Results from the Java EE 8 Community Survey

Our Java EE 8 Community Survey generated several thousand responses! Our thanks to all of you who contributed!

Due to the large number of questions that we wanted to ask, we ran the survey in two parts. Here is a summary of the results.

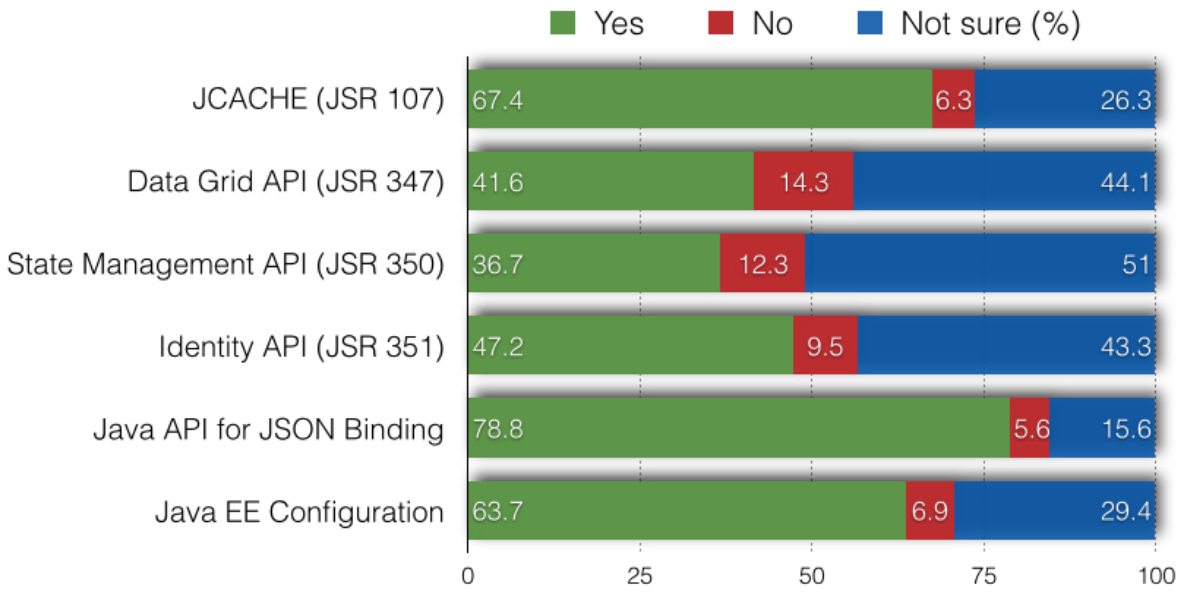
Part 1

The first part of the survey covered topics in the areas of :

- New JSRs
- Web tier / HTML 5
- CDI alignment
- NoSQL

New JSRs

Which of these APIs do you think is important to be included into Java EE 8?



Approximately 79% of you thought that we should provide a Java API for JSON Binding and 67% thought that we should include JCache into the Java EE platform. A JSR for Java EE Configuration also received a strong showing at 64%.

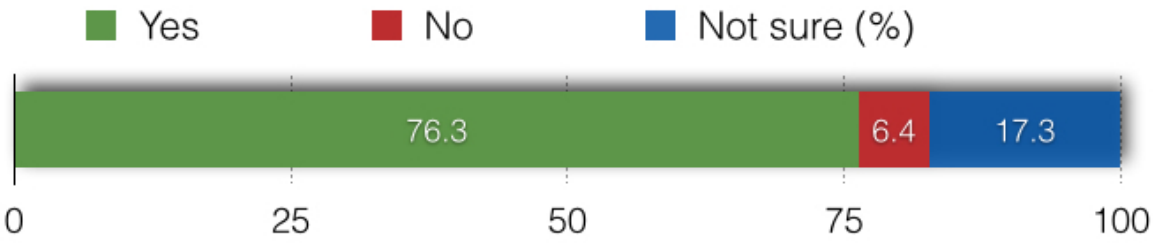
Respondents were less sure about the inclusion of the Identity API, Data Grid, or State Management. Responses for these JSRs showed a high "Not Sure" rate, which for Data Grid and State Management exceeded the "Yes" rate.

Many of the comments echoed support for the APIs listed in this question, some with further qualifications. For example, people wanted to make sure that caching, state management, and data grids were well aligned.

A number of respondents also cautioned us against proliferating the number of specifications and APIs, against redundant ways of doing things, and against introducing complexity. Several people advised that APIs should be well integrated with CDI. A few people asked us to add support for modularity.

Server-sent events

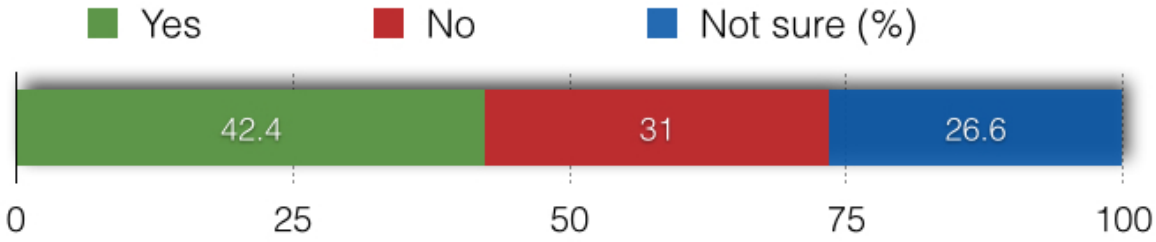
Should we also standardize a Java API for server-sent events?



Respondents were clearly strongly in favor of adding support for server-sent events.

JavaScript on the Server

Should we define APIs to support use of JavaScript on the server (e.g., Project Avatar)?



With only a 42% "Yes" rate, support for this was quite reserved.

HTML5

What should we add to better enable HTML5 mobile apps?

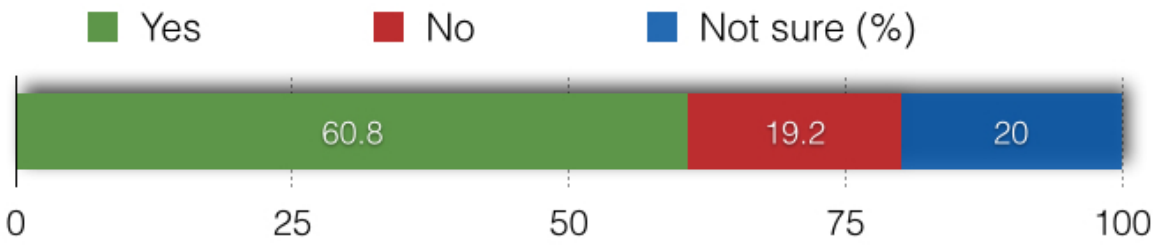
Responses to this open-ended question were fairly diverse.

One group of responses focused on suggestions that take action mostly or entirely in the user agent. Within this group, one third suggested standardizing a JavaScript framework or anointing one particular framework and building integration that assumes that framework. The remainder were split among suggestions to offer some kind of direct binding between one or more particular pieces of Java EE and the user agent, suggestions to standardize one of the client adapter layers such as Sencha touch or Apache Cordova, and suggestions to put more Java on the user agent.

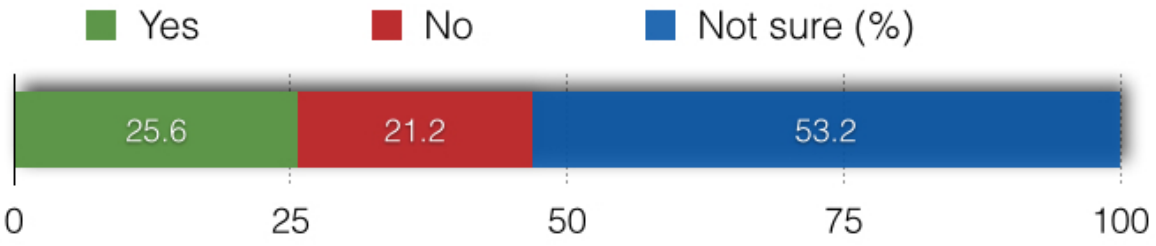
Another group of responses were focused on JSF. Most suggested building further on JSF's existing HTML5 support. Many liked the ability of JSF to cleanly address responsive web design concerns by virtue of its renderkit concept. The remainder suggested improvements to the markup generated by JSF.

MVC Support

Should Java EE provide support for MVC, alongside JSF?



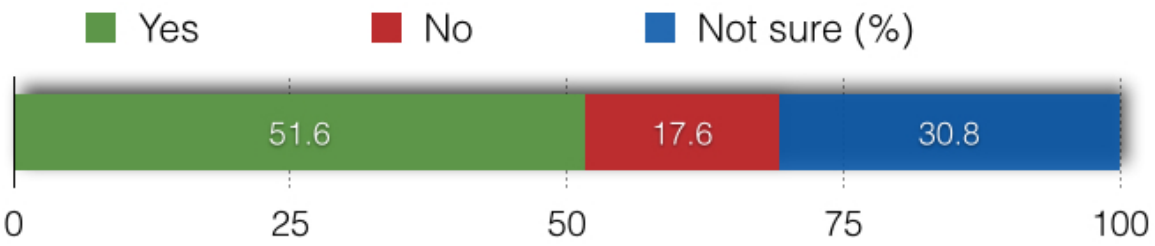
Is there any one de-facto standard technology in this space to which we should look for inspiration?



While 61% of respondents answered "Yes" to the question as to whether we should provide MVC support alongside JSF, only 26% felt they could identify an existing technology here. Although this weak "Yes" response rate somewhat devalues the answers that we received, suggestions around Spring MVC or an approach similar to that of Spring MVC accounted for about 42% of the responses to the "If so, which?" part of this question. Angular and Play! were the next most frequently mentioned.

Client API for TSA

Should we investigate standardizing a client API for Thin Server Architecture?

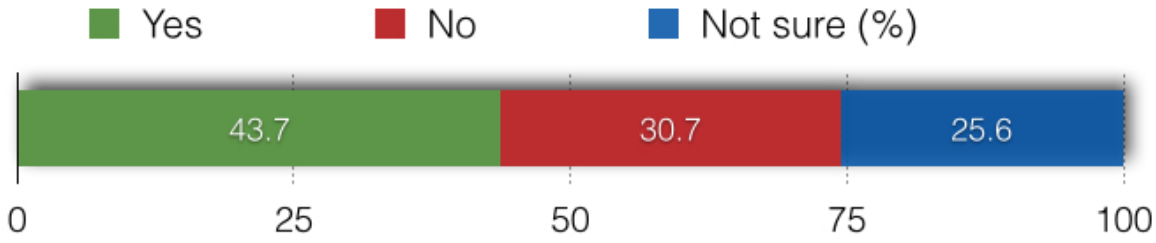


A weak (52%) majority of the respondents thought that we should investigate this area.

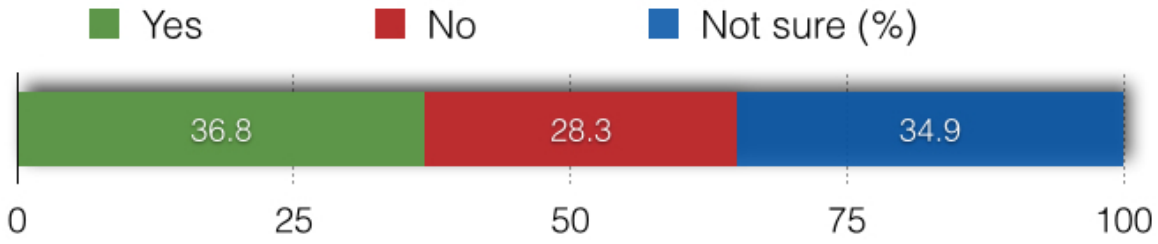
Templating

The next group of questions focused on templating frameworks. Support here was fairly weak and for two of the questions the "Not Sure" rates rivaled or exceeded the "Yes" rates.

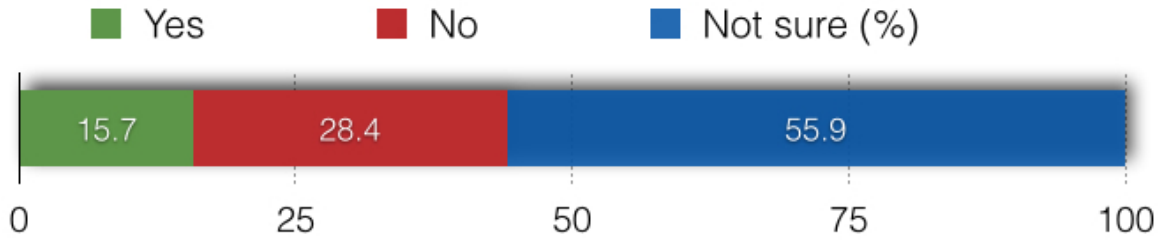
Should we define a (new) standard templating framework?



Should we extract Facelets from JSF as the standard template engine for Java EE?



Should we standardize on another existing technology?

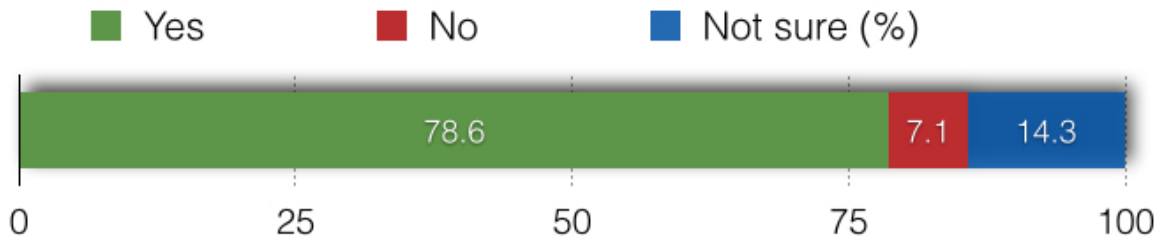


Only a small fraction of respondents answered the open-ended "If so, which?" part of this last question. Thymeleaf, Freemarker, and Velocity were the most frequently mentioned here.

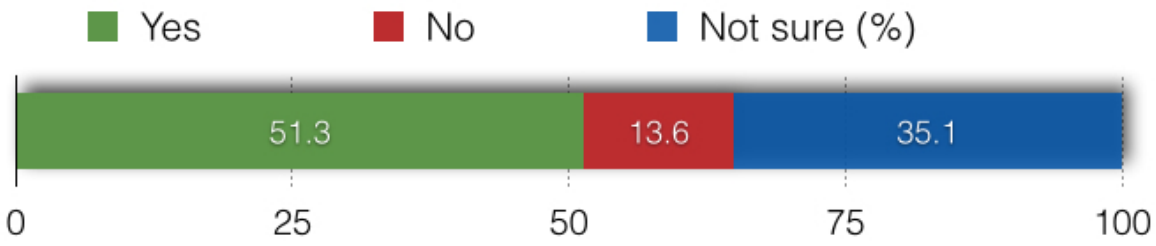
CDI

The next four questions focused on continued CDI alignment. This was one of the focus areas of Java EE 7.

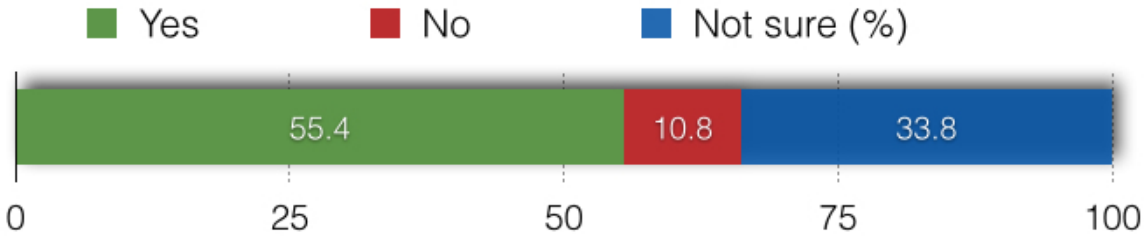
Should we consider adding Security Interceptors in Java EE 8?



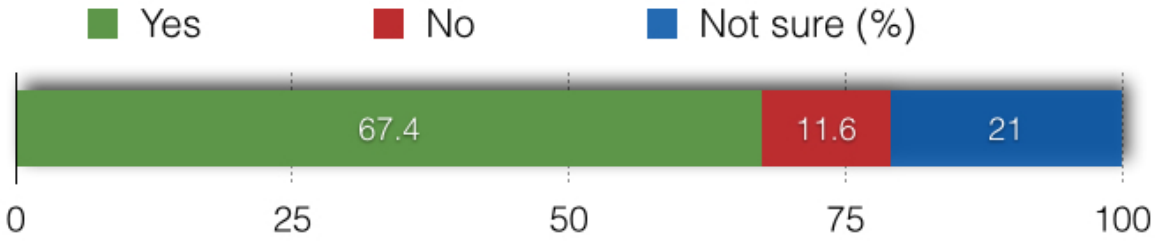
Should we consider replacing @Resource with Implicit Producers?



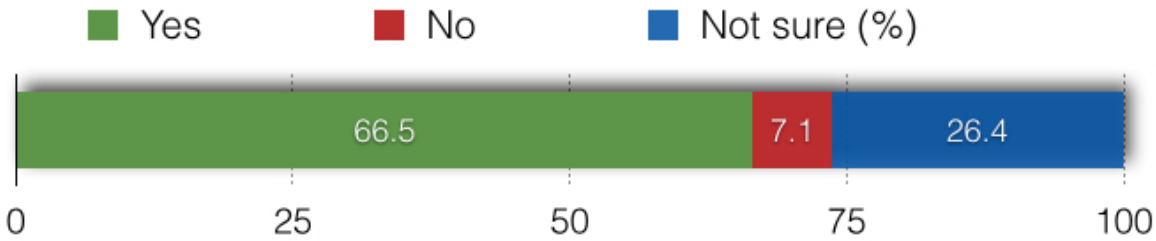
Should we consider expanding the use of CDI stereotypes?



Should we consider generalization of the EJB Timer Service for use by other managed beans?



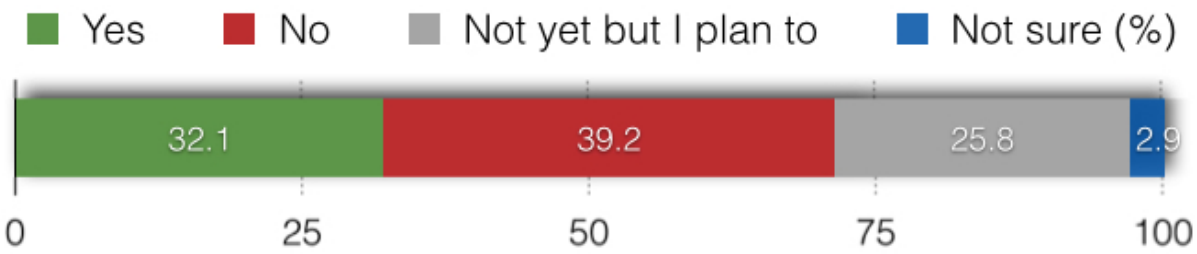
Should we consider expanding CDI events and observer methods for other services (e.g., timeout events, JMS messages, …)?



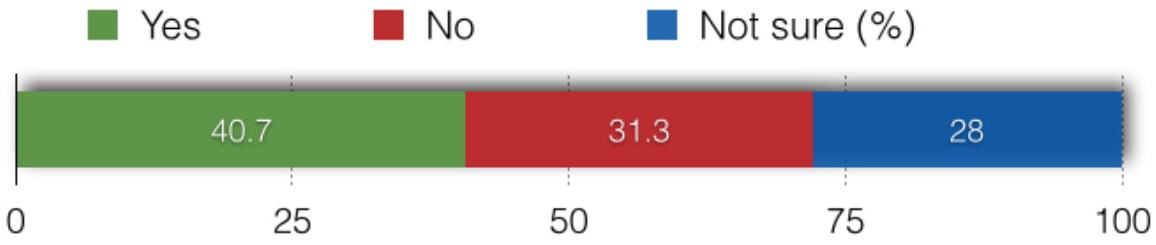
All of these items received a majority of "Yes" responses, although the strength of these was significantly higher for security interceptors at 79%, generalization of the EJB Timer Service at 67%, and expanding the use of CDI events and observer at 67%. Expanding the use of CDI stereotypes received 55% "Yes" votes and Implicit Producers received only a weak majority at 51%.

NoSQL

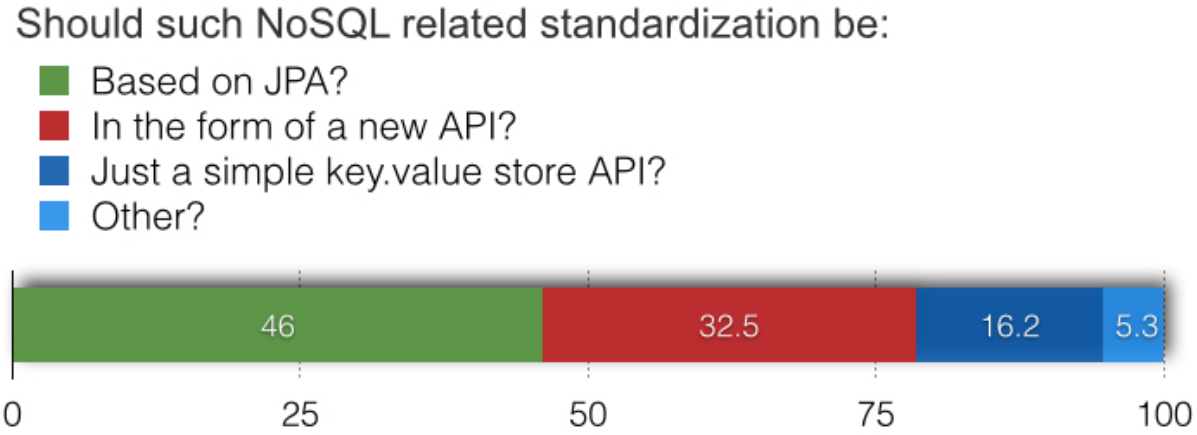
Are you using NoSQL today?



Is it time yet to standardize in this area?



Should such NoSQL related standardization be:



While relatively few indicated that they were using NoSQL today, approximately 41% thought it was time yet to standardize in this area.

When asked on what such standardization should be based, a relatively small number of participants responded. The largest number of these (although not a majority) thought such standardization should be based on JPA.

We received even fewer responses to our "Other?" option as to whether NoSQL standardization, if any, should be based on an API other than JPA or a simple key/value store API. Most of the comments here echoed the existing choices and elaborated on those options. Some respondents thought that we should offer multiple APIs.

Additional Comments

We concluded Part 1 of our survey with an open-ended question designed to solicit input on things we might have missed and to enable more general feedback.

By far the most common comment was "thanks" - thanks for Java, thanks for all the work we do, thanks for the survey, etc.

The most common technical theme in the comments was about security. Many people want improvements in Java EE security, especially to improve the portability of applications.

The second most common technical theme was around simplicity - keep it simple, make it simple, make it simpler, remove X to make it simpler, don't add Y to keep it simple, don't do Z because that would make it more complicated, etc. Often "simple" was equated with "smaller".

Part 2

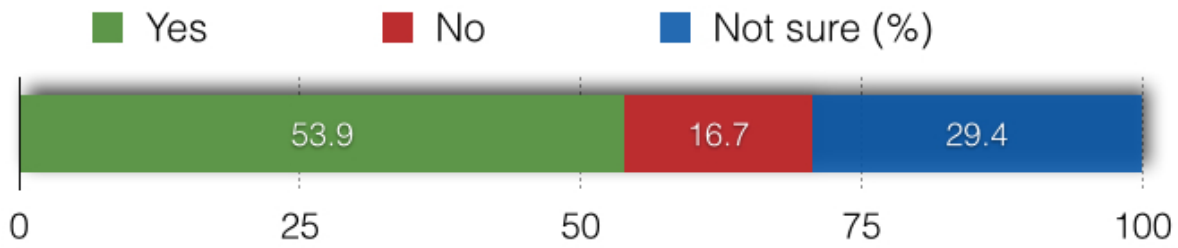
The second part of the survey covered topics in the areas of :

- Cloud (PaaS / SaaS / Multitenancy)
- Logging
- Security
- Testability
- Deployment and Management
- Profiles and/or Pruning

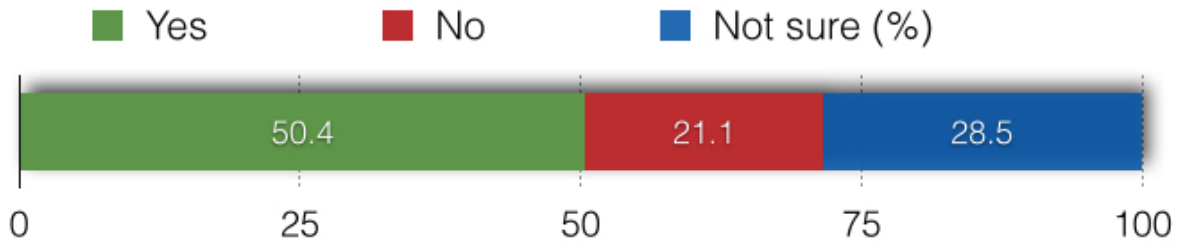
Cloud

The first group of questions addressed the areas of PaaS, SaaS, and multitenancy.

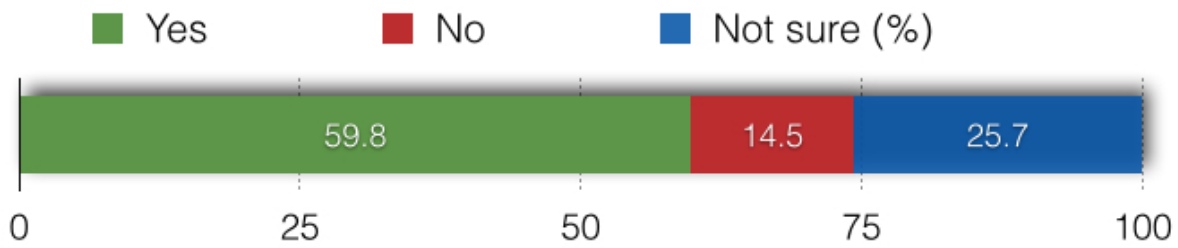
Is it time to standardize support for PaaS?



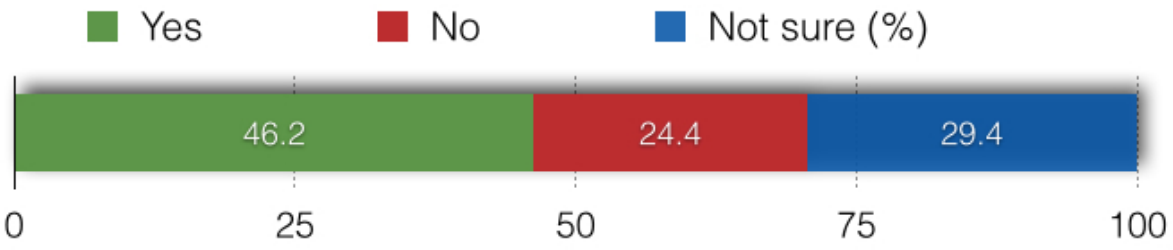
Is it time to standardize support for SaaS?



Is it time to standardize support for single application deployment with multiple application runtime instances in a single application server instance?



Is it time to standardize support for multiple tenants to run within a single application instance?



A relatively weak majority supported standardization in the PaaS and SaaS areas (54% and 50% respectively).

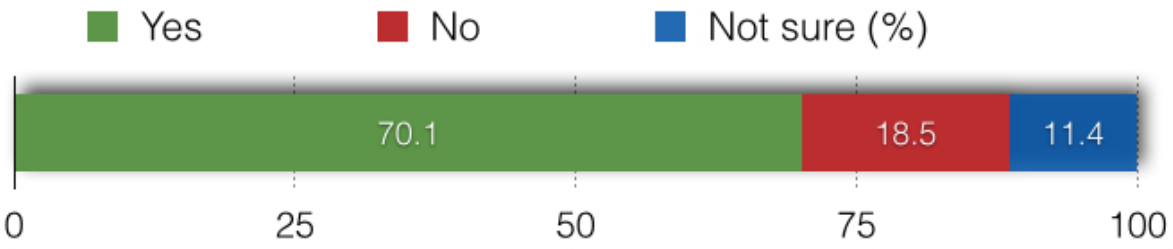
In response to the question as to whether it was time to standardize support for a single application deployment with multiple application runtime instances in a single application server instance, approximately 60% of the respondents thought that we should. However, only 46% thought that it was time to standardize support for multiple tenants to run within a single application instance.

In the open-ended comments, a majority felt that Java EE should do more to address the cloud space, and that starting small and adding features was the right way to go. Some respondents asked for JVM support (e.g., better isolation, Jigsaw) before tackling multitenancy. Some commented that customers felt safer with multi-instance multitenancy, citing concerns around security and isolation. Others thought that we should start preparing for multi-instance multitenancy now.

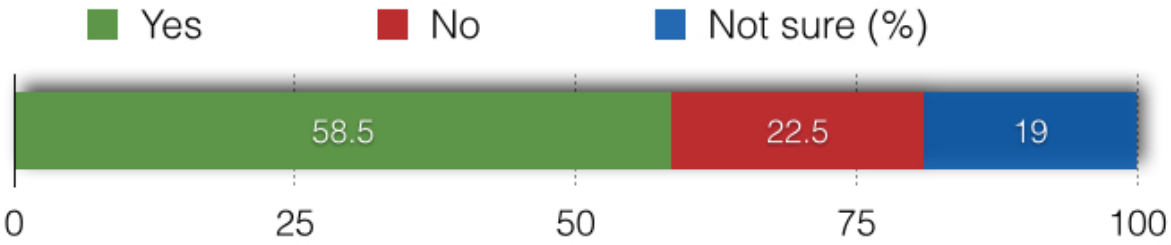
Logging

The next set of questions addressed the logging area. All of these questions received a majority of "Yes" responses.

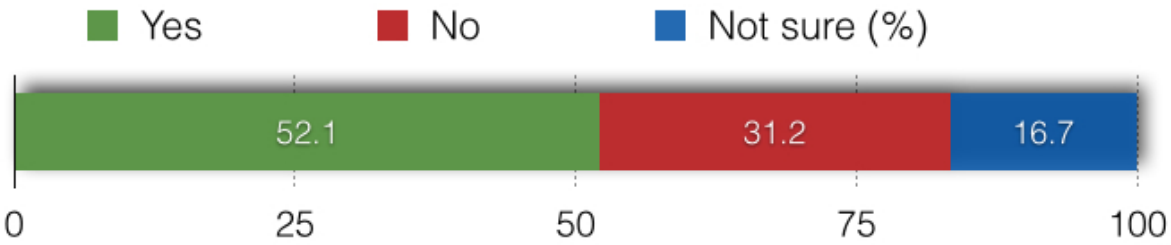
Should we standardize the use of java.util.logging by applications?



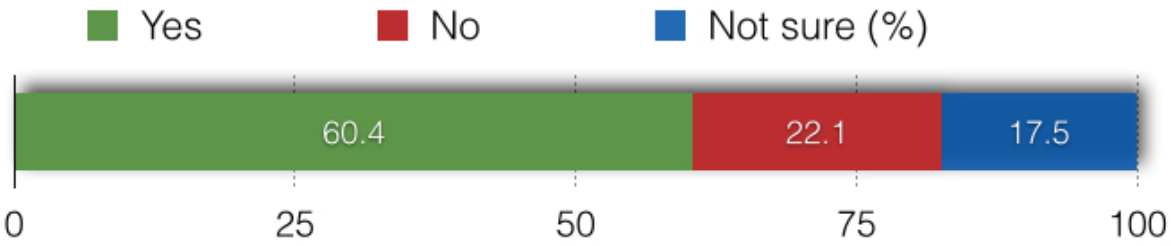
Should we standardize the names of loggers used by the application server?



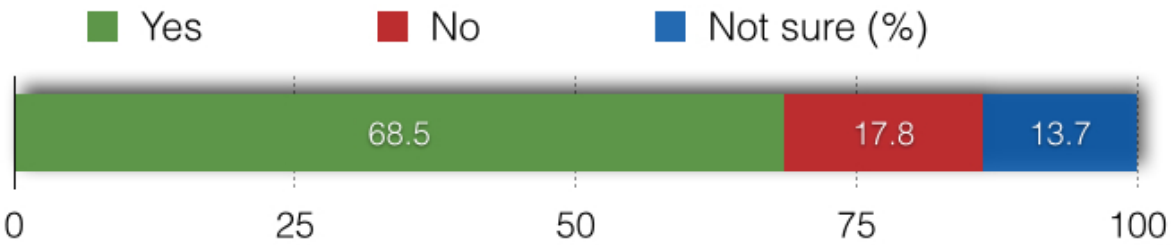
Should we standardize the format of log files?



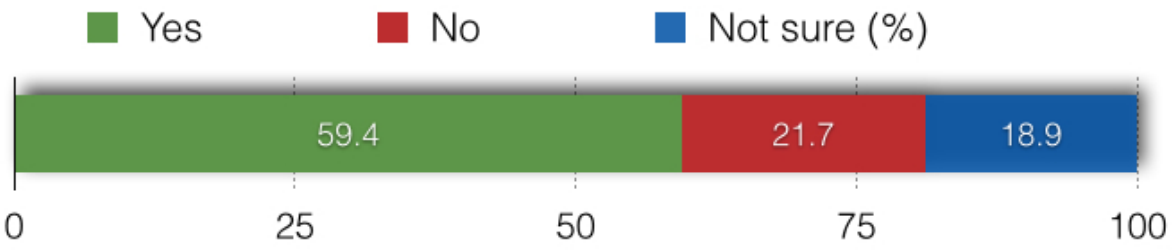
Should we standardize the management of log files?



Should we standardize the logging configuration for application loggers?



Should we standardize access to log messages by applications?



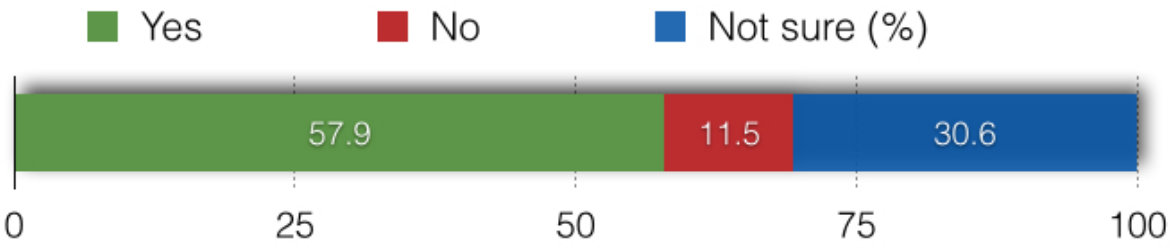
Respondents strongly supported standardizing use of java.util.logging by applications, standardizing the management of log files, and standardizing the logging configuration for application loggers. A majority of respondents also supported standardizing the names of loggers used by the application server, the format of log files, and the access to log messages by applications.

Of the small fraction of participants who added comments, most of these strongly urged for improvements in logging – either by the revision of java.util.logging, or replacement of its use by either slf4j, slf4j with LogBack, or log4j.

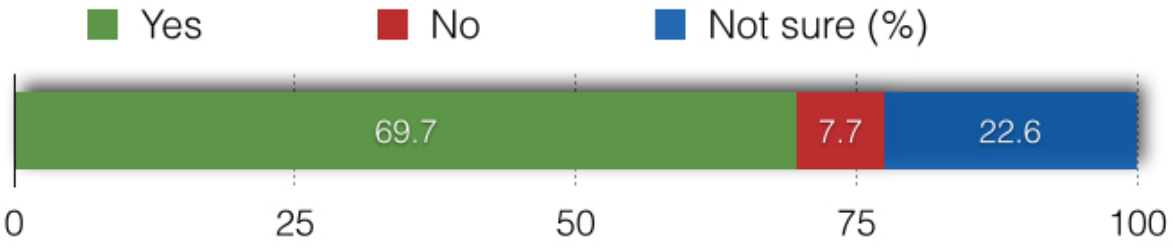
Security

Most of the suggested improvements in the security area received strong support.

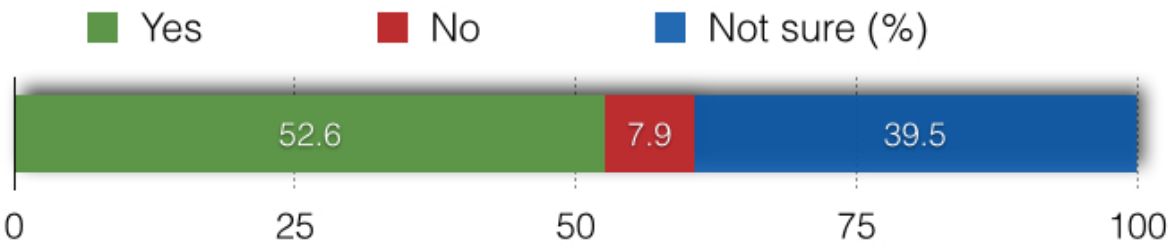
Should we add support for password aliases (including the ability to provision credentials along with the application)?



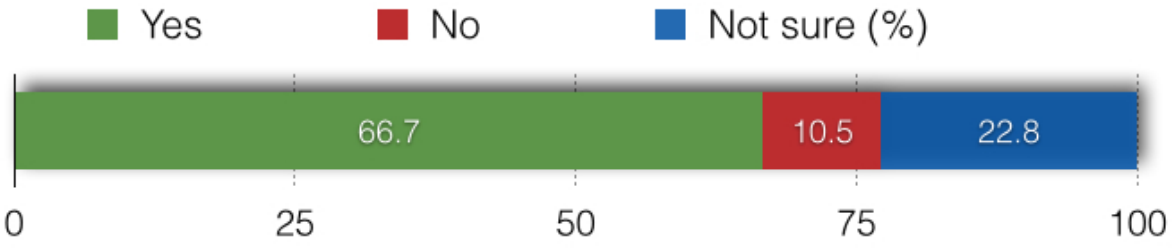
Should we standardize group-to-role mapping?



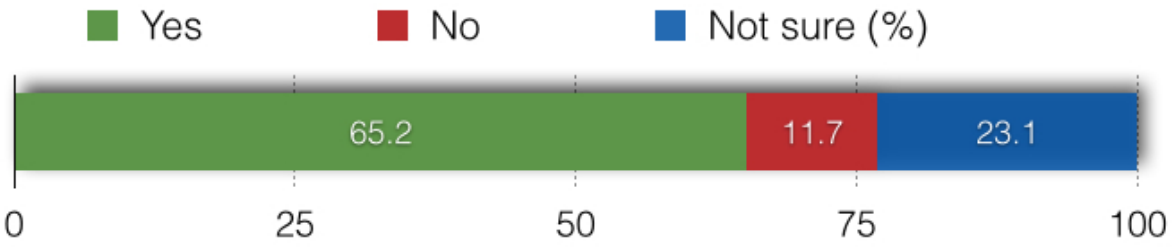
Should we simplify JASPIC?



Should we simplify authorization by introducing an EL-enabled authorization annotation?



Should we standardize on requirements for simple security providers and their configuration?



Approximately 58% thought we should add support for password aliases, including the ability to provision credentials along with the application.

70% thought that we should standardize group-to-role mapping.

53% thought we should simplify JASPIC authentication.

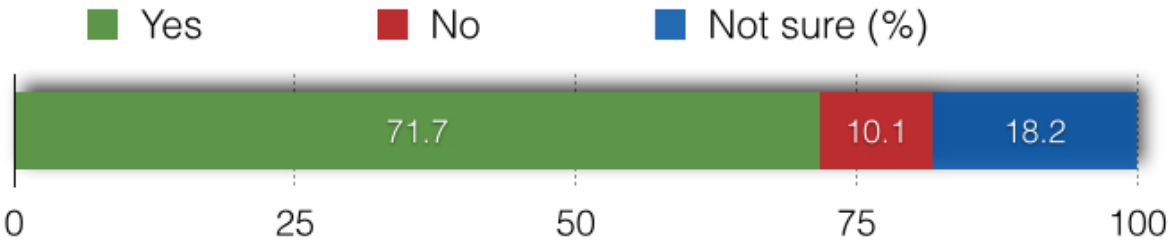
67% thought that we should simplify authorization and make it more flexible by introducing EL-based authorization annotations, introducing a capability more general than use of @RolesAllowed and simpler than use of interceptors to do programmatic authorization.

65% thought we should standardize on requirements for simple security providers and their configuration.

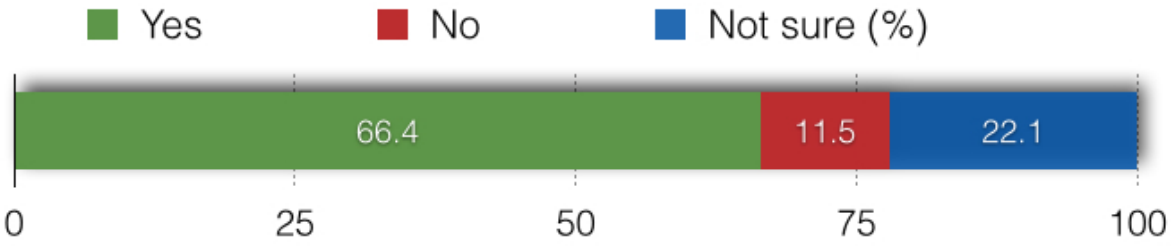
Embedded Containers and Testability

The next set of questions addressed the area of embedded containers and other means by which we could improve testability. This area received strong support

Should we define an embedded web container?



Should we define an embedded Java EE container?

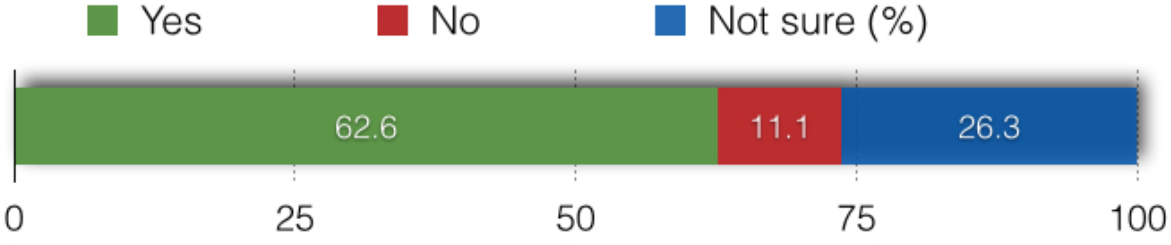


The free form suggestions also echoed the desire for embedded containers. Some also suggested that we standardize in the area of testing frameworks.

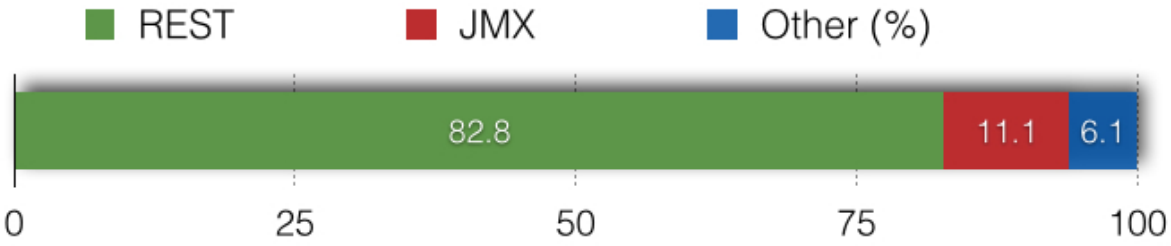
Deployment, Management, and Monitoring

The addition of new APIs for deployment, management, and monitoring received strong support. When forced to choose between REST APIs and JMX APIs, the overwhelming majority chose REST. In the comments, a number of respondents stated the desire to have both REST APIs and JMX APIs.

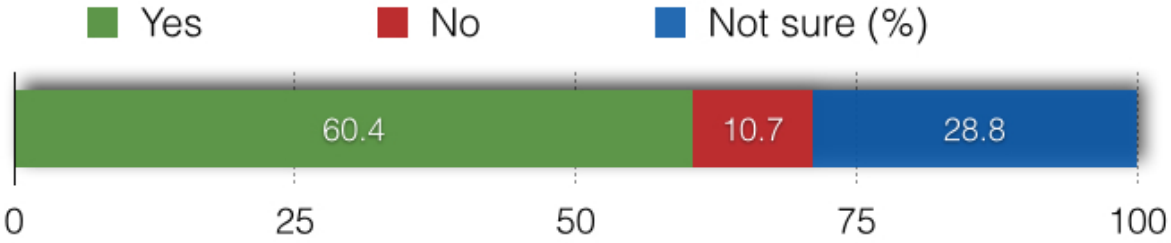
Should we define new APIs to deploy and manage applications?



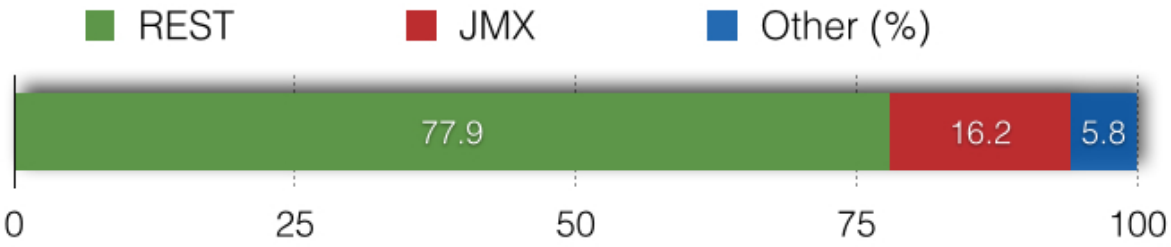
Should such new Deployment and Management APIs be REST APIs or JMX APIs?



Should we define new APIs to monitor applications?

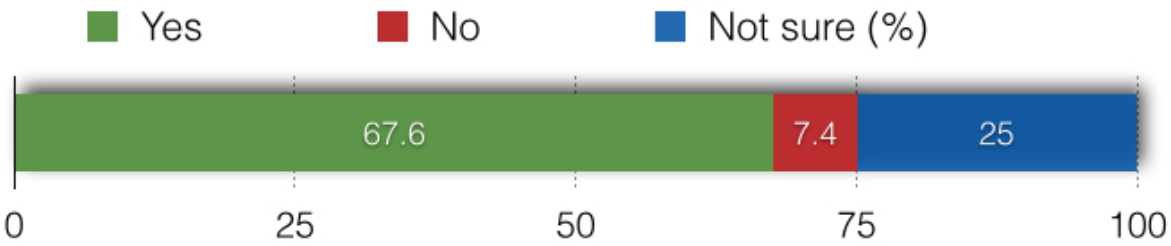


Should such new monitoring APIs be REST APIs or JMX APIs?



The addition of a new trace context API also received strong support:

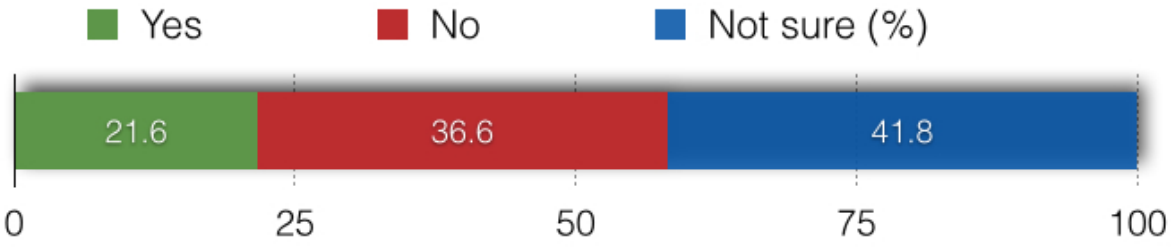
Should we define a trace context API to track a request as it goes through the system?



Profiles

Support for introducing new profiles was very low.

Should we define any additional Java EE profiles?

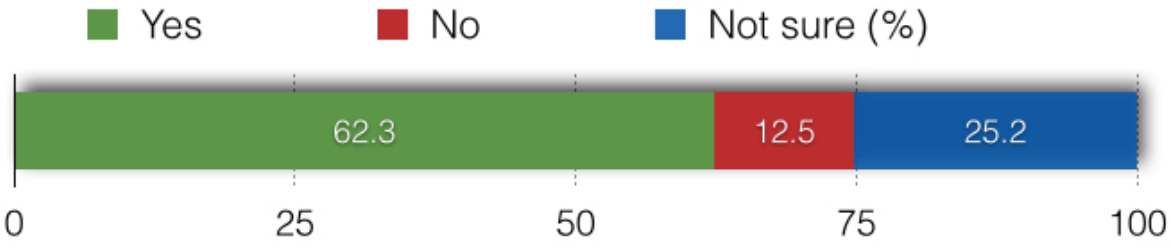


Suggestions in the open-ended comments section here were quite diverse, and there seemed to be no consensus in this area. Some people suggested a profile without a UI layer, suitable for JavaScript front-end/mobile applications. Some suggested that general modularity was a better idea than specific profiles. A few wanted a batch profile, a few a messaging profile, and a few a cloud profile.

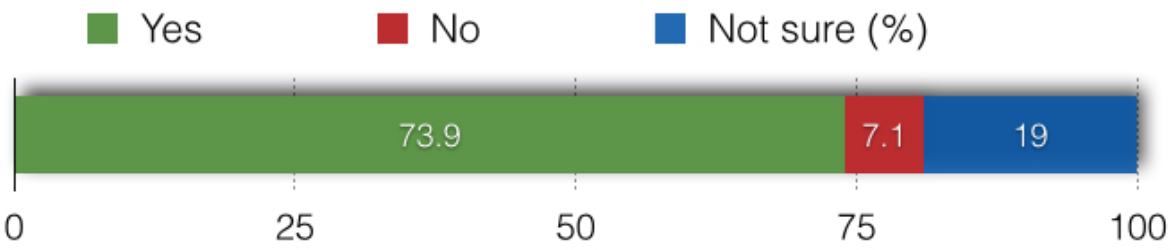
Pruning

The next set of questions dealt with "pruning" (i.e., defining that a technology was optional for a Java EE Platform vendor to implement).

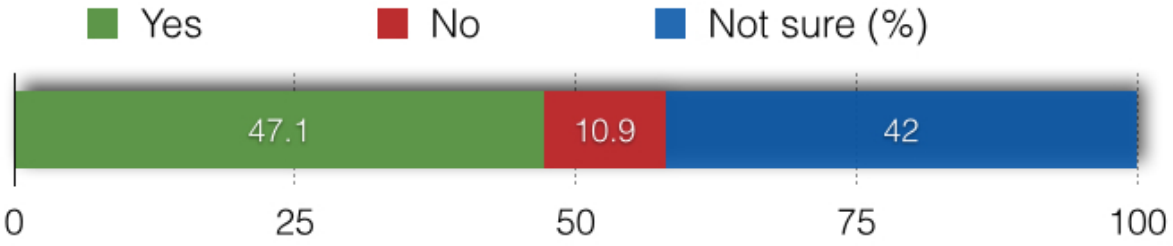
Should we prune CORBA, including support for interoperability by means of IIOP?



Should we prune EJB 2.x remote and local client view (EJBObject, EJBLocalObject, EJBHome, and EJBLocalHome interfaces)?



Should we prune JSR-77 (J2EE Management)?



While approximately 62% thought that we should prune CORBA, including support for IIOP interoperability, an even larger number (74%) thought that we should prune the EJB 2.x remote and local client view (EJBObject, EJBLocalObject, EJBHome, EJBLocalHome). Support was lacking for pruning J2EE Management (JSR 77), with only a 47% yes rate and 42% "Not sure".

Responses to our question as to whether there were other technologies that we should prune were quite varied. The most frequently mentioned technologies were, in order, JSF (or parts of JSF), JSP, and CORBA. A few people suggested pruning JAX-RPC, EJB CMP, and JSR-88, which were made optional as of Java EE 7. A number of people also requested that we not prune CORBA or IIOP interop.

Additional Feedback

The survey concluded with a general open-ended request for feedback. The most requested item was for simplifications and enhancements in the security area. We also received many requests for continuing and improving CDI alignment and for more standardization in the configuration area.