Handling Runtime Errors in JavaScript

Runtime errors can cause unexpected application crashes. JavaScript provides mechanisms to handle errors gracefully and prevent applications from breaking.

1. `try...catch` for Error Handling

The `try...catch` statement allows you to catch and handle errors without stopping the execution of your program.

Example:

```
try {
    let result = 10 / 0;
    console.log(result);
    throw new Error("Something went wrong!");
} catch (error) {
    console.log("Error caught:", error.message);
}
```

2. `throw` for Custom Errors

The `throw` statement allows you to create custom errors with meaningful messages.

Example:

```
function checkAge(age) {
    if (age < 18) {
        throw new Error("You must be 18 or older.");
    }
    console.log("Access granted");
}

try {
    checkAge(16);
} catch (error) {
    console.log("Caught error:", error.message);
}
```

3. `finally` Executes Regardless of Errors

The `finally` block runs whether or not an error occurs, making it useful for cleanup operations.

Example:

```
try {
    console.log("Trying to execute");
    throw new Error("Oops!");
} catch (error) {
    console.log("Caught error:", error.message);
} finally {
    console.log("This will always run");
}
```

Conclusion

Using `try...catch`, `throw`, and `finally` ensures that runtime errors are handled properly, preventing application crashes and improving user experience.