全栈常见面试题目及答案

（本文档来自九章学员整理并无偿分享）

Q1: What is Inversion of Control?

Inversion of control is a broad term but for a software developer it's most commonly described as a pattern used for decoupling components and layers in the system.

For example, say your application has a text editor component and you want to provide spell checking. Your standard code would look something like this:

```
public class TextEditor {

private SpellChecker checker;
public TextEditor() {
this.checker = new SpellChecker();
}
}
```

What we've done here creates a dependency between the TextEditor and the SpellChecker. In an IoC scenario we would instead do something like this:

```
public class TextEditor {

private IocSpellChecker checker;

public TextEditor(IocSpellChecker checker) {
this.checker = checker;
}
}
```

You have *inverted control* by handing the responsibility of instantiating the spell checker from the TextEditor class to the caller.

```
SpellChecker sc = new SpellChecker; // dependency
TextEditor textEditor = new TextEditor(sc);
```

Q2: What is Bridge pattern?

Bridge pattern is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under *structural* pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

The bridge pattern is useful when both the class and what it does vary often. The class itself can be thought of as the abstraction and what the class can do as the implementation. The bridge pattern can also be thought of as two layers of abstraction.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

The example of bridge pattern implementation is when:

```
----Shape---
/ \
Rectangle Circle
/ \ / \
BlueRectangle RedRectangle BlueCircle RedCircle
```

refactored to:

```
----Shape--- Color
/ \ / \
Rectangle(Color) Circle(Color) Blue Red
```

or in general when:

```
A
/ \
Aa Ab
/ \ / \
Aa1 Aa2 Ab1 Ab2
```

refactored to:

```
A N
/ \ / \
Aa(N) Ab(N) 1 2
```

Q3: What are the success factors for Continuous Integration?

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment

- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

Q4: Explain a use case for Docker

- Docker a low overhead way to run virtual machines on your local box or in the cloud. Although they're not strictly distinct machines, nor do they need to boot an OS, they give you many of those benefits.
- Docker can encapsulate legacy applications, allowing you to deploy them to servers that might not otherwise be easy to setup with older packages & software versions.
- Docker can be used to build test boxes, during your deploy process to facilitate continuous integration testing.
- Docker can be used to provision boxes in the cloud, and with swarm you can orchestrate clusters too.

Q5: Explain the main difference between REST and GraphQL

The main and most important difference between REST and GraphQL is that *GraphQL is not dealing with dedicated resources, instead everything is regarded as a graph and therefore is connected and can be queried to app exact needs*.

Q6: What is Event Loop?

Node.js is a single threaded application but it support concurrency via concept of event and callbacks. As every API of Node js are asynchronous and being a single thread, it uses async function calls to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

Q7：List three key things to consider when coding with SEO in mind.

In order to build a site optimized for organic search engine rankings, it is important to implement certain standards throughout the code. These include:

- Specifying an alt tag on images
- Using the correct HTML tags for content hierarchy
  i.e., <h1>/<h2>/<h3> and p
- Connect the site to the company’ s social pages
- Add an XML sitemap
- Avoid broken links
- Use vanity/friendly URLs (human readable)
- Add a robots.txt file
- Integrate Google analytics (or alternative)
- Specify a favicon, bonus for specifying browser specific icons
- Ensure lightning fast page load time
- Avoid JavaScript errors
- Optimize assets (including minification)
- Enable and force SSL
- Specify unique titles for each page without exceeding 70 characters
- Include a meta description on each page
- Ensure there is enough content with enough relevant keywords (search engines will penalize your site if all pages are one-sentence pages)
- Leverage browser caching
- Avoid W3C markup validation errors
- Specify relevant meta tags

Q8：If you were to write an endpoint for checking if a resource exists, what path and method would you use?

The purpose of this question is to test the candidate's knowledge of RESTful API standards. A common mistake when building endpoints is to use descriptive verbs in the path. For example:

- GET /users/search
- GET /posts/create

Instead, a truly RESTful path should only contain nouns—the method used on the endpoint should determine the action. For example:

- POST /users (create a user model)
- PUT /users/{id|slug} (replace a user model)
- PATCH /users/{id|slug} (update part of a user model)
- DELETE /users/{id|slug} (delete a user model)
- GET /users/{id|slug} (retrieve a user model)

Determining whether a resource exists is an action that is frequently required in APIs, but is rarely done correctly according to the RESTful and industry standards. The commonly accepted way to determine if a resource exists, using the above "user" resource as an example, is like so:

- HEAD /users/{id|slug}

This request will use the least amount of bandwidth as it will return no data, simply just a 200 (resource exists) or 404(resource does not exist) HTTP status.

Q9：List five or more ways you could optimize a website to be as efficient and scalable as possible.

Optimizing websites is an art that few are familiar with. The more the engineer is able to list off the top of their head, the more likely they are to do all of the following naturally as they code instead of having to return later.

(Also, typically a professionally constructed site should score over 75 percent when analyzed by gtmetrix.com, which can also serve as a checklist.)

- Optimize all assets
- Place all assets on a separate, cookie-free domain. Using a CDN is best
- Avoid inline JavaScript and CSS
- Enable gzipping
- Ensure all code is minified
- Defer parsing of JavaScript
- Use srcset for responsive images
- Leverage browser caching
- Reduce DNS lookups
- Avoid duplicate code
- Avoid URL redirects
- Enable HTTP keep-alive
- Serve scaled images
- Use image sprites where appropriate
- Prefer asynchronous resources
- Specify the character set at server level
- Avoid CSS @import
- Specify a cache validator
- Minimize request size
- Avoid bad requests and 404s
- Specify image dimensions
- Reduce cookie size
- Make fewer HTTP requests, i.e., load as few external resources as possible
- Avoid unnecessary images; where possible, use CSS
- Ensure no validation errors with W3C

Q10：Consider the following database table design.What is wrong with the above and how could it be improved?

```
CREATE TABLE `notifications` (
`id` INT NOT NULL AUTO_INCREMENT,
`type` INT(8) NOT NULL,
`notifiable_id` INT unsigned NOT NULL,
`notifiable_type` VARCHAR(10) NOT NULL,
`relation_id_1` INT unsigned,
`relation_type_1` VARCHAR(10),
`relation_id_2` INT unsigned,
`relation_type_2` VARCHAR(10),
`updated_at` TIMESTAMP NOT NULL,
`created_at` TIMESTAMP NOT NULL,
PRIMARY KEY (`id`)
);
```

The key issue with the proposed table design are the object_id_X and object_type_X fields. It is considered poor design to increment named fields when the data could be stored in a related table like so:

Notifications Table

```
CREATE TABLE `notifications` (
`id` INT NOT NULL AUTO_INCREMENT,
`type` INT(8) NOT NULL,
`notifiable_id` INT unsigned NOT NULL,
`notifiable_type` VARCHAR(10) NOT NULL,
`updated_at` TIMESTAMP NOT NULL,
`created_at` TIMESTAMP NOT NULL,
PRIMARY KEY (`id`)
);
```

Notification Relations Table

```
CREATE TABLE `notification_relations` (
`notification_id` INT unsigned NOT NULL,
`relation_id` INT unsigned NOT NULL,
`relation_type` VARCHAR(10) NOT NULL,
PRIMARY KEY (`notification_id`)
);
```

Q11：A common issue when integrating third party services within your own API requests is having to wait for the response, and as such, forcing the user to have to wait for longer.How would you go about avoiding this? Name any relevant technologies if appropriate.

The most effective way to solve this problem is to use queues.

When a request is made to our API, a separate job is then created and added to a queue. This job will then be executed independently to the requested endpoint, thus allowing the server to respond without delay.

There are many queue providers but the most notable are:

- Amazon SQS
- Redis
- Beanstalkd

Q12：How would you prevent a bot from scraping your publicly accessible API?

If the data within the API is publicly accessible then, technically, it is not possible to completely prevent data scraping. However, there is an effective solution that will deter most people/bots: rate limiting (also known as throttling).

Throttling will prevent a certain device from making a defined number of requests within a defined time. Upon exceeding the defined number of requests, a 429 Too Many Attempts HTTP error should be thrown.

Note: It is important to track the device with more than just an IP address as this is not unique to the device and can result in an entire network losing access to an API.

Other less-than-ideal answers include:

- Blocking requests based on the user agent string (easy to circumvent)
- Generating temporary "session" access tokens for visitors at the front end. This isn't secure: Storing a secret on the front end can be reverse-engineered, thus allowing anyone to generate access tokens.

Q13：If a user attempts to create a resource that already exists—for example, an email address that's already registered—what HTTP status code would you return?

Although the answer to this is debatable, the widely accepted practice would be to use a 409 Conflict HTTP status code.

It would also be acceptable to return a 422 Unprocessable Entity.

Some may argue that a 400 Bad Request is acceptable, but we discourage this, since conventionally it implies the server did not understand the request, which in this case is not true.