인사말 목차 ~~

우선 자바스크립트가 어떤 언어인지 가볍게 알아보고 가겠습니다. 사실 지금 설명할 내용은 웹 프로그래밍이 어떤 분야인지 한 번 알아만 볼까? 라고 생각하고 검색하거나 강의를 들으면 제일 먼저 나오는 가장 기본적인 분야이기 때문에 아주 가볍게만 설명하고 넘어가겠습니다. 중요한 건 그게 아니니까요. 웹 프로그래밍은 크게 사용자의 눈에 보이는 부분을 개발하는 프론트엔드와, 서버나 데이터베이스 구축 등 눈에 보이지 않는 부분을 개발하는 백엔드로 나뉩니다. 둘 중 프론트엔드를 구축하는 웹퍼블리싱 단계에서는 html, css, 그리고 javascript라는 언어를 사용합니다. 이 셋 중에서 html은 뼈대와 구조를 설계하는 역할입니다. 사이트에 어떤 요소들이 어떤 순서와 내용을 가지고 있을 것인지 넣어주는 역할을 합니다. css는 이 html 파일에 살과 옷을 입혀 꾸며주는 역할을 합니다. 디자인의 영역이죠. 글자 서체는 무엇인지, 크기나 색은 무엇인지 정합니다. 마지막으로 자바스크립트는 인터렉션을 위한 언어입니다. 클릭이나 스크롤 등의 이벤트가 발생할 경우에 어떻게 반응할지를 정하는 역할입니다. 이중에서 저희가 오늘 사용할 언어는 자바스크립트입니다.

이번에는 동기 처리와 비동기 처리가 무엇인지 알아볼 건데요. 동기 처리는 서버에서 요청을 보냈을 때 응답이 돌아와야 다음 동작을 수행할 수 있는 방식을 말합니다. 비동기 처리는 이와 달리 요청을 보냈을 때 응답 상태와 상관 없이 다음 동작을 수행할 수 있는 방식을 말합니다. 쉽게 이해하기 위해 간단한 예시를 들어 설명해보겠습니다. 저희는 이제 정보 전문가가 되어 비밀 요원 3명의 정보를 받아 와야 합니다. 각 요원의 정보를 받아오는 데 1초에서 2초 정도의 시간이 걸린다고 가정해 봅시다. 그렇다면 요원 3명의 정보를 모두 받아오는 데에는 몇 초의 시간이 걸릴까요?

요원 1의 정보를 받은 후 요원 2, 3 순서대로 정보를 요청하고 받아온다면 3초에서 6초 정도의 시간이 걸릴 것입니다. 요원의 수가 많아지면 많아질수록 시간이 오래 걸리게 되겠죠.

그런데 이걸 요청은 셋 다 동시에 보내 놓고 정보를 얻어올 수 있게 한다면 어떨까요? 그럼 아무리 많은 요원의 정보를 알아와야 한다고 하더라도 2초 정도밖에 걸리지 않겠죠. 훨씬 효율적일 것입니다. 이걸 어떻게 코딩할 수 있을까요? 코드로 한 번 알아보겠습니다.

원래 저희가 쓰는 방식의 코드입니다. js 코드인데 문법을 잘 모르실 수 있는데 간단하게 설명드리자면 이 getData 함수는 현재 시간을 받고 계속해서 시간을 확인해 가면서 주어진 시간만큼 지나면 이 멘트를 출력하는 함수입니다. 그리고 저희는 요원 3명의 정보를 받아왔고, 각각 2초, 1.5초, 1초가 걸리나 봅니다. 그렇다면 이 결과가 나오기까지는 대략 4.5초 정도가 걸리겠네요. 이 시간을 여러 컴퓨터에서 연산하듯 동시에 받아오기 위해서는 어떻게 해야 할까요?

이 코드를 보시면 setTimeout이라는 함수를 사용합니다. 이 함수에 들어가는 인자를 보면 콜백 함수가 있고요, 기다릴 시간을 밀리초 단위로 넣어주고, 그리고 콜백 함수에 파라미터로 넣어줄 인자를 맨 뒤에 넣어줍니다. setTimeout이라는 함수는 인자로 들어온 콜백 함수를 예약하기만 하고 바로 return, 그러니까 실행을 종료합니다. 그리고 이 예약된 콜백 함수의 동작은 본래 코드 흐름과는 상관 없이 따로따로 독립적으로 돌아갑니다. 그러니까 지금 setTimeout 함수를 세 번 호출하니까 실행되고 예약한 후 바로 종료되고, 실행되고 예약하고 종료되고, 다시 실행하고 예약하고 종료되는 거죠. 그 다음에 정보 요청을 모두 보냈다는 말을 출력하는 겁니다. 출력된 순서를 보시면 요청을 보냈다는 말이 제일 먼저 출력되고, 3 2 1 순서대로 정보를 받아왔다고 출력되어 있죠? 세 setTimeout 함수가 따로따로 독립적으로 돌아가기 때문에 제일 짧게 걸리는 3번 요원의 정보, 그리고 1.5초짜리 2번 요원, 그리고 제일 긴 2초짜리 1번 요원의 정보를 받아오는 겁니다.

그림으로 보자면 이렇습니다. 거의 동시에 시작해서 정보 요청을 보냈다는 메시지가 출력된 후, 먼저 끝나는 대로 출력이 되는 거죠. 이 결과가 나오기까지는 총 2초 정도만 필요하겠네요. 이런 식으로 독립적으로 돌아가게 하는 작업 방식을 비동기 처리라고 합니다. 이해가 되시나요?

이런 비동기 작업의 단위를 promise라고 합니다. promise는 promise 객체를 선언함으로써 활용할 수 있는데요. 코드를 한 번 보겠습니다. promise 객체를 선언하는 데 여기 포인터 함수가 하나 들어가 있네요. 이 포인터 함수를 executor라고 합니다. executor는 resolve와 reject라는 인자를 받습니다. resolve는 executor 안에 들어가 있는 비동기 작업이 성공했을 때 호출하는 함수입니다. 성공했을 때 호출한다기보다는 resolve가 호출되면 성공했다는 뜻으로 본다고 설명하는 게 맞겠네요. reject는 반대로 호출되었다면 실패했다는 뜻이 되구요. 지금은 if 안에 1을 넣어놔서 무조건 성공하도록 해놨습니다. 이 promise1이라는 객체가 가지고 있는 메서드가 2개 있네요. then과 catch입니다. then은 작업이 성공했을 때, 즉 resolve가 호출되었을 때 어떤 동작을 할 것인지를 지정해줍니다. 지금은 성공한다면 then을 출력하고 실패한다면 catch를 출력해라 라고 코딩되어 있죠? 결과값으로는 then이 나온 것을 확인할 수 있습니다. 그런데 매번 이렇게 new ~~를 써서 선언하기보다는 이 new promise~를 return 해주는 함수를 만들어 사용하기도 합니다. 이런 식으로요.

startAsync 함수가 바로 소환되었을 때 promise 객체를 선언하여 return해주는 함수입니다. 보시면 이번에는 나이를 인자로 받아서 20을 넘으면 resolve를, 넘지 않으면 reject를 호출하도록 했네요. resolve에는 받은 age 변수가 들어 있는 문자열을 인자로 넣어줬고, reject에서는 에러 객체를 인자로 넣어줬네요. 그래서 25를 인자로 첫 promise1이라는 객체를 선언했고, promise1의 then과 catch method를 정의해놨습니다. 두 번째는 15를 인자로 promise2라는 객체를 선언하고 똑같이 했습니다. 결과를 보면 처음에는 25가 20을 넘으니까 resolve, then 메서드가 호출되어 25 success라는 출력이 나왔고 두 번째에는 15는 20보다 작으니까 에러 메시지가 떴네요. promise에는 또 여러 특성들이 있는데 ~~하다는 겁니다. 자세한 설명은 생략하도록 할게요.

그림으로 promise의 구조를 대충 설명해보면 이렇습니다. 반듯한 글씨만 나오다가 갑자기 삐뚤거리는 제 손글씨가 등장했네요. 양해해주셨으면 좋겠습니다. executor가 소환되어 비동기 작업을 하고, reject를 호출하면 실패했다는 뜻으로 catch 메서드를 호출, resolve를 호출하면 성공했다는 뜻으로 then 메서드를 호출합니다. 다들 이해 가시나요?

이 promise를 사용하는 더 쉬운 방법이 바로 async를 활용하는 방법입니다. promise를 이용한 아까 그 코드에 async 키워드를 추가하고, 객체 선언부를 삭제하고, resolve와 reject를 호출하는 것이 아니라 이렇게 return과 throw error를 사용하는 등 기타 문법에 맞춰 조금만 손보면 똑같은 결과가 나옵니다. 여기서 저희가 알아둬야 할 것은 async 함수의 return 값은 promise 객체라는 점입니다. 그것 말고는 다를 바가 없습니다!

그렇다면 await은 무엇일까요? await은 async 함수 내에서만 사용할 수 있는 함수로, promise가 끝날 때까지 기다리는 함수입니다. 못생긴 그림은 놔두고 코드를 먼저 보시면, 제일 먼저 startAsyncJobs라는 함수가 호출됩니다. 그 안에서 await 함수인 setTimeoutPromise 함수가 실행되는데요, 이 함수는 그냥 주어진 시간만큼 기다리는 함수입니다. 1초만큼 기다린 후에야 25를 인수로 하는 promise1 객체를 선언하고, 15를 인수로 하는 promise2 객체를 선언합니다. 이 기다리는 역할을 하는 await 함수는 왜 async 내에서만 사용이 가능할까요? 그건 동기 환경에서는 비동기 작업을 기다리는 게 의미가 없기 때문입니다. 그림처럼 예시를 들어 설명하면, 항구에서 고기잡이 배가 떠난 후에 그것만을 애타게 기다리고 있다면 그건 비동기 작업의 의의를 없애는 일입니다. 그럴 거면 그냥 동기 코드를 사용하면 되겠죠. 하지만 그건 효율이 떨어지기 때문에 배가 떠난 후에 항구 사람들은 항구의 일을 계속 할 것입니다. 뭐 다른 배를 또 보든가 있는 물고기들을 손질하든가 하겠죠. 항구 입장에서는 배에서 동기 작업을 하든 비동기 작업을 하든, 모두 비동기 작업일 겁니다. 그런데 비동기 환경에서 비동기 작업의 결과를 기다리는 것은 의미가 있습니다. 고기잡이 배에서 소형 배를 내보냈는데 그 소형 배가 돌아와야 할 수 있는 작업이 있다면 소형 배가 돌아올 때까지 기다려야겠죠. 마치 비동기 환경에서 동기 코드를 사용하는 것처럼 말이에요. 이처럼 비동기 환경에서 비동기 작업의 결과를 기다리는 걸 위해서 바로 await를 사용하는 겁니다! 이해가 가시나요?

추가로 설명드리고 싶은 부분은 promise.all이라는 겁니다. 이건 여러 비동기 동작을 한 번에 기다리게 하는 키워드입니다. 다시 처음의 예시로 돌아와서, 요원 10명의 나이 정보를 얻어와서 평균을 낸다고 하면 10명의 정보를 모두 얻어야 평균을 계산할 수 있기 때문에 await를 사용하게 될 것입니다. 코드를 보시면 여기서는 그냥 랜덤변수를 이용해서 나이로 받아왔다고 가정했습니다. 나이를 받아올 때는 보시는 것처럼 1초가 걸리네요. ages라는 배열에 얻어 온 나이 정보를 push하고, 평균 나이를 계산해서 출력하는 코드입니다. 결과를 보시면 이렇게 출력됩니다. 그런데 함정이 있습니다! 이 코드의 결과가 나오기까지는 총 10초 정도의 시간이 걸립니다. for문 안에 await가 있기 때문에 동기 코드나 다름없는 작동을 하게 된 것입니다. 이를 막기 위해서 promise.all이라는 키워드를 사용합니다.

promise.all은 promise의 배열을 인자로 받아서 그 안의 모든 비동기 작업들이 성공하면 resolve를 호출하고, 하나라도 실패하면 reject를 호출하는 함수입니다. 코드를 보시면 이번에는 promises라는 배열에 10개의 promise 객체들을 넣고 하나의 새로운 promise 객체를 만들어서 모든 요원의 정보를 받아오는 것에 성공했을 때 평균 나이를 출력하도록 했습니다. 이건 대략 1초 조금 넘는 시간이 걸렸습니다. 이게 바로 저희가 원하던 결과죠?

그렇다면 단순히 promise를 사용하는 것과 async, await를 사용하는 것에는 어떤 차이가 있을까요? 우선 promise를 이용할 때는 catch를 통해 에러 핸들링이 가능하지만 async와 await를 사용할 경우 에러 핸들링 기능이 없어 try-catch문을 사용해여 합니다. 그렇다면 저희는 왜 async와 await를 사용할까요? async와 await를 사용하면 비동기 코드도 저희가 원래 사용하던 동기 코드처럼 읽을 수 있습니다. 코드의 흐름을 이해하기 쉬워진다는 매우 큰 장점이 있죠! then과 catch를 여러 개 연속적으로 사용할 경우 promise에서는 코드도 매우 길어지고 가독성이 떨어집니다. then 지옥에 빠질 수 있죠. 코드가 길어질수록 async와 await를 활용한 코드의 가독성이 훨씬 좋아집니다. 모두 이 세 가지를 적절히 활용하여 해피코딩하도록 합시다.

여기까지 오늘 발표를 바치도록 하겠습니다. 혹시 질문이 있으시다면 편하게 해 주세요! 지금 대답하지 못한다면 알아보고 답변드리도록 하겠습니다 감사합니다.