Qi Wan.

#1. @ Get. → get ~~a request~~
@ Put. → update
@ Delete. → delete
@ Post. → create

#2. @RestController = @ Controller + @ ResponseBody.

#3. produces =({"application/xml", "application/json"})

#4. cookie is ~~for~~ front-end storage. size is small and it can be used with every http request.

session ~~has a tab~~ and is for back-end storage. bigger size and it has a expiration time. It will expire if time is out without user's interaction or the tab is closed.

#5. CORS is cross origin resoure sharing. overide dispatcher servlet.
For example:

or @CrossOrigin

```
@ Configuration
@ EnableWebMVC
public class WebConfig implements WebMvc Config {
    //...
}.
```

#6. 400 : bad request.

404: Not found

500 : Internal server Error

#7. Both are container. Both are interface.
ApplicationContext extends from BeanFactory.
Both can do bean instantiation / wiring.
~~But~~ ApplicationContext has more features, like convenient massager.. (for language transfer)

#8. Singleton : only one instance. But it is not thread safe, since some requests may share the same instance.
And it is default.
prototype : any number of object instances usually used for domain object.
request : for http request
session : for http session
application : for a servletContext.
webSocket : for websocket.

#9. driver, url, username, password.

#10. @Autowired
@Qualifier ("publicUtil") you can name its id.

#11. ① contructor-based

② setter - based

③ Field.-based.

#12. for service. @ Service

for DAO. @ Repository.

#13. ① Go to ~~website~~ : spring initianlizr.

② choose spring version, ^maven groupID, artifactID,

④ add dependencies : web, jpa,

⑤ click the zip button and save zip file to local computer

⑥ unzip the ~~file~~ project and import into IntelliJ.

⑦ when using jpa to connect, just use annotation.

#14.
```
@ RestController
public class ItemViewController {
        @GetMapping ("/product/{sequence}/search
        public Product getProduct (@PathVariable ~~[illegible]~~ int sequence,
                                   @RequestParam (name = "soureId") String ~~[illegible]~~ sourceId,
                                   @RequestParam (name = "log loc") String ~~[illegible]~~ logloc,
                                   ~~[illegible]~~ ){
              Product p = service.getProduct(sequence, sourceId, logloc);

              return p;
        }
}
```

#15. @Service
```
public class ItemViewService{
    @Autowired
    ~~public~~
    private ItemDao dao;
    public ItemDao itemDao(){
        return new ItemDao();
    }
}.
```

#16. ① Annotation:
```
@Configuration
public class ServiceHelper{
    @Bean
    public ItemViewHelper helper(){
        //....
        return new ItemViewHelper();
            @Autowired
    }
```

② XML:
```
<Beans>
    <Beans id=" ~~[illegible]~~ service"  class="demo.service">
        < constructor-arg  ref=" ~~[illegible]~~ ItemViewHelper">
    </Beam>
    ~~[illegible]~~ <Beans ~~ref~~ id="ItemViewHelper"  Class="demo. ItemViewHelper">

    </Bean>
<Beans>
```

#17.

```
@Autowired
Environment env;

public void getFinancial () {
    String ~~return~~ finaricial = env.getProperty("walmart.es.service" ~~financial~~);
    //...
}.
```