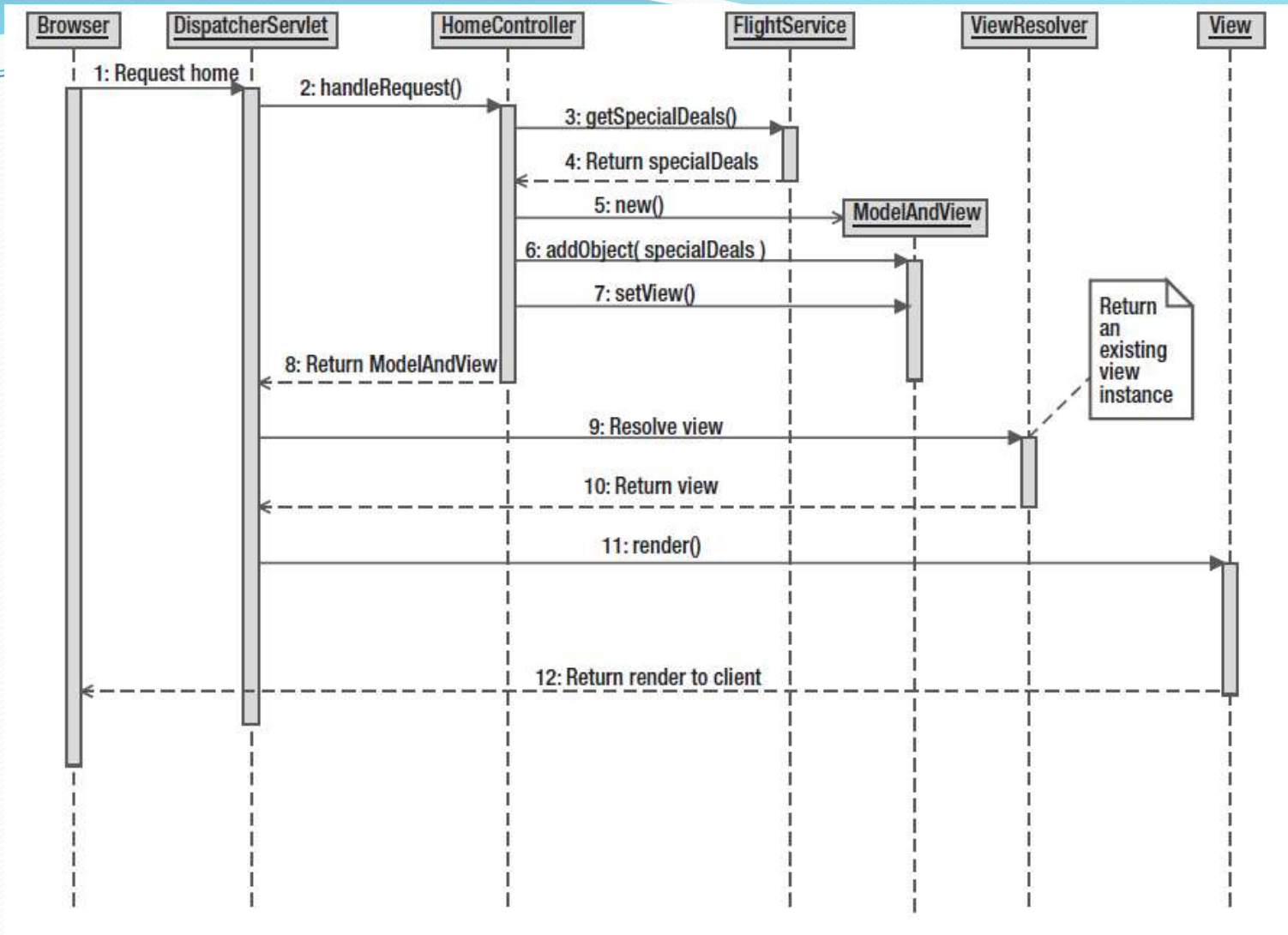
Configuration

- <servlet>
- <servlet-name>dispatcher</servlet-name>
- <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class>
- <init-param><param-name> contextConfigLocation</param-name>
- <param-value>/WEB-INF/todo-servlet.xml</param-value>
- </init-param><load-on-startup>1</load-on-startup>
- </servlet>


Browser
DispatcherServlet
HomeController
FlightService
ViewResolver
View
1: Request home
2: handleRequest()
3: getSpecialDeals()
4: Return specialDeals
5: new()
ModelAndView
6: addObject(specialDeals)
7: setView()
Return an existing view instance
8: Return ModelAndView
9: Resolve view
10: Return view
11: render()
12: Return render to client

BeanFactory

- The ApplicationContext is a specialization of a BeanFactory, which is the registry of all the objects managed by Spring.
- Under normal circumstances, the BeanFactory is responsible for
  - creating the beans,
  - wiring them with any dependencies, and
  - providing a convenient lookup facility for the beans.
- Other interfaces are there to help to define the life cycle of beans managed by the BeanFactory
- Spring configuration consists of at least one and typically more than one bean definition that the container must manage
- Typically you define service layer objects, data access objects (DAOs), presentation objects

```
ApplicationContext context = new ClassPathXmlApplicationContext(new
String[] {"services.xml", "daos.xml"});
```

ApplicationContext and BeanFactory

- Applications typically interact with an ApplicationContext instead of a BeanFactory
- Additionally ApplicationContext provides internationalization (i18n) facilities for resolving messages

- **Spring** **@Bean** annotation tells that a method produces a **bean** to be managed by the **Spring** container.
- It is a method-level annotation.
- During Java configuration (@Configuration), the method is executed and its return value is registered as a **bean** within a BeanFactory

DispatcherServlet

- The WebApplicationContext is a special ApplicationContext implementation that is aware of the servlet environment and the ServletConfig object
- For interfaces like ViewResolvers, the DispatcherServlet can be configured to locate all instances of the same type
- The DispatcherServlet uses the Ordered interface to sort many of its collections of delegates.
  - This is done through a property named *order*
- Usually, the first element to respond with a non-null value wins
- During initialization, the DispatcherServlet will look for all implementations by type of HandlerAdapters, HandlerMappings, HandlerExceptionResolvers, and ViewResolvers
- The DispatcherServlet is configured with default implementations for most of these interfaces.
  - This means that if no implementations are found in the ApplicationContext (either by name or by type), the DispatcherServlet will create and use them

Authentication and Authorization

WebSecurityConfigurerAdapter

```
/***https://spring.io/guides/gs/securing-web/*/
/***https://docs.spring.io/spring-cloud-skipper/docs/1.0.0.BUILD-SNAPSHOT/reference/html/configuration-security-enabling-https.html*/
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http
                        .authorizeRequests()
                                .antMatchers("/").permitAll()
                                .anyRequest().authenticated()
                                .and()
                        .formLogin()
                                //.loginPage("/login")
                                .permitAll()
                                .and()
                        .logout()
                                .permitAll()
                                .and()
                                .httpBasic();
                http
            .csrf().disable();

        }
        @Bean
        @Override
        public UserDetailsService userDetailsService() {
                UserDetails user =
                         User.withDefaultPasswordEncoder().username("user") .password("password").roles("USER").build();
                return new InMemoryUserDetailsManager(user);
        }
}
```

Settings for 8443

```
keytool -genkey -noprompt -alias tomcat-localhost -keyalg RSA -keystore
C:\Users\chand\localhost-rsa.jks -keypass 123456 -storepass 123456 -dname
"CN=tomcat-cert, OU=JU, O=JU, L=WB, ST=WB, C=IN“
```

```
<Connector
        protocol="org.apache.coyote.http11.Http11NioProtocol"
        port="8443" maxThreads="200"
        scheme="https" secure="true" SSLEnabled="true"
        keystoreFile="C:\my-cert-dir\localhost-rsa.jks"
        keystorePass="123456"
        clientAuth="false" sslProtocol="TLS"/>
```

Spring

```
keytool -genkey -alias skipper -keyalg RSA -keystore
c:\User\user1\skipper.keystore   -validity 3650 -storetype
JKS -dname "CN=localhost, OU=Spring, O=Pivotal, L=Holualoa,
ST=HI, C=IN" -keypass skipper -storepass skipper
```

- This method generates the key needed for HTTPS.
- Excute this command from jdk/bin of your machine
- Move the generated keystore file to the “resources” folder of your application

Confidentiality

Keep the following methods in the file where the main method is present

```
 @Bean
public ServletWebServerFactory servletContainer() {
TomcatServletWebServerFactory tomcat = new
TomcatServletWebServerFactory() {
@Override
 protected void postProcessContext(Context context) {
SecurityConstraint securityConstraint = new
SecurityConstraint();
securityConstraint.setUserConstraint("CONFIDENTIAL");
SecurityCollection collection = new SecurityCollection();
collection.addPattern("/*");
securityConstraint.addCollection(collection);
context.addConstraint(securityConstraint);             }
};
tomcat.addAdditionalTomcatConnectors(redirectConnector());
return tomcat;    }
```

Confidentiality

- Keep the following methods in the file where the main method is present

```
 private Connector redirectConnector() {
        Connector connector = new
Connector("org.apache.coyote.http11.Http11NioProtocol");
        connector.setScheme("http");
        connector.setPort(8080);
        connector.setSecure(false);
        connector.setRedirectPort(8443);
        return connector;
    }
```