put("pageNum", pageNum);      map.put("companyList", companyService.listCompanies(pageNum));      return "companyList";    }  ...  } |

Similarly, you can use spring annotation @RequestHeader to bind request headers.

**@ModelAttribute**

An @ModelAttribute on a method argument indicates the argument should be retrieved from the model. If not present in the model, the argument should be instantiated first and then added to the model. Once present in the model, the argument's fields should be populated from all request parameters that have matching names. This is known as data binding in Spring MVC, a very useful mechanism that saves you from having to parse each form field individually.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @Controller  @RequestMapping("/company")  public class CompanyController {      @Autowired    private CompanyService companyService;      @RequestMapping("/add")    public String saveNewCompany(@ModelAttribute Company company) {      companyService.add(company);      return "redirect:" + company.getName();    }  ...  } |

**@SessionAttributes**

@SessionAttributes spring annotation declares session attributes. This will typically list the names of model attributes which should be transparently stored in the session, serving as form-backing beans between subsequent requests.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Controller  @RequestMapping("/company")  @SessionAttributes("company")  public class CompanyController {      @Autowired    private CompanyService companyService;  ...  } |

@SessionAttribute works as follows:

* @SessionAttribute is initialized when you put the corresponding attribute into model (either explicitly or using @ModelAttribute-annotated method).
* @SessionAttribute is updated by the data from HTTP parameters when controller method with the corresponding model attribute in its signature is invoked.
* @SessionAttributes are cleared when you call setComplete() on SessionStatus object passed into controller method as an argument.

The following listing illustrate these concepts. It is also an example for pre-populating Model objects.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | @Controller  @RequestMapping("/owners/{ownerId}/pets/{petId}/edit")  @SessionAttributes("pet")  public class EditPetForm {        @ModelAttribute("types")        public Collection<PetType> populatePetTypes() {          return this.clinic.getPetTypes();      }        @RequestMapping(method = RequestMethod.POST)      public String processSubmit(@ModelAttribute("pet") Pet pet, BindingResult result,              SessionStatus status) {          new PetValidator().validate(pet, result);          if (result.hasErrors()) {              return "petForm";          }else {              this.clinic.storePet(pet);              status.setComplete();              return "redirect:owner.do?ownerId=" + pet.getOwner().getId();          }      }  } |

**Spring Security Annotations**

**@PreAuthorize**

Using Spring Security @PreAuthorize annotation, you can authorize or deny a functionality. In our example below, only a user with Admin role has the access to delete a contact.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Transactional  @PreAuthorize("hasRole('ROLE\_ADMIN')")  public void removeContact(Integer id) {    contactDAO.removeContact(id);  } |

<http://stackoverflow.com/questions/4093504/resource-vs-autowired>

[**@Resource vs @Autowired**](http://stackoverflow.com/questions/4093504/resource-vs-autowired)

As for the original question: both, without specifying any attributes of the annotation, perform injection by type. The difference is:

* @Resource allows you to specify a name of the injected bean
* @Autowired allows you to mark it as non-mandatory.

<http://blogs.sourceallies.com/2011/08/spring-injection-with-resource-and-autowired/>

Conclusions

With the exception of test 2 & 7 the configuration and outcomes were identical. When I looked under the hood I determined that the ‘@Autowired’ and ‘@Inject’ annotation behave identically. Both of these annotations use the [‘AutowiredAnnotationBeanPostProcessor’](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/beans/factory/annotation/AutowiredAnnotationBeanPostProcessor.html) to inject dependencies. ‘@Autowired’ and ‘@Inject’ can be used interchangeable to inject Spring beans. However the ‘@Resource’ annotation uses the [‘CommonAnnotationBeanPostProcessor’](http://static.springsource.org/spring/docs/3.1.0.M2/javadoc-api/org/springframework/context/annotation/CommonAnnotationBeanPostProcessor.html) to inject dependencies. Even though they use different post processor classes they all behave nearly identically. Below is a summary of their execution paths.

**@Autowired and @Inject**

1. Matches by Type
2. Restricts by Qualifiers
3. Matches by Name

**@Resource**

1. Matches by Name
2. Matches by Type
3. Restricts by Qualifiers (ignored if match is found by name)

While it could be argued that ‘@Resource’ will perform faster by name than ‘@Autowired’ and ‘@Inject’ it would be negligible. This isn’t a sufficient reason to favor one syntax over the others. I do however favor the ‘@Resource’ annotation for it’s concise notation style.

|  |
| --- |
| @Resource(name="person") |
| @Autowired  @Qualifier("person") |

|  |
| --- |
| @Inject  @Qualifier("person") |

You may argue that they can be equal concise if you use the field name to identify the bean name.

|  |
| --- |
| @Resource  **private** Party person; |
| @Autowired  **private** Party person; |

|  |
| --- |
| @Inject  **private** Party person; |

True enough, but what happens if you want to [refactor](http://en.wikipedia.org/wiki/Code_refactoring) your code? By simply renaming the field name you’re no longer referring to the same bean. I recommend the following practices when wiring beans with annotations.

**Spring Annotation Style Best Practices**

1. Explicitly name your component [@Component("beanName")]
2. Use ‘@Resource’ with the ‘name’ attribute [@Resource(name="beanName")]
3. Avoid ‘@Qualifier’ annotations unless you want to create a list of similar beans. For example you may want to mark a set of rules with a specific ‘@Qualifier’ annotation. This approach makes it simple to inject a group of rule classes into a list that can be used for processing data.
4. Scan specific packages for components [context:component-scan base-package="com.sourceallies.person"]. While this will result in more component-scan configurations it reduces the chance that you’ll add unnecessary components to your Spring context.

Following these guidelines will increase the readability and stability of your Spring annotation configurations.